Frontiers of Computer Science CN 10-1014/TP
DOI 10.1007/s11704-018-7054-8 ISSN 2095-2228
RESEARCH ARTICLE

A Verification Framework for Spatio-Temporal Consistency Language with CCSL as a Specification Language

Yuanrui ZHANG[1], **Frédéric MALLET**[2], **Yixiang CHEN**[1]

1 MoE Engineering Research Center for Software/Hardware Co-design Technology and Application, East China Normal University, Shanghai 200062, China
2 University Nice Sophia Antipolis, I3S, UMR 7271 CNRS, INRIA, 06900 Sophia Antipolis, France

Front. Comput. Sci., **Just Accepted Manuscript** • 10.1007/s11704-018-7054-8
http://journal.hep.com.cn on December 22, 2017

Front.Comput.Sci.
DOI

RESEARCH ARTICLE

A Verification Framework for Spatio-Temporal Consistency Language with CCSL as a Specification Language

Yuanrui Zhang [1], Frédéric Mallet (✉)[2], Yixiang Chen (✉)[1]

1 MoE Engineering Research Center for Software/Hardware Co-design Technology and Application, East China Normal University, Shanghai 200062, China
2 University Nice Sophia Antipolis, I3S, UMR 7271 CNRS, INRIA, 06900 Sophia Antipolis, France

Abstract

The Spatio-Temporal Consistency Language (STeC) is a high-level modeling language that deals natively with spatio-temporal behaviour, i.e., behaviour relating to certain locations and time. Such restriction by both locations and time is of first importance for some types of real-time systems. CCSL is a formal specification language based on logical clocks. It is used to describe some crucial safety properties for real-time systems, due to its powerful expressiveness of logical and chronometric time constraints. We consider a novel verification framework combining STeC and CCSL, with the advantages of addressing spatio-temporal consistency of system behaviour and easily expressing some crucial time constraints. We propose a theory combining these two languages and a method verifying CCSL properties in STeC models. We adopt UPPAAL as the model checking tool and give a simple example to illustrate how to carry out verification in our framework.

Keywords Spatio-temporal consistency, real-time systems, spatio-temporal systems, high-level modelling language, clock constraint specification, model checking, verification framework

1 Introduction

Real-time systems are widespread nowadays and are quickly evolving. To achieve strong results on the correctness of system, it often requires, from the beginning of system design, modeling formalism to describe systems at an abstract level and corresponding verification techniques to reason about abstract system behaviour. In some sorts of real-time systems like mobile distributed systems or Cyber Physical Systems (CPSs), agents' behaviour often relate not only to real time, but also to physical or logical locations. For example in Intelligent Railroad Crossing System (IRCS) (Fig. 8), the 'smart' train should inform the 'smart' gate that it is coming when it arrives at the location $Appr$ at time t. Such constraint — that an event must be triggered at explicit time and location is called 'spatio-temporal consistency' [1]. At high-level of system design, a formalism, where such constraints can be easily expressed, is required to help engineers design more reliable programs when it comes to refinements. Right specification languages are also needed to give an easy expression of some important properties that are of interest.

Spatio-Temporal Consistency Language (STeC) [1, 2] is a modeling formalism proposed for describing spatio-temporal behaviour of real-time systems. It is a process algebra-like modeling language, and looks like an extension of CSP [3] and CCS [4], with location taken as a primitive in its syntax [1]. It aims at modeling system at high level. Not like other process algebras extended with location or time, like timed CSP [5], timed CCS [6], ambient calculus [7], π−calculus [8], etc, STeC addresses a stronger constraint between time and location—spatio-temporal consistency, rather than only considering one of them alone. For example, in timed CCS an event can be triggered at an explicit time, while in ambient calculus an event can be triggered at a specific location (ambience). In STeC, an event can only be triggered at some

Received month dd, yyyy; accepted month dd, yyyy

E-mail: ××××@×××.×××

explicit time and location.

Clock Constraint Specification Language (CCSL) [9] is a specification language based on logical clocks [10]. It was initially proposed as a companion language for the modeling language MARTE [11], and now has been fully developed as an independent language. Compared with traditional specification languages like LTL, CTL, TCTL [12] and PSL [13], logical clock provides an intuitive way to express event sequences and its chronometric attributes. The logical and chronometric constraints between events can be easily expressed by the relationships between logical clocks. Comparing to traditional specification languages, CCSL alleviates the burden for specifying some crucial safety properties, since it provides a library of off-the-shelf often-used property patterns. A comparison of expressiveness between CCSL and PSL is analyzed in [14].

In order to give a verification support for STeC language, and inspired by the MARTE/CCSL framework [15], in this paper we proposed a STeC/CCSL verification framework for modeling and verifying spatio-temporal systems—a special type of real-time systems where agents' behaviour is related not only to real time, but also to locations. Compared to other frameworks for real-time systems, STeC/CCSL framework focuses on capturing system behaviour at high level, with the advantage of addressing the spatio-temporal consistency of system behaviour at the syntax level. And it supports specifying and verifying logical and chronometrical timing constraints(, and possibly some carefully selected spatial constraints in the future) in CCSL style, which could be complex or difficult to be expressed by other traditional specification languages like LTL, CTL.

In STeC/CCSL framework, in order to connect STeC and CCSL, we propose a linking theory and a model checking framework to verify CCSL specifications. We formalize some concepts in CCSL and build an observational mapping between CCSL clocks and STeC events. To carry out model checking, we propose theory and algorithms to translate STeC and CCSL into their equivalent Timed Automata (TA). We make a general analysis for the computation complexity of the whole verification process, and show that our framework runs 'nearly as fast as' the traditional model checking process, despite the translation process. At last, in order to show that our proposed verification framework is applicable, we propose the translation into UPPAAL [16] TA, a model checking tool widely used in academia and industry.

Technically, our work can be seen as a combination and extension of [17, 18], where the verification aspect of CCSL and STeC has been explored separately. In [18], we introduced TA semantics of STeC. The translation from STeC into TA was given inductively based on the syntax structure of STeC. In this paper we take a more general approach by dividing STeC configurations into 'regions'. In [17], a translation strategy was proposed to encode CCSL into TA and two examples were given to illustrate the model checking scheme in UPPAAL. There the TA of CCSL is based on untimed synchronous model cLTS [19], which is not suitable for our framework since STeC is an asynchronous language. We make some improvements by translating into asynchronous TA models. And we give a full translation from CCSL into UPPAAL TA. Our proposed model checking framework is partially based on the idea of the verification scheme in [17], but is quite different as we consider the combination with STeC and the CCSL generators.

This paper is organized as follows: Section 2 introduces some backgrounds about STeC, CCSL and TA. In Section 3, we introduce a spatio-temporal system—Intelligent Railroad Crossing System (IRCS) as an example and model it using STeC. In Section 4 we present the main contribution — we propose the linking theory that connects STeC and CCSL and the model checking framework. In Section 5, we carry out model checking in UPPAAL, based on the translation into TA. In Section 6 we verify three IRCS safety properties in UPPAAL as an example. Section 7 concludes our work and discusses possible future works. Section 8 compares to some similar verification frameworks proposed in recent years.

2 Preliminaries

Given any two sets A, B, $A - B = \{a \mid a \in A \wedge a \notin B\}$ is the difference set of A and B . $A \uplus B = \{(a, i) \mid (a \in A \wedge i = 0) \vee (a \in B \wedge i = 1)\}$ is the disjoint union of A and B. $\mathbb{R}^+$ denotes the set of non-negative real numbers, $\mathbb{N}^+$ denotes the set of natural numbers (excluding 0).

Given a function $f : A \to B$, $dom(f) = \{a \mid a \in A \wedge f(a) \in B\}$ denotes its domain, and $cod(f) = \{b \mid b \in B \wedge \exists a.(f(a) = b)\}$ denotes its codomain. A function is partial iff $dom(f) \subset A$. A function $f : \mathbb{N}^+ \to B$ is down-closed iff $i \in dom(f)$ implies $\forall j < i.(j \in dom(f))$. A function is finite iff $dom(f)$ is a finite set.

A partial order relation $\leq \subseteq A \times A$ is a reflexive, antisymmetric and transitive relation defined on A. A strict partial order relation $< \subseteq A \times A$ is transitive, but neither reflexive or antisymmetric. A total (strict) partial order $\leq$ ($<$) is a (strict) partial order such that for all $x, y \in A$, $x \leq y$ ($x < y$) or $y \leq x$ ($y < x$) holds.

In the remaining of this section, we introduce the basic concepts of STeC, CCSL and TA.

2.1 Introduction of STeC

We restate the syntax and operational semantics of STeC in this section. The version of STeC we give is based on [1], but differs in some details. The main difference is that we distinguish two types of parallel compositions between pocesses: the interleaving '$\|$' between processes in agents, and the concurrency '$\bowtie$' between agents. In operational semantics, we split original STeC transitions into two kinds: time-only transitions and process transitions. This would ease the way to translate STeC into TA (Section 4.6.2). The expressiveness of this version remains the same as in [1].

2.1.1 Syntax

An agent Ag in STeC is defined based on the following rules in Backus-Naur form:

$$\begin{aligned}
A &::= \mathbf{Send}^{C}_{(l,t)}(m) \mid \mathbf{Get}^{C}_{(l,t)}(m)\\
B &::= \alpha_{(l,t)}(l',\delta) \mid \beta_{(l,t)}(\delta)\\
AT &::= E \mid \mathbf{Stop}_{(l,t)} \mid \mathbf{Skip}_{(l,t)} \mid A \mid B\\
Ag &::= AT \mid Ag; Ag \mid Ag[]Ag \mid Ag \parallel Ag \mid\\
&\quad\;\; Ag \trianglerighteq_{\delta} Ag \mid Ag \trianglerighteq ([]_{i\in I} Ag_i \rightarrow Ag_i)
\end{aligned}$$

A system P can be an agent, or the composition of several agents running in parallel:

$$P \;::= Ag \mid P \bowtie P$$

In STeC, symbol P, Q, R ranges over STeC processes. The set of event names **Alp** is ranged over by a, b, c. The set of location names **Loc** is ranged over by l. The set of message names **Msg** is ranged over by m. Symbol $\mathbb{A}_{STeC}$, $\mathbb{B}_{STeC}$, $\mathbb{AT}_{STeC}$ denote the set of processes of type A, B and AT respectively, and symbol $\mathbb{P}_{STeC}$ denotes the set of all processes. Symbol t and δ denote the time that ranges over $\mathbb{R}^{+}$.

We now give an intuitive explanation of each STeC sentance.

In atomic processes AT, processes of type A are communicating channels between agents. $\mathbf{Send}^{C}_{(l,t)}(m)$ means that an agent sends message m through channel C. $\mathbf{Get}^{C}_{(l,t)}(m)$ means that an agent receives message m through channel C. $\alpha_{(l,t)}(l',\delta)$ means that an event starts at location l and time t, and consumes δ time to terminate. By executing it, the agent moves from location l to l'. $\beta_{(l,t)}(\delta)$ means that an event starts at location l and time t, and takes δ time to terminate. During its execution the agent stays in l. $\mathbf{Stop}_{(l,t)}$ means that an agent at (l,t) does not terminate, it does nothing but consumes time. $\mathbf{Skip}_{(l,t)}$ means that the program terminates successfully. E indicates the ending of a process, it does nothing and does not consume time.

In an agent Ag, ';' is the sequence operator, $P; Q$ means that the process first behaves like P, then behaves like Q. '[]' is the nondeterministic choice, $P[]Q$ means that either P proceeds or Q. '$\|$' is the interleave between processes in an agent. $P \parallel Q$ means that agent P and Q proceed in parallel without communications. '$\trianglerighteq$' is the interrupt operator. $P \trianglerighteq_{\delta} Q$ behaves as P for up to δ time and then is interrupted by Q. $P \trianglerighteq ([]_{i\in I} A_i \rightarrow P_i)$ initially behaves as P and is interrupted by event $A_i \in \mathbb{A}_{STeC}$ and then behaves like P_i. I is a finite index set. If several A_i occurs simultaneously, the choice '[]' turns out to be a nondeterministic choice. $P \bowtie Q$ means that agent P and Q are running in parallel with communications.

2.1.2 Operational Semantics

An environment in STeC is a pair (l, t). $\mathcal{E}$ denotes the set of environments, $\mathcal{E} \subseteq \mathbf{Loc}\times\mathbb{R}^{+}$. A configuration of process P is a triple $\langle P, l, t\rangle$, where $(l, t) \in \mathcal{E}$. We use **Stconf** to denote the set of configurations. Transition relation $\rightarrow \subseteq \mathbf{Stconf}\times\mathbf{Stconf}$ is defined as: $\langle P, l, t\rangle \rightarrow \langle P', l', t'\rangle$ iff $(\langle P', l', t'\rangle, \langle P, l, t\rangle) \in\rightarrow$.

There are two types of transitions in the operational semantics of STeC: time-only transitions and process transitions. During time-only transitions, the process remains unchanged while time progresses. During process transitions, the process emits an event instantly. The transition rules for atomic processes are given in Table 1. Time-only transitions are denoted as $\xrightarrow{\epsilon}$. Symbol $\epsilon \notin \mathbf{Alp}$ means 'there is no events triggered during transitions'. Process transitions are denoted by $\xrightarrow{b}$, with $b \in \mathbf{Alp}$. We follow the convention in automata theory that using $C?m/C!m$ to express input/output communication events. Note that there is no transition rules for process E, because E indicates the ending of a process. So semantically, we can take any process-

1.	$\dfrac{}{\langle \mathbf{Send}^{C}_{(l,t)}(m),l,t\rangle \xrightarrow{C!m} \langle E,l,t\rangle}$	5.	$\dfrac{\forall d\in\mathbb{R}^{+}, t'\geq t, t'+d<t+\delta}{\langle \beta_{(l,t)}(\delta),l,t'\rangle \xrightarrow{\epsilon} \langle \beta_{(l,t)}(\delta),l,t'+d\rangle}$
2.	$\dfrac{}{\langle \mathbf{Get}^{C}_{(l,t)}(m),l,t\rangle \xrightarrow{C?m} \langle E,l,t\rangle}$	6.	$\dfrac{}{\langle \beta_{(l,t)}(\delta),l,t+\delta\rangle \xrightarrow{\beta} \langle E,l,t+\delta\rangle}$
3.	$\dfrac{\forall d\in\mathbb{R}^{+}, t'\geq t, t'+d<t+\delta}{\langle \alpha_{(l,t)}(l',\delta),l,t'\rangle \xrightarrow{\epsilon} \langle \alpha_{(l,t)}(l',\delta),l,t'+d\rangle}$	7.	$\dfrac{\forall d,t'\in\mathbb{R}^{+}, t'\geq t}{\langle \mathbf{Stop}_{(l,t)},l,t'\rangle \xrightarrow{\epsilon} \langle \mathbf{Stop}_{(l,t)},l,t'+d\rangle}$
4.	$\dfrac{}{\langle \alpha_{(l,t)}(l',\delta),l,t+\delta\rangle \xrightarrow{\alpha} \langle E,l',t+\delta\rangle}$	8.	$\dfrac{}{\langle \mathbf{Skip}_{(l,t)},l,t\rangle \xrightarrow{\mathbf{Skip}} \langle E,l,t\rangle}$

Table 1: Transition rules for atomic processes

1. $$\frac{\langle P,l,t\rangle \xrightarrow{\mu} \langle P',l',t'\rangle,\ \mu\neq\mathbf{Skip}}{\langle P;Q,l,t\rangle \xrightarrow{\mu} \langle P';Q,l',t'\rangle} \quad \frac{\langle P,l,t\rangle \xrightarrow{\mathbf{Skip}} \langle E,l',t'\rangle}{\langle P;Q,l,t\rangle \xrightarrow{\mathbf{Skip}} \langle Q,l',t'\rangle}$$

2. $$\frac{\langle P,l,t\rangle \xrightarrow{\mu} \langle P',l',t'\rangle}{\langle P[]Q,l,t\rangle \xrightarrow{\mu} \langle P',l',t'\rangle} \quad \frac{\langle Q,l,t\rangle \xrightarrow{\mu} \langle Q',l',t'\rangle}{\langle P[]Q,l,t\rangle \xrightarrow{\mu} \langle Q',l',t'\rangle}$$

3. $$\frac{\langle P,l_1,t\rangle \xrightarrow{b} \langle P',l'_1,t\rangle,}{\langle P\|Q,\langle l_1,l_2\rangle,t\rangle \xrightarrow{b} \langle P'\|Q,\langle l'_1,l_2\rangle,t\rangle} \quad \frac{\langle Q,l_2,t\rangle \xrightarrow{b} \langle Q',l'_2,t\rangle}{\langle P\|Q,\langle l_1,l_2\rangle,t\rangle \xrightarrow{b} \langle P\|Q',\langle l_1,l'_2\rangle,t\rangle}$$
$$\frac{\langle P,l_1,t\rangle \xrightarrow{\epsilon} \langle P,l_1,t'\rangle,\langle Q,l_2,t\rangle \xrightarrow{\epsilon} \langle Q,l_2,t'\rangle}{\langle P\|Q,\langle l_1,l_2\rangle,t\rangle \xrightarrow{\epsilon} \langle P\|Q,\langle l_1,l_2\rangle,t'\rangle}$$

4. $$\frac{\langle P,l,t\rangle \xrightarrow{\mu} \langle P',l',t'\rangle,t'-t=\delta}{\langle P\unrhd_{\delta}Q,l,t\rangle \xrightarrow{\epsilon} \langle Q,l',t'\rangle} \quad \frac{\langle P,l,t\rangle \xrightarrow{\mu} \langle P',l',t'\rangle,t'-t<\delta}{\langle P\unrhd_{\delta}Q,l,t\rangle \xrightarrow{\mu} \langle P'\unrhd_{\delta-(t'-t)}Q,l',t'\rangle}$$

5. $$\frac{\langle A_i,l_i,t_i\rangle \xrightarrow{A_i} \langle E,l_i,t_i\rangle,\ for\ i\in I,A_i\in\{C_{A_i}!m,C_{A_i}?m\}}{\langle P\unrhd([]_{i\in I}A_i\rightarrow P_i),l_i,t_i\rangle \xrightarrow{A_i} \langle P_i,l_i,t_i\rangle}$$
$$\frac{\langle P,l,t\rangle \xrightarrow{\mu} \langle P',l',t'\rangle,\mu\notin\{A_i\}_{i\in I}}{\langle P\unrhd([]_{i\in I}A_i\rightarrow P_i),l,t\rangle \xrightarrow{\mu} \langle P'\unrhd([]_{i\in I}A_i\rightarrow P_i),l',t'\rangle}$$

6. $$\frac{\langle P,l_1,t\rangle \xrightarrow{C?m} \langle P',l_1,t\rangle,\langle Q,(l_2,t,\sigma_2)\rangle \xrightarrow{C!m} \langle Q',l_2,t\rangle}{\langle P\bowtie Q,\langle l_1,l_2\rangle,t\rangle \xrightarrow{C.m} \langle P'\bowtie Q',\langle l_1,l_2\rangle,t\rangle}$$
$$\frac{\langle P,l_1,t\rangle \xrightarrow{C!m} \langle P',l_1,t\rangle,\langle Q,(l_2,t,\sigma_2)\rangle \xrightarrow{C?m} \langle Q',l_2,t\rangle}{\langle P\bowtie Q,\langle l_1,l_2\rangle,t\rangle \xrightarrow{C.m} \langle P'\bowtie Q',\langle l_1,l_2\rangle,t\rangle}$$
$$\frac{\langle P,l_1,t\rangle \xrightarrow{b} \langle P',l'_1,t\rangle,b\notin\{C!m,C?m\}}{\langle P\bowtie Q,\langle l_1,l_2\rangle,t\rangle \xrightarrow{b} \langle P'\bowtie Q,\langle l'_1,l_2\rangle,t\rangle} \quad \frac{\langle Q,l_2,t\rangle \xrightarrow{b} \langle Q',l'_2,t\rangle,b\notin\{C!m,C?m\}}{\langle P\bowtie Q,\langle l_1,l_2\rangle,t\rangle \xrightarrow{b} \langle P\bowtie Q',\langle l_1,l'_2\rangle,t\rangle}$$
$$\frac{\langle P,l_1,t\rangle \xrightarrow{\epsilon} \langle P,l_1,t'\rangle,\langle Q,l_2,t\rangle \xrightarrow{\epsilon} \langle Q,l_2,t'\rangle}{\langle P\bowtie Q,\langle l_1,l_2\rangle,t\rangle \xrightarrow{\epsilon} \langle P\bowtie Q,\langle l_1,l_2\rangle,t'\rangle}$$

Table 2: Transition rules for compositional processes

es for example $E;P$, $E \parallel E$, $E[]E$, ... which does not proceed as the equivalent processes of process E, denoted by $E;P = E \parallel E = E[]E = ... = E$.

We define $\rightarrow^*$ as the transitive closure of $\rightarrow$. For any $A \subseteq \mathbf{Alp}$, $\langle P, l, t\rangle \xrightarrow{A}{}^{*} \langle P_n, l_n, t_n\rangle$ means $\langle P, l, t\rangle \xrightarrow{\mu_1} \langle P_1, l_1, t_1\rangle \xrightarrow{\mu_2} ... \xrightarrow{\mu_n} \langle P_n, l_n, t_n\rangle$ where $n \geq 0$, $\mu_i \in \mathbf{Alp} \cup \{\epsilon\}$, $A = \{a \mid a \in \{\mu_0, \mu_1, ..., \mu_n\} - \{\epsilon\}\}$. We write $\langle P, l, t\rangle \xrightarrow{\mu_1}\xrightarrow{\mu_2} \langle P', l', t'\rangle$ to mean that there exists a $\langle P'', l'', t''\rangle$ such that $\langle P, l, t\rangle \xrightarrow{\mu_1} \langle P'', l'', t''\rangle \xrightarrow{\mu_2} \langle P', l', t'\rangle$.

Table 2 lists the transition rules for compositional processes, where $\mu \in \mathbf{Alp} \cup \{\epsilon\}$. Rule 2 indicates that the choice operator [] in STeC is an internal choice. In Rule 3, $\langle l_1, l_2\rangle$ is the location of process $P \parallel Q$, indicating that P is at location l_1, and Q is at location l_2. In Rule 6, if pair $C!m$, $C?m$ of communication events are matched, the whole process emits an event $C.m$ indicating the communication between two agents. The last three subrules say that if events are not communicating events, $P \bowtie Q$ just behaves like $P \parallel Q$.

For any process $P \in \mathbb{P}_{STeC}$, we use $\mathbf{At}(P)$ to denote the set of events that P emits during its execution. For example, $\mathbf{At}(a_{(l,t)}(l_1, \delta_1); \mathbf{Send}^{C}_{(l_1,t+\delta_1)}(m); b_{(l_1,t+\delta_1)}(l_2, \delta_2)) = \{a, C!m, b\}$. Sometimes, we simply write $\mathbf{At}(b) = alp$ if $\mathbf{At}(b)$ contains only one element.

2.2 Introduction of CCSL

In this section we restate the basic definitions of clock, clock constraint, and clock specification. The version we propose is based on [9, 17, 19–21], and the definitions follow the style of [9]. We distinguish the definition of '*Delay On*' for discrete and dense clocks, since we adopt different translating approaches for two different types of clocks (see Table 3).

2.2.1 Clock

A CCSL clock consists of an ordered sequence of ticks (also called 'instants'). Each instant can be assigned to a name by a labelling function. Formally, a clock is a tuple $\langle \mathcal{I}, <, \mathcal{D}, \lambda\rangle$, where

1) $\mathcal{I}$ is a set of instants (possibly infinite).
2) $<$ is a strict total partial order relation on $\mathcal{I}$.
3) $\mathcal{D}$ is a set of labels.
4) $\lambda : \mathcal{I} \rightarrow \mathcal{D}$ is a labelling function, assigning each instant of clock with a name.

An ideal physical clock $IdealClk = \langle \mathcal{I}_0, <_0, \mathcal{D}_0, \lambda_0\rangle$ is the only dense clock discussed in this paper. $\lambda_0 : \mathcal{I}_0 \rightarrow \mathcal{D}_0$ is a bijection and $\mathcal{D}_0 = \mathbb{R}^+$.

In a discrete-time clock $\mathcal{I}$ can be indexed by natural numbers in a way that respects the ordering on $\mathcal{I}$. A function $idx : \mathcal{I} \rightarrow \mathbb{N}^+$ is defined as: $\forall i \in \mathcal{I}, idx(i) = k$ iff i is the k^{th} instant in $\mathcal{I}$. Let i_0 be the first element in $\mathcal{I}$, i.e., for any $x \in \mathcal{I}$, there is $i_0 < x \vee x = i_0$. For a discrete clock, $P(i)$ (with i not the first element in $\mathcal{I}$) is the unique immediate predecessor of i. We have: $idx(P(i)) = idx(i) - 1$. $S(i)$ is the unique immediate successor of i. We have: $idx(S(i)) = idx(i) + 1$. Let $c = \langle \mathcal{I}, <, \mathcal{D}, \lambda\rangle$ be a CCSL clock. $c[k]$ denotes the k^{th} element in $\mathcal{I}$, we have $k = idx(c[k])$. Without special mentioning all clocks appearing in this paper are discrete clocks.

For convenience, we define projections of items for a clock $c = \langle \mathcal{I}, <, \mathcal{D}, \lambda\rangle$: $\pi_{\mathcal{I}}(c) = \mathcal{I}$, $\pi_{<}(c) =<$, $\pi_{\mathcal{D}}(c) = \mathcal{D}$, and $\pi_{\lambda}(c) = \lambda$.

2.2.2 Primitive Clock Constraints

There are different versions of constraints given in [9, 17, 19–21]. We only introduce a subset of clock constraints which are relevant to the time constraints discussed in this paper. Still, our theory proposed in this paper can be easily extended to the rest of them.

We first define 'time structure' where instants from different clocks can be compared [9]. A time structure is a tuple $\langle \mathcal{I}, <, \equiv, H\rangle$, where $\mathcal{I}$ is a set of instants of all clocks, $<$ is a strictly partial order relation between instants, and $\equiv$ is a coincidence relation, indicating the simultaneous occurrence of instants in different clocks, H is a set of constraint mappings in $\mathcal{I}$ (defined explicitly later). From $<$ we derive a partial order relation $\leq := < \cup \equiv$, it means that $a \leq b$ iff $a < b$ or $a \equiv b$.

Let C be a set of clocks, a time structure $\langle \mathcal{I}, <, \equiv, H\rangle$ (over C) is well-structured if it satisfies the following properties:

1) $\leq$ is a partial order relation in TS.
2) $\mathcal{I} = \bigcup_{c \in C} \mathcal{I}_c$, i.e., $\mathcal{I}$ contains instants of all clocks in C.
3) $\forall c_a = \langle \mathcal{I}_a, <_a, \mathcal{D}_a, \lambda_a\rangle \in C$, $\forall i, j \in \mathcal{I}_a$, if $i <_a j$, then $i < j$, i.e., partial order $<$ must be consistent with $<_a$ in each clock $c_a \in C$.
4) $\forall c_a = \langle \mathcal{I}_a, <_a, \mathcal{D}_a, \lambda_a\rangle$, $c_b = \langle \mathcal{I}_b, <_b, \mathcal{D}_b, \lambda_b\rangle \in C$, $\forall i_a, j_a \in \mathcal{I}_a, i_b, j_b \in \mathcal{I}_b$, if $i_a \equiv i_b \wedge j_a \equiv j_b \wedge i_a <_a j_a$, then $i_b <_b j_b$, i.e., $\equiv$ preserves the order in each clock.

Clock constraints is derived by defining different sets of well-structured time structures. Let $c_a = \langle \mathcal{I}_a, <_a, \mathcal{D}_a, \lambda_a\rangle$, $c_b = \langle \mathcal{I}_b, <_b, \mathcal{D}_b, \lambda_b\rangle$, $c_d = \langle \mathcal{I}_d, <_d, \mathcal{D}_d, \lambda_d\rangle$ be any three clocks. The definitions of clock constraints are given as follows:

Sub Clock — constraint $Cn := c_a \subseteq c_b$ describes that clock c_a can tick only if clock c_b ticks. Formally it is defined by a (possibly infinite) set of time structures $\mathcal{TS} = \{TS_i\}_{i \in I}$ (I is a countable index set). Each $TS_i = \langle \mathcal{I}_i, <_i, \equiv_i, \{h_i\}\rangle$ is well structured and $\mathcal{I}_i = \mathcal{I}_a \cup \mathcal{I}_b$. The constraint mapping $h_i : \mathcal{I}_a \to \mathcal{I}_b$ satisfies: 1) h_i is injective. 2) $\forall x \in \mathcal{I}_a.(x \equiv_i h(x))$.

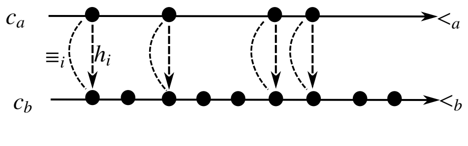


Fig. 1: A time structure of $c_a \subseteq c_b$

Fig. 1 shows a time structure for the constraint $c_a \subseteq c_b$, where only finite number of ticks of clocks are drawn (the same for all figures below). Each dot lying on a long arrow indicates a tick of corresponding clock. The long arrow itself means the order ($<_a$, $<_b$) defined in each clock. The dashed arrow indicates the mapping h, and the dash line represents the 'inter-clock' relation $\equiv_i$, which means that two ticks occur simultaneously.

Note that usually there is more than one time structures for a given constraint, see Fig. 2 for example as another time structure for $c_a \subseteq c_b$ (with two more ticks of c_b ahead of c_a). Each time structure indicates a possible behaviour of how clocks should tick to satisfy the constraint.

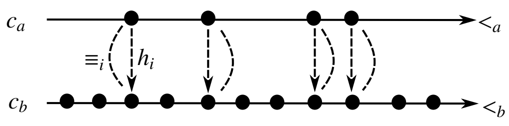


Fig. 2: Another time structure of $c_a \subseteq c_b$

Strict Precedence — the constraint $Cn := c_a \prec c_b$ describes that clock c_a always ticks before clock c_b. It is defined over a set of well-structured time structures $\mathcal{TS} = \{TS_i\}_{i \in I}$. In each $TS_i = \langle \mathcal{I}_i, <_i, \equiv_i, \{h_i\}\rangle$ where $\mathcal{I}_i = \mathcal{I}_a \cup \mathcal{I}_b$, $h_i : \mathcal{I}_b \to \mathcal{I}_a$ satisfies:

1) h_i is injective.
2) $\forall x, y \in \mathcal{I}_b.(x <_b y \Rightarrow h(x) <_a h(y))$.
3) $\forall x \in \mathcal{I}_b.(h(x) <_i x)$.

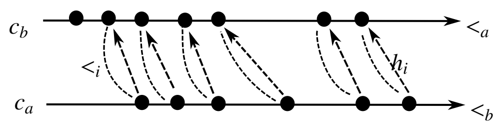


Fig. 3: A time structure of $c_a \prec c_b$

Fig. 3 gives a time structure of $c_a \prec c_b$.

Precedence — the constraint $Cn := c_a \preceq c_b$ describes that clock c_a always ticks 'not-after' clock c_b. In each $TS_i = \langle \mathcal{I}_i, <_i, \equiv_i, \{h_i\}\rangle \in \mathcal{TS}$ where $\mathcal{I}_i = \mathcal{I}_a \cup \mathcal{I}_b$, $h_i : \mathcal{I}_b \to \mathcal{I}_a$ satisfies:

1) h_i is injective.
2) $\forall x, y \in \mathcal{I}_b.(x <_b y \Rightarrow h(x) <_a h(y))$.
3) $\forall x \in \mathcal{I}_b.(h(x) \leq_i x)$.

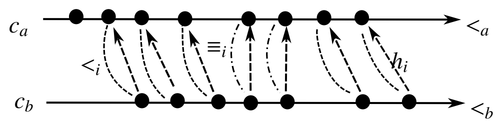


Fig. 4: A time structure of $c_a \preceq c_b$

Fig. 4 gives a time structure of $c_a \preceq c_b$, where the dash-and-dot line indicates the $\equiv_i$ relations between clocks.

Alternation — the constraint $Cn := c_a\ alternatesWith\ c_b$ describes that two clocks ticks alternately. It is defined over $\mathcal{TS}$. In each $TS_i = \langle \mathcal{I}_i, <_i, \equiv_i, \{h_i\}\rangle \in \mathcal{TS}$ where $\mathcal{I}_i = \mathcal{I}_a \cup \mathcal{I}_b$, $h_i : \mathcal{I}_a \to \mathcal{I}_b$ satisfies:

1) h_i is a bijection.
2) $\forall x \in \mathcal{I}_a.(x <_i h(x) \wedge h(x) <_i S(x))$.
3) $\forall x, y \in \mathcal{I}_a.(x <_a y \Rightarrow h(x) <_b h(y))$.

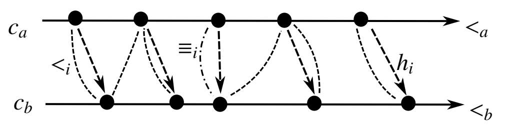


Fig. 5: A time structure of c_a *alternateWith* c_b

Fig. 5 gives an example of the time structure.

Delay On — the constraint $Cn := c_b = c_a$ *delay n on* c_d, where $n \in \mathbb{N}$, c_d is a discrete clock, means that clock c_b is delayed n ticks of clock c_d for c_a. In other words, if c_a ticks, then c_b ticks synchronously with the nth following ticks of c_d. Formally, in each $TS_i = \langle \mathcal{I}_i, <_i, \equiv_i, \{h_i^1, h_i^2\}\rangle \in \mathcal{TS}$ where $\mathcal{I}_i = \mathcal{I}_a \cup \mathcal{I}_b \cup \mathcal{I}_d$, $h_i^1 : \mathcal{I}_b \to \mathcal{I}_a$ and $h_i^2 : \mathcal{I}_b \to \mathcal{I}_d$ satisfy:

1) h_i^1, h_i^2 are both injective.
2) $\forall x \in \mathcal{I}_b.(x \equiv_i h_i^2(x) \wedge h_i^1(x) \leq_i P^{n-1}(h_i^2(x)) \wedge P^n(h_i^2(x)) <_i h_i^1(x))$.

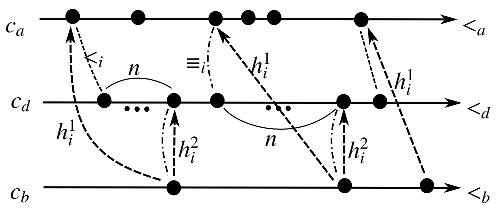


Fig. 6: A time structure of $c_b = c_a$ *delay n on* c_d

Fig. 6 gives an example of the time structure, where n indicates that there are n ticks between those two ticks (including them).

Delay On (for Dense Clock) — the constraint $Cn := c_b = c_a$ *delay r on IdealClk* where $r \in \mathbb{R}^+$ is defined over $\mathcal{TS}$. In each $TS_i = \langle \mathcal{I}_i, <_i, \equiv_i, \{h_i^1, h_i^2\}\rangle \in \mathcal{TS}$ where $\mathcal{I}_i = \mathcal{I}_a \cup \mathcal{I}_b \cup \mathcal{I}_0$, $h_i^1 : \mathcal{I}_b \to \mathcal{I}_a$ and $h_i^2 : \mathcal{I}_b \to \mathcal{I}_0$ satisfies:

1) h_i^1, h_i^2 are both injective.
2) $\forall x \in \mathcal{I}_b, \exists k \in \mathcal{I}_0.(x \equiv_i h_i^2(x) \wedge h_i^1(x) \equiv_i k \wedge \lambda_0(h_i^2(x)) - \lambda_0(k) = r)$.

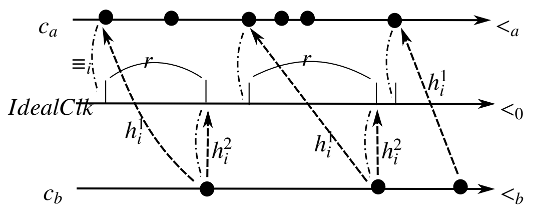


Fig. 7: A time structure of $c_b = c_a$ *delay r on IdealClk*

Fig. 7 gives an example of the time structure, where we use a short line to indicate a tick of *IdeaClk* to stress that it is a dense clock (i.e., it ticks everywhere on the long arrow $<_0$).

In this paper we use Cn to range over all clock constraints. We often write $\mathcal{TS}_{Cn}$ as the corresponding set of time structures of a given constraint Cn. We use $\mathbb{C}_{CCSL}$ to denote the set of all clock constraints.

2.2.3 CCSL Specification, Safe Specification

A CCSL specification $spec \subseteq \mathbb{C}_{CCSL}$ is a finite set of constraints. We use $spec$ to range over all specifications. Sometimes we also call a specification 'a set of constraints' or just 'constraints' instead.

A CCSL specification can be equivalently understood as a type of Labelled Transition System (LTS) [19]. Some of primitive constraints, called unsafe constraints, correspond to LTSs with infinite states (e.g., $c_a \prec c_b$ in Table 3). Such LTSs are not allowed when we analyze the model checking aspects of CCSL. However, a specification containing one or several unsafe constraints does not have to be unsafe. An example is the specification of **Property 1** in Section 6 (see Fig. 19). If a specification corresponds to an LTS with finite states, we call it a safe specification.

When dealing with model checking aspects of CCSL, we always consider safe specifications. [22] introduces an efficient algorithm to check whether a specification is safe or not using directed graph.

2.3 Introduction to Timed Automata [23]

2.3.1 Definition

Let C be a finite set of non-negative real-valued variables—clocks. Guard expressions is defined by the grammar $g := true \mid false \mid c \star n \mid g \wedge g$ where $c \in C$, $n \in \mathbb{N}$ ($\mathbb{N} = \mathbb{N}^+ \cup \{0\}$) and $\star \in \{<, \leq, >, \geq, =\}$. $G(C)$ denotes the set of guards related to C. A timed automata is a tuple $A = (Q, \Sigma, C, i, Ed, I, AP, L, F)$, where:

1) Q is a set of states.
2) Σ is a set alphabets.
3) C is a finite set of clocks.
4) $i \in Q$ is the initial location.
5) $Ed \subseteq Q \times \Sigma \times G(C) \times 2^C \times Q$ is a transition.
6) $I : Q \to G(C)$ is an invariant-assignment function.
7) AP is a set of atomic propositions.
8) $L : Q \to AP$ is a labeling function.
9) $F \subseteq Q$ is a set of accepting states.

2.3.2 States and Transitions

A configuration in TA is a pair $\langle q, v\rangle$ where $q \in Q$ is a state and $v : C \to \mathbb{R}^+$ is a valuation of clocks. For $r \in \mathbb{R}^+$, we define $v + r$ as: for each clock $x \in C$, $(v + r)(x) = v(x) + r$. If $Y \subseteq C$ then a valuation $v[Y := 0]$ is such that for each clock $x \in C - Y$, $v[Y := 0](x) = v(x)$ and for each clock $x \in Y$, $v[Y := 0](x) = 0$. A satisfaction relation $v \models g$ holds iff the valuation v makes guard g true. Given a TA $A = (Q, \Sigma, C, i, Ed, I, AP, L, F)$, the transition function between states is a relation $\rightarrow \subseteq (Q \times \mathbb{R}^{+C}) \times (Q \times \mathbb{R}^{+C})$, defined as:

$$\rightarrow (q, v) = \begin{cases} \langle q, v + \alpha\rangle, & \text{if } \forall \beta \leq \alpha \in \mathbb{R}^+, v + \beta \models I(q) \\ \langle q', v'\rangle, & \text{if } \langle q, a, g, Y, q'\rangle \in Ed, a \in \Sigma, \\ & v \models g, v' = v[Y := 0] \end{cases}$$

The first transition above is the time-only transition, we write it as $\langle q, v\rangle \xrightarrow{\epsilon} \langle q, v + \alpha\rangle$. The second transition is written as $\langle q, v\rangle \xrightarrow{a} \langle q', v'\rangle$. $\rightarrow^*$ is the transitive closure of $\rightarrow$, defined as: $\langle q, v\rangle \xrightarrow{A}{}^* \langle q_n, v_n\rangle$ iff $\langle q, v\rangle \xrightarrow{\mu_1} \langle q_1, v_1\rangle \xrightarrow{\mu_2} \langle q_2, v_2\rangle \xrightarrow{\mu_3} \ldots \xrightarrow{\mu_{n-1}} \langle q_{n-1}, v_{n-1}\rangle \xrightarrow{\mu_n} \langle q_n, v_n\rangle$ where $n \geq 0$, $\mu_i \in \Sigma \cup \{\epsilon\}$, $i = 1, 2, ..., n$, $A = \{a \mid a \in \{\mu_1, ..., \mu_n\} - \{\epsilon\}\}$. $\langle q, v\rangle \xrightarrow{\mu_1}\xrightarrow{\mu_2} \langle q', v'\rangle$ means that there exists a $\langle q'', v''\rangle$ such that $\langle q, v\rangle \xrightarrow{a} \langle q'', v''\rangle \xrightarrow{b} \langle q', v'\rangle$.

In a TA A, the initial configuration, denoted as $ic_A = \langle i, v^i_A\rangle$ where i is the initial state, satisfies that $\nexists\langle q, v\rangle.(\langle q, v\rangle \xrightarrow{B}{}^* \langle i, v^i_A\rangle)$. The set of all its configurations of A, denoted as $\mathbf{Conf}_{TA}(A)$, is given by $\mathbf{Conf}_{TA}(A) = \{\langle q, v\rangle \mid ic_A \xrightarrow{B}{}^* \langle q, v\rangle\}$.

2.3.3 TA Traces

A time word is a pair $\langle a, t\rangle$ where $a \in \Sigma$ and $t \in \mathbb{R}^+$. We use $\mathbb{E}_{TA}$ to denote the set of all time words. A finite TA trace $tr : \mathbb{N}^+ \rightarrow \mathbb{E}_{TA}$ is a partial, down-closed, finite function. A trace $tr = \langle a_1, t_1\rangle\langle a_2, t_2\rangle...\langle a_n, t_n\rangle$ is in TA A iff there are transitions $\langle i, v^i_A\rangle \xrightarrow{\emptyset}{}^* \xrightarrow{a_1,t_1} \langle q_1, v_1\rangle \xrightarrow{\emptyset}{}^* \xrightarrow{a_2,t_2} \ldots \xrightarrow{\emptyset}{}^* \xrightarrow{a_n,t_n} \langle q_n, v_n\rangle$ in A such that $q_n \in F$. t_i here indicates the elapsing time of transition $\xrightarrow{a_i}$ from the beginning (where the time equals 0). The set of all finite traces of A is denoted by $\mathbf{Traces}_{TA}(A)$.

3 Intelligent Railroad Crossing System

In this section we analysis a simple spatio-temporal system—Intelligent Railroad Crossing System (IRCS) and model it using STeC. In IRCS (Fig. 8) there are two agents: a 'smart' train and a 'smart' gate. The 'smart' gate is located at the crossing where a railroad lies in east-west direction and a road lies in south-north direction. The 'smart' train communicates with the 'smart' gate dynamically when it tries to pass the crossing. Each 'Lapp', 'Lpass', 'Lstop', 'Lleave' on the track indicates the logical locations in which only proper behaviour can be performed by agent train at the right time. The states of agent gate ('Lclose', 'Lopen') are also considered as logical locations, in which only proper behaviour can be performed by the gate at the right time.

The scenario of IRCS is as follows:

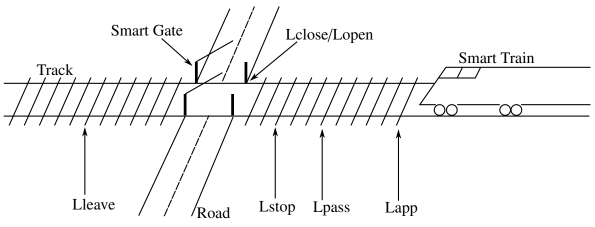


Fig. 8: Intelligent Railroad Crossing System

Train—

1) The train approaches the crossing at location 'Lapp', taking 20s, and arrives at location 'Lpass', during which it sends message 'Appr' to the gate.
2) At 'Lpass' the train waits for gate's messages, if it receives message 'NonCross', then stops after 30s and reaches location 'Lstop'. If it receives 'Cross', it keeps running, passing the crossing within 60s, and finally reaches 'Lleave'.
3) At 'Lstop' the train waits for message 'Cross', after receiving it, it restarts and passes the crossing.
4) After passing the crossing, the train sends 'Leave' to the gate at location 'Lleave'.

Gate—

1) At the beginning, the gate is assumed to be open, waiting for messages from the train.
2) After the gate receives message 'Appr' from the train, it tries to close itself in 10s.
3) The gate spends 1s for judging if itself is successfully closed. If so, the gate sends message 'Cross'. If not, the gate sends message 'NonCross'.
4) After receiving message 'Leave' from the train, the gate opens itself for 10s.

Agent train can be described by STeC as follows:

$$T = \mathbf{Send}^C_{(Lapp,t)}(Appr); Approach_{(Lapp,t)}(Lpass, 20) \trianglerighteq_{20} \left\{ \begin{array}{l} \mathbf{Get}^C_{(Lpass,t+20)}(Cross) \rightarrow Pass_{(Lpass,t+20)}(Lleave, 60); \\ \mathbf{Send}^C_{(Lleave,t+80)}(Leave); \mathbf{Skip}_{(Lleave,t+80)} \\ \talloblong \\ \mathbf{Get}^C_{(Lpass,t+20)}(NonCross) \rightarrow (Stop_{(Lpass,t+20)}(Lstop, \\ 30) \parallel \mathbf{Wait}^C_{(Lpass,t+20,Lstop,t+21)}(Cross)); \\ Pass_{(Lstop,t+50)}(Lleave, 60); \mathbf{Send}^C_{(Lleave,t+110)}(Leave); \\ \mathbf{Skip}_{(Lleave,t+110)} \end{array} \right\}$$

where **Wait** is a process, defined as:

$$\mathbf{Wait}^C_{(l,t,l',t')}(m) = \mathbf{Stop}_{(l,t)} \trianglerighteq (\mathbf{Get}^C_{(l',t')}(m) \rightarrow \mathbf{Skip}_{(l',t')}).$$

It describes an agent waits for a message m at (l, t) until it receives the message at location l' and time t'.

Agent gate can be described by STeC as follows:

$$G \;=\; \mathbf{Wait}^C_{(Lopen,t',Lopen,t)}(Appr); Closing_{(Lopen,t)}(10);$$

$$\left\{\begin{array}{l} Closed_{(Lclose,t+10)}(1); \mathbf{Stop}_{(Lclose,t+11)} \trianglerighteq (\\ \mathbf{Send}^C_{(Lclose,t+20)}(Cross); \mathbf{Wait}^C_{(Lclose,t+20,Lclose,t+80)}(Leave); \\ Open_{(Lclose,t+80)}(Lopen, 10)); \mathbf{Skip}_{(Lopen,t+90)} \\ [] \\ Unclosed_{(Lopen,t+10)}(1); ((\mathbf{Stop}_{(Lopen,t+11)} \trianglerighteq \\ \mathbf{Send}^C_{(Lopen,t+20)}(NonCross)) \parallel \\ Close_{(Lclose,t+11)}(Lclose, 10)); \mathbf{Send}^C_{(Lclose,t+21)}(Cross); \\ \mathbf{Wait}^C_{(Lclose,t+21,Lclose,t+110)}(Leave); \\ Open_{(Lclose,t+110)}(Lopen, 10); \mathbf{Skip}_{(Lopen,t+120)} \end{array}\right\}$$

By composing them together by parallel operator we have:

$$P_{IRCS} = T \bowtie G.$$

Initially, we set $t = 300$ to be the starting point of this system and set $t' = 0$.

4 The Verification of STeC/CCSL Framework

This section is the main contribution of this paper. We build the STeC/CCSL framework by proposing a theory linking them and a model checking scheme for verifying CCSL properties in STeC models. Since CCSL constraint is not a CTL-like language and its semantics does not support tree-like searching machanism, adding that we have to consider generators—a special clock (introduced below) as a companion in model checking STeC processes, we can not directly apply the conventional model checking algorithm [12] in our purpose. Instead, we proposed an alternative approach by taking a CCSL specification as an observer for STeC model, rather than an assertion. In this way, as we will see soon, essentially the satisfaction of a CCSL specification can be transformed into a classical model checking problem with a simple CTL formula to be checked.

In order to reason about STeC and CCSL models, we first introduce the notions of STeC traces and traces in CCSL specifications (Section 4.1-4.3). Then in Section 4.4 we connect STeC and CCSL by defining how a STeC process satisfy a CCSL specification in a natural way. In Section 4.5, 4.6 we propose the theory and algorithms to translate both STeC and CCSL into TA, in order to carry out the model checking. Finally we propose a model checking framework for STeC and CCSL in Section 4.7. Section 4.8 analyzes the time complexity of our model checking algorithm. We show that the translation process does not increase the complexity of the model checking procedure. In Section 4.9, we consider the model checking framework in which the model can contains infinite traces. With it we can extend STeC/CCSL framework to a general one with any modelling languages with infinite traces. Similar linking theory can be built while the same model checking scheme can be applied to it.

4.1 Traces in STeC

Since there is no recursion rules in the syntax of STeC, so in this paper, it is enough for us to consider only finite traces. However, as pointed out in Section 4.9, our theory proposed in this section can be easily extended to the case for other modelling languages with recursion rules (which thus have infinite traces).

Definition 4.1 (Initial Configuration, Configurations). *The initial configuration of a process P, denoted as $ic_P = \langle P, l^i_P, t^i_P \rangle$, is defined as follows:*

1) $ic_{\mathbf{Send}^C_{(l,t)}(m)} = \langle \mathbf{Send}^C_{(l,t)}(m), l, t \rangle$,
   $ic_{\mathbf{Get}^C_{(l,t)}(m)} = \langle \mathbf{Get}^C_{(l,t)}(m), l, t \rangle$,
   $ic_{\alpha_{(l,t)}(l',\delta)} = \langle \alpha_{(l,t)}(l', \delta), l, t \rangle$,
   $ic_{\beta_{(l,t)}(\delta)} = \langle \beta_{(l,t)}(\delta), l, t \rangle$,
   $ic_{\mathbf{Stop}_{(l,t)}} = \langle \mathbf{Stop}_{(l,t)}, l, t \rangle$,
   $ic_{\mathbf{Skip}_{(l,t)}} = \langle \mathbf{Skip}_{(l,t)}, l, t \rangle$.
2) *For any $P = P_1 * P_2$ where $* \in \{;, [], \parallel, \trianglerighteq_\delta, \trianglerighteq, \bowtie\}$, if $ic_{P_1} = \langle P_1, l, t \rangle$, then $ic_P = \langle P, l, t \rangle$.*

The set of all configurations of P is defined as:

$$\mathbf{Conf}_{STeC}(P) = \{\langle Q, l, t \rangle \mid ic_P \xrightarrow{A}{}^* \langle Q, l, t \rangle\}.$$

Definition 4.2 (STeC Traces). *A word in STeC is a quadruple $e = \langle a, l, t, \delta \rangle$, where $a \in$ **Alp** is an alphabet, l is a location, t is a time point and δ is the duration of an event. The projection of each item in word e is defined as π_{act}, π_l, π_t and π_δ respectively. $\pi_{act}(\langle a, l, t, \delta \rangle) = a$, $\pi_l(\langle \mu, l, t, \delta \rangle) = l$, $\pi_t(\langle \mu, l, t, \delta \rangle) = t$, $\pi_\delta(\langle \mu, l, t, \delta \rangle) = \delta$. $\mathbb{E}_{STeC}$ denotes the set of all words in STeC.*

A trace in STeC is a finite sequence of STeC words which is defined as a partial, down-closed, finite function $tr : \mathbb{N}^+ \rightarrow \mathbb{E}_{STeC}$. Given a process $P \in \mathbb{P}_{STeC}$, a trace

$$tr = \langle a_1, l_1, t_1, \delta_1 \rangle ... \langle a_n, l_n, t_n, \delta_n \rangle$$

is a trace of P iff there are transitions:

$$\langle P, l^i_P, t^i_P \rangle \xrightarrow{\emptyset}{}^* \xrightarrow{a_1} \langle P_1, l_1, t_1 \rangle \xrightarrow{\emptyset}{}^* \xrightarrow{a_2} ... \xrightarrow{\emptyset}{}^* \xrightarrow{a_n} \langle P_n, l_n, t_n \rangle$$

such that $t_1 - t_P^i = \delta_1$*,* $t_i - t_{i-1} = \delta_i$ *for* $i = \{2, 3, ..., n\}$*,* $P_n = E$*. The set of all finite traces of P is denoted as* $\textbf{Traces}_{STeC}(P)$*.*

As an example, from agent gate G (in Section 3) we have one trace $tr \in \textbf{Traces}_{STeC}(G)$:

$$\begin{aligned} tr = &\langle C?Appr, Lopen, t, t - t' \rangle \langle Clossing, Lclose, t + 10, 10 \rangle \\ &\langle Closed, Lclose, t + 11, 1 \rangle \langle C!Cross, Lclose, t + 20, 0 \rangle \\ &\langle C?Leave, Lclose, t + 80, 0 \rangle \langle \textbf{Skip}, Lclose, t + 80, 0 \rangle \\ &\langle Open, Lopen, t + 90, 10 \rangle \langle \textbf{Skip}, Lopen, t + 90, 0 \rangle. \end{aligned}$$

4.2 Assumptions in CCSL Clock and STeC Model

From the definition of CCSL clock in Section 2.2.1 we see that a clock is assumed to make only one observation at a time for the same event. In other word, each clock ticks only once at a time. This stipulation is mainly for two reasons: 1) At the beginning, CCSL aims at describing logical and chronometrical relations between events of real-time embedded systems, in which it is unnecessary to consider that two same events occur at a time (for example, in hardware systems, signals are only expected to occur once at a time). 2) From the view of observers, when two same events occur simultaneously, it is natural to think that actually there is no way to tell them apart.

In our case in STeC models, it is meaningfulless to consider that an event occurs more than once at a time. For example, in reality, it is not possible that a system can send two same messages (to the same channel) at a single time point (e.g., $P = \textbf{Send}^C_{(l,t)}(m_1); \textbf{Send}^C_{(l,t)}(m_1)$). So we keep this assumption made for CCSL clock and only consider those STeC processes in which an event can only be triggered once at a time. We call them 'normal processes', as defined below.

Definition 4.3 (Normal STeC Process)**.** *A normal STeC process P is a process such that for any* $tr \in \textbf{Traces}_{STeC}(P)$*, it satisfies*

$$\nexists \langle a, l, t, \delta \rangle, \langle a', l', t', \delta' \rangle \in cod(tr).(a = a' \wedge t = t').$$

In the rest of this paper, all STeC processes we discuss are normal processes.

For the same reason, we shall only consider 'normal traces' in TA.

Definition 4.4 (Normal TA Traces)**.** *Given a TA A, the set of finite normal traces, denoted by* $\textbf{Traces}^N_{TA}(A)$*, is a subset of finite traces of A that satisfies for any* $tr \in \textbf{Traces}^N_{TA}(A)$*,*

$$\nexists \langle a, t \rangle, \langle a', t' \rangle \in cod(tr).(a = a' \wedge t = t').$$

4.3 Traces of a Time Structure, $\mathcal{TS}$ of a Specification

We introduce the notion of traces in a time structure, and the notion of time structures of a specification. They are helpful for us to characterize STeC traces in time structures , and to describe the relationship between CCSL TA and specifications latter.

Definition 4.5 (Traces of a Time Structure)**.** *A finite trace*[1] $tr : \mathbb{N}^+ \rightarrow \mathcal{I}$ *of a time structure* $TS = \langle \mathcal{I}, <, \equiv, H \rangle$ *is a partial, down-closed, finite function which satisfies that*

1) $tr(1) = c_i[1]$ *for some* c_i *in* TS*.*
2) *For any* $i, j \in \mathbb{N}^+$*,* $i < j$ *implies* $tr(i) \leq tr(j)$*.*
3) *For any* $i \in \mathbb{N}^+$*, if there is a* $k \in \mathcal{I}$ *such that* $tr(i) \leq k \leq tr(i+1)$*, then* $k = tr(i)$ *or* $k = tr(i+1)$*.*

Fig. 9 gives an example of TS trace. The dashed arrow indicates the partial order between trace elements (square dots).

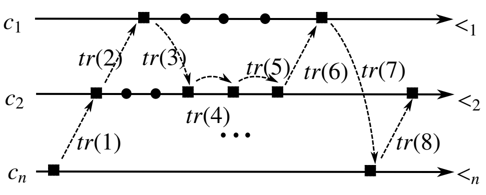


Fig. 9: An example of TS trace

The notion of time structures for a specification underlies in the definition of specifications [9]. We formalize it here.

Definition 4.6 (Time Structure of Specification)**.** *Given two time structures* $TS_1 = \langle \mathcal{I}_1, <_1, \equiv_1, H_1 \rangle$*,* $TS_2 = \langle \mathcal{I}_2, <_2, \equiv_2, H_2 \rangle$*,* $TS_1 \cup TS_2$ *computes the union of the two time structures pointwisely:*

$$TS_1 \cup TS_2 = \langle \mathcal{I}_1 \cup \mathcal{I}_2, <_1 \cup <_2, \equiv_1 \cup \equiv_2, H_1 \cup H_2 \rangle.$$

Let $spec = \{Cn_1, Cn_2, ..., Cn_k\}$ *be a specification, we can give its time structures* $\mathcal{TS}_{spec}$*. Each* $TS \in \mathcal{TS}_{spec}$ *is given by selecting* $TS_1 \in \mathcal{TS}_{Cn_1}$*,* $TS_2 \in \mathcal{TS}_{Cn_2}$*, ...,* $TS_k \in \mathcal{TS}_{Cn_k}$ *such that:*

$$TS = TS_1 \cup TS_2 \cup ... \cup TS_k$$

is well-structured.

As an example, Fig. 10 shows a time structure of constraints $spec = \{c_a \prec c_b, c_b \prec c_d, c_a \ alternatesWith \ c_d\}$, with h_1, h_2, h_3 the well-selected constraint mappings in TS_1, TS_2, TS_3.

[1] As the same reason discussed in Section 4.2, we only consider finite traces.

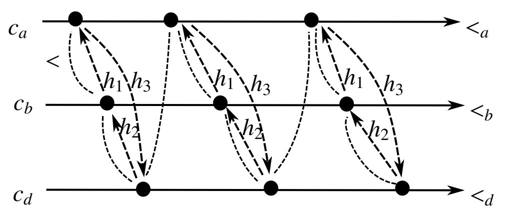


Fig. 10: A time structure of $\{c_a \prec c_b, c_b \prec c_d, c_a\ alternatesWith\ c_d\}$

4.4 Satisfaction of a CCSL Specification

Intuitively, each tick of a CCSL clock corresponds to an occurrence of one or more STeC events. The satisfaction of a CCSL specification by a STeC process can thus be defined as the behaviour of the process match each constraint of the specification, i.e., all traces of the process must obey the constraints of the specification.

In all discussions below, the clock we are actually interested in is the one that keeps track of only one event.

Definition 4.7 (Event Clock). *A single-labelled clock $c_a = \langle \mathcal{I}_a, <_a, \mathcal{D}_a, \lambda_a \rangle$ associated to a label set Σ is a discrete clock with*

$$\mathcal{D}_a = \{a\},\ a \in \Sigma,$$

i.e., a clock with only one label.

To associate with the notion of events in STeC, without ambiguity, we also call label a an event, Σ an event set, c_a an event clock.

In the rest of this paper, unless we specially point out, all discrete clocks we discuss are event clocks.

The next two definitions about traces will easy our illustrations of Definition 4.11 coming soon.

Definition 4.8 (Characteristic Function). *Given a trace $tr : \mathbb{N}^+ \to \Sigma$, Σ is a set of labels. A characteristic function of trace tr for a given set $A \subseteq \Sigma$, denoted by $\phi^{tr}_A : dom(tr) \to \{0, 1\}$, is defined as*

$$\phi^{tr}_A(i) = \begin{cases} 1, \textit{ if } tr(i) \in A \\ 0, \textit{ otherwise} \end{cases}.$$

Definition 4.9 (Sub Trace). *Given a trace tr and an event set $A \subseteq \Sigma$, a sub trace of tr, denoted by $tr|_A : \mathbb{N}^+ \to \Sigma$, it is defined as:*

$$tr|_A(i) = tr(j)$$

provided that

$$j = min\{k \mid tr(k) \in A \wedge \sum_{l=1}^{k} \phi^{tr}_A(l) = i\}$$

exists.

Intuitively, the ith element of $tr|_A$ is the element of tr whose event name is in A and is the ith such element appeared in tr. It is clear to check that the sub trace $tr|_A : \mathbb{N}^+ \to \Sigma$ built by Definition 4.9 is a partial, down-closed function.

To link CCSL constraints to STeC processes, the crucial idea is to build a mapping from STeC events to CCSL clocks. We next introduce the notion of observational events and clocks.

Definition 4.10 (Observational Event, Observational Clock). *For a STeC event $a \in$ **Alp**, $\overline{a}$ denotes the event in CCSL constraints that observes a. We call it an observational counterpart, or a synchronizing counterpart of a.*

Call an event set

$$\overline{S} = \{\overline{a} \mid a \in S\}$$

the observational event set of S.

*For an event clock $c_{\overline{a}}$ in a specification with $a \in$ **Alp**, we call $c_{\overline{a}}$ an observational clock of a.*

Call the mapping $\overline{(\cdot)} : a \mapsto \overline{a}$ the observational mapping.

Note that the observational mapping is not necessarily an injection, an example is **Property 3** (Section 6), where the mapping is defined as $\overline{(\cdot)} = \{Appr \mapsto appr, Closed \mapsto close, Close \mapsto close\}$.

We introduce a function $C(Cn)$ to return the set of all clocks appeared in Cn. For instance, $C(c_a \prec c_b) = \{c_a, c_b\}$. A function $\mathcal{D}(Cn)$ returns the set of all labels appeared in Cn. It is defined as $\mathcal{D}(Cn) = \{a \mid a \in \pi_{\mathcal{D}}(c) \wedge c \in C(Cn)\}$. For example, $\mathcal{D}(c_a \prec c_b) = \{a, b\}$. For a specification $spec$, we have $C(spec) = \bigcup_{Cn \in spec} C(Cn)$ and $\mathcal{D}(spec) = \bigcup_{Cn \in spec} \mathcal{D}(Cn)$ to compute the set of clocks and events appeared in it.

The next definition declares the link between CCSL and STeC.

Definition 4.11 (Satisfaction of CCSL Specification). *Let P be a STeC process, spec be a CCSL specification defined over $\mathcal{TS}_{spec}$, let*

$$\overline{(\cdot)} : S \to \mathcal{D}(spec)$$

*be an observational mapping, $S \subseteq$ **At**(P). Given a trace $tr \in$ **Traces**$_{STeC}(P)$, an injective mapping*

$$f : dom(tr|_S) \to \bigcup_{a \in \mathcal{D}(spec)} \mathcal{I}_a$$

is defined as

$$f(i) = c_{\overline{x_i}}[j]$$

where $x_i = \pi_{act}(tr|_S(i))$, $j = \sum_{k=1}^{i} \phi^{tr|_S}_{x_i}(k)$, $c_{\overline{x_i}}$ is an observational clock of x_i.

Say tr satisfies spec with respect to $\overline{(\cdot)}$, denoted as tr $\models_{\overline{(\cdot)}}$ spec, iff there exists a time structure

$$TS = \langle \bigcup_{a \in \mathcal{D}(spec)} \mathcal{I}_a, <, \equiv, H \rangle \in \mathcal{TS}_{spec}$$

such that the mapping f satisfies for any $i, j \in dom(tr|_S)$ with $\pi_{act}(tr|_S(i)) \neq \pi_{act}(tr|_S(j))$,

1) $\pi_t(tr|_S(i)) < \pi_t(tr|_S(j))$ implies $f(i) < f(j)$.
2) $\pi_t(tr|_S(i)) = \pi_t(tr|_S(j))$ implies $f(i) \equiv f(j)$.

In fact, f is a trace of such a TS.

*If tr $\models_{\overline{(\cdot)}}$ spec for all tr $\in$ **Traces**$_{STeC}(P)$, we say the process P satisfies spec with respect to $\overline{(\cdot)}$, written as*

$$P \models_{\overline{(\cdot)}} spec.$$

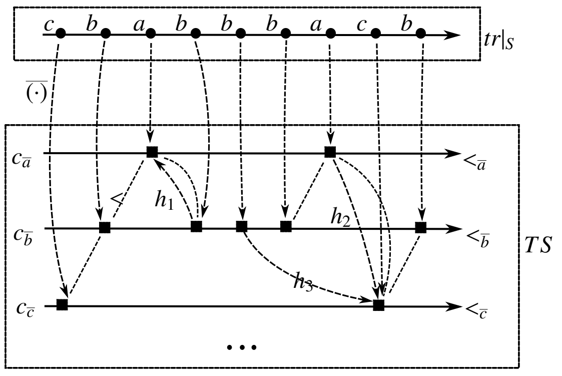


Fig. 11: Relation between $tr|_S$ and TS

Fig. 11 shows a clear picture of how we build the trace f in TS, where $\overline{(\cdot)}$ is an injection that maps x to its observational event $\overline{x}$ ($x \in \{a, b, c\}$) in TS. The ith event x is mapped to the jth tick of the corresponding clock. The sequence (made of square dots) is exactly the trace f.

Definition 4.11 essentially links an occurrence of a STeC event to a tick of CCSL clock by the mapping $\overline{(\cdot)}$ and f. It is easy to prove that f is an injection and is a trace of TS. The time structure TS plays two roles: on one hand, it p-reserves the order from the STeC trace to its corresponding observational trace f in TS (shown as the conditions '1)', '2)' above). On the other hand, TS is itself well-structured over *spec* (because $TS \in \mathcal{TS}_{spec}$), which means that the f satisfies the specification *spec*.

Note that in the above definition, we do not assume that *spec* only contains observational clocks. It may contain other clocks, we call them generated clocks, which will be introduced next.

4.5 Generated Clocks

Sometimes when we specify properties, we need to produce an auxiliary clock (which does not observe any STeC events) in order to give a correct description. We call such clock a *generated clock* [24]. For example in **Property 2** (in Section 6), we wish to express that if the train stops, it must stop 50s after the train approaches. For this property, it is wrong to use the specification

$$c_{stop} = c_{appr} \; delay \; 50 \; on \; IdealClk$$

since the train may not stop after it approaches the gate according to the behaviour of P_{IRCS} given in Section 3. The solution is to introduce a new clock named $c_{mayStop}$, to record the time points at which c_{stop} may occur. Thus the correct specification results in

$$\{c_{mayStop} = c_{appr} \; delay \; 50 \; on \; IdealClk, c_{stop} \subseteq c_{mayStop}\},$$

which means, if the train stops, it must stop at the time points 50s after the train approaches. Here the generated clock $c_{mayStop}$ ticks restrictively by the two primitive constraints above.

Definition 4.12 (Generated Event, Generated Clock)**.** *An event $a \notin$ **Alp** is called a generated event. An event clock $c_a = \langle \mathcal{I}_a, <_a, \mathcal{D}_a, \lambda_a \rangle$ in a specification is called a generated clock.*

A set of generated events is denoted by Σ_G.

Apart from these definitions, we give an illustration for the 'observational relationship' between STeC models and CCSL contraints, see Fig. 12.

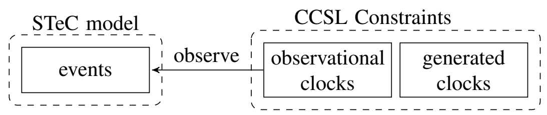


Fig. 12: Relations between a STeC model and CCSL constraints

4.6 Timed Model for CCSL and STeC

As indicated at the beginning of Section 4, an alternative approach is to take CCSL constraints as observers for STeC processes. In order to implement such an observational relationship indicated in Fig. 12 so that model checking can be carried out, we need to translate both STeC and CCSL to LTSs.

For CCSL, a translation into TA [17] is based on Clock Labelled Transition System (cLTS) in [19], which is an untimed synchronous model where if two ticks occur simultaneously, it is expressed as a set of labels. For example, constraint

$$c_{\overline{a}} \subseteq c_{\overline{b}}$$

is expressed as a cLTS in Fig. 13. If $c_{\overline{a}}$ and $c_{\overline{b}}$ tick simultaneously, the cLTS emits a set $\{\overline{a}, \overline{b}\}$.

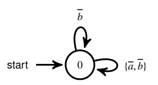


Fig. 13: cLTS of $c_{\overline{a}} \subseteq c_{\overline{b}}$

However, cLTS is not suitable for asynchronous languages where only one occurrence of an event is expected at a time. If two events occur at the same time, it causes nondeterminism to choose which one come first. Since STeC is an asynchronous language, when dealing with synchronization with STeC models, we need to consider the event $\{\overline{a}, \overline{b}\}$ separately. For the example above if there are events a, b happen at the same time in a STeC model, we need to consider the synchronization between a and $\overline{a}$, b and $\overline{b}$ separately, rather than $\{a, b\}$ and $\{\overline{a}, \overline{b}\}$ as a whole. In addition, cLTS does not support time issues, thus in which we can not express constraints relating to dense clocks, e.g., the *Delay On* operator for dense clock.

For this reason, based on [17] we consider an asynchronous TA model for CCSL. In TA, we turn a set of synchronous events in cLTS into several asynchronous events that occur at the same physical time. For example, if two ticks $\overline{a}$, $\overline{b}$ happen at the same time, we introduce two transitions indicating the occurrences of $\overline{a}$ and $\overline{b}$ separately, but with no time consuming between them (as in Table. 3 shows). As a result our timed model of CCSL not only can reflect the logical aspect of clocks, but also the chronometrical aspect of them.

For STeC, the operational semantics is itself time sensitive. So we propose a translation from STeC into its equivalent TA.

We now implement the observations from CCSL to STeC echoing Fig. 12. It turns out to be the synchronized product of timed automata originally owed to [23]. We modify the synchronizing condition there in order to adapt it to the observational mappings defined in Definition 4.10.

Definition 4.13 (Synchronized Product of TA)**.** *Given two TA $A_1 = (Q_1, S_1 \uplus O, C_1, i_1, Ed_1, I_1, AP_1, L_1, F_1)$, $A_2 = (Q_2, S_2 \uplus \overline{O}, C_2, i_2, Ed_2, I_2, AP_2, L_2, F_2)$, where $\overline{O}$ is the observable event set of O by some mapping $\overline{(\cdot)}$. The synchronized product*

$$A_1 \times A_2 = (Q, S, C, i, Ed, I, AP, L, F)$$

is defined as:

1) *$Q \subseteq Q_1 \times Q_2$ contains states of the form $\langle q_1, q_2\rangle$ where $q_1 \in Q_1$ and $q_2 \in Q_2$.*
2) *$S = S_1 \cup S_2 \cup \{\tau\}$, τ is an internal event, $\tau \notin S_1 \cup S_2$.*
3) *$C = C_1 \uplus C_2$.*
4) *$i = \langle i_1, i_2\rangle$.*
5) *$Ed = Ed_a \cup Ed_b \cup Ed_c$ where*
$$\begin{aligned} Ed_a = \ & \{\langle\langle q_1, q_2\rangle, s_1, g_1, Y_1, \langle q_1', q_2\rangle\rangle \mid s_1 \in S_1 \wedge \\ & \langle q_1, s_1, g_1, Y_1, q_1'\rangle \in Ed_1\}. \\ Ed_b = \ & \{\langle\langle q_1, q_2\rangle, s_2, g_2, Y_2, \langle q_1, q_2'\rangle\rangle \mid s_2 \in S_2 \wedge \\ & \langle q_2, s_2, g_2, Y_2, q_2'\rangle \in Ed_2\} \\ Ed_c = \ & \{\langle\langle q_1, q_2\rangle, \tau, g_1 \wedge g_2, Y_1 \cup Y_2, \langle q_1', q_2'\rangle\rangle \mid \\ & o \in O \wedge \overline{o} \in \overline{O} \wedge \langle q_1, o, g_1, Y_1, q_1'\rangle \in Ed_1 \wedge \\ & \langle q_2, \overline{o}, g_2, Y_2, q_2'\rangle \in Ed_2\}. \end{aligned}$$
6) *For each $\langle q_1, q_2\rangle \in Q$, $I(\langle q_1, q_2\rangle) = I_1(q_1) \wedge I_2(q_2)$.*
7) *$AP = AP_1 \cup AP_2$.*
8) *For each $\langle q_1, q_2\rangle \in Q$, $L(\langle q_1, q_2\rangle) = L(q_1) \cup L(q_2)$.*
9) *F contains state $\langle q_1, q_2\rangle$ where $q_1 \in F_1$ and $q_2 \in F_2$.*

4.6.1 TA for CCSL Specification

The timed semantics of primitive constraints in TA we give is similar to the untimed one in cLTS in [19], except for the primitives that concerns time issues, such as 'sub clock' and 'delay on' for dense clock.

The translation from primitive constraints into TA is given in Table 3, where all states of each TA are accepting states. t, c are clocks. Each transition is labelled by three parts: an event, a guard and a set of reset clocks. For example, in the TA of $c_b = c_a Delay\ n\ On\ c_d$, the condition '$b, c = n-1, t := 0$' means that the event b is triggered when $c = n-1$, t is reset to zero. The constraints drawn beside nodes are invariants. e.g., in the TA of $c_a \subseteq c_b$ the invariant of state 2 is $t \leq 0$.

The TA of a specification is the composition of the TA of its primitive constraints. The composition of cLTS is given in [19], based on which here we give a version for TA.

Definition 4.14 (Composition for CCSL TA)**.** *Given two TA $A_1 = (Q_1, S_1, C_1, i_1, Ed_1, I_1, AP_1, L_1, F_1)$, $A_2 = (Q_2, S_2, C_2, i_2, Ed_2, I_2, AP_2, L_2, F_2)$ of primitive constraints, let*

$$S_3 = S_1 \cap S_2$$

be the common events of A_1 and A_2. The composition for CCSL TA, denoted by $A_1 \otimes A_2$, is defined just the same as $A_1 \times A_2$ in Definition 4.13, except that

$$S = S_1 \cup S_2$$

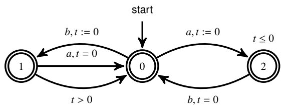
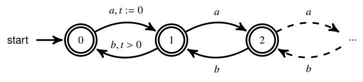
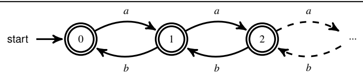
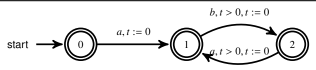
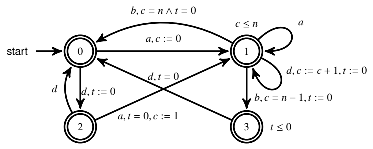
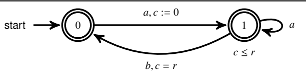
spec	TA
$c_a \subseteq c_b$	
$c_a \prec c_b$	
$c_a \preceq c_b$	
c_a *alternateWith* c_b	
$c_b = c_a Delay\ n$ $On\ c_d$	
$c_b = c_a\ Delay\ r$ $On\ IdealClk$	

Table 3: Translation for Primitive Constraints

and

$$Ed = Ed_a \cup Ed_b \cup Ed_c$$

where

$$\begin{aligned} Ed_a = &\ \{\langle\langle q_1, q_2\rangle, s_1, g_1, Y_1, \langle q_1', q_2\rangle\rangle \mid s_1 \in S_1 - S_3 \wedge \\ &\ \langle q_1, s_1, g_1, Y_1, q_1'\rangle \in Ed_1\}. \\ Ed_b = &\ \{\langle\langle q_1, q_2\rangle, s_2, g_2, Y_2, \langle q_1, q_2'\rangle\rangle \mid s_2 \in S_2 - S_3 \wedge \\ &\ \langle q_2, s_2, g_2, Y_2, q_2'\rangle \in Ed_2\} \\ Ed_c = &\ \{\langle\langle q_1, q_2\rangle, s_3, g_1 \wedge g_2, Y_1 \cup Y_2, \langle q_1', q_2'\rangle\rangle \mid \\ &\ s_3 \in S_3 \wedge \langle q_1, s_3, g_1, Y_1, q_1'\rangle \in Ed_1 \wedge \\ &\ \langle q_2, s_3, g_2, Y_2, q_2'\rangle \in Ed_2\}. \end{aligned}$$

The synchronized transition here (in Ed_c) is different from that in Definition 4.13. The same clock events in two CCSL TA must be synchronized when making the composition.

With Table 3 and Definition 4.14, we give the translation from CCSL to TA.

Definition 4.15 (Translation from CCSL to TA)**.** *The TA semantics of a CCSL specification, denoted as $[\![spec]\!]_{CCSL}$, is defined as*

$$[\![spec]\!]_{CCSL} = \bigotimes_{Cn \in spec} [\![Cn]\!]_{CCSL}$$

, where each $[\![Cn]\!]_{CCSL}$ is defined according to Table 3.

As an example, Fig. 14 gives the translated TA of specification $spec_2$ of **Property2** in Section 6, where c_1, c_2 corresponds to the clock in $[\![c_{mayStop} = c_{appr}\ delay\ 50\ on\ IdealClk]\!]_{CCSL}$ and $[\![c_{stop} \subseteq c_{mayStop}]\!]_{CCSL}$ respectively. Events $\overline{x}$ expresses the observational events in CCSL.

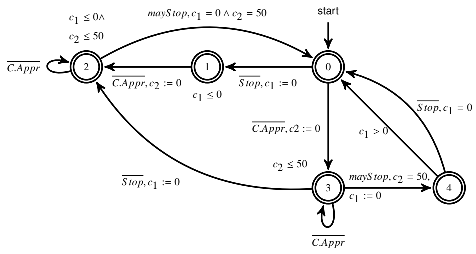

start
$c_1 \le 0 \wedge$ $c_2 \le 50$
$mayStop, c_1 = 0 \wedge c_2 = 50$
$\overline{C.Appr}$
$\overline{C.Appr}, c_2 := 0$
$c_1 \le 0$
$\overline{Stop}, c_1 := 0$
$\overline{Stop}, c_1 = 0$
$c_1 > 0$
$\overline{C.Appr}, c2 := 0$
$c_2 \le 50$
$\overline{Stop}, c_1 := 0$
$mayStop, c_2 = 50,$ $c_1 := 0$

Fig. 14: The encoded TA of $spec_2$

The following theorem says, in some sense, we can think a CCSL specification is equivalent to its encoded TA.

Theorem 4.1. *Given a specification spec defined over $\mathcal{TS}_{spec}$. A surjective function*

$$f : \mathbf{Traces}^N_{TA}([\![spec]\!]_{CCSL}) \rightarrow \bigcup_{TS \in \mathcal{TS}_{spec}} \mathbf{Traces}_{TS}(TS)$$

is a 'forget' function that forgets the time information about a trace. It is defined that for any trace

$$tr = \langle a_1, t_1\rangle\langle a_2, t_2\rangle ... \langle a_n, t_n\rangle,$$

$$f(tr) = a_1 a_2 ... a_n : \mathbb{N}^+ \rightarrow \{a_1, a_2, ..., a_n\}$$

is a trace. Define the set

$$f^{-1}(tr) \triangleright TS = \{tr' \mid tr' \in f^{-1}(tr) \wedge f(tr') \in \mathbf{Traces}_{TS}(TS)\}.$$

We claim that $[\![spec]\!]_{CCSL}$ coincides with spec in the sense that f satisfies

1) *If $tr \in \mathbf{Traces}^N_{TA}([\![spec]\!]_{CCSL})$, then we can find a time structure TS in which $f(tr) \in \mathbf{Traces}_{TS}(TS)$, and*
   - *for any $\langle a_i, t_i\rangle$, $\langle a_j, t_j\rangle$ with $t_i < t_j$ in $[\![spec]\!]_{CCSL}$, then $a_i < a_j$ in TS,*
   - *for any $\langle a_i, t_i\rangle$, $\langle a_j, t_j\rangle$ with $t_i = t_j$ in $[\![spec]\!]_{CCSL}$, then $a_i \equiv a_j$ in TS.*
2) *If $tr \in \mathbf{Traces}_{TS}(TS)$ for some $TS \in \mathcal{TS}_{spec}$, then for all $tr' \in f^{-1}(tr) \triangleright TS$, $tr' \in \mathbf{Traces}^N_{TA}([\![spec]\!]_{CCSL})$.*

Proof. It can be proved by the induction based on the structure of $spec$. For primitive constraints, we just give an example: $c_a \subseteq c_b$. Others are similar.

(Base case, $\Rightarrow$). Let $tr = \langle a_1, t_1\rangle\langle a_2, t_2\rangle ... \langle a_n, t_n\rangle$ is a normal TA trace in $[\![spec]\!]_{CCSL}$, and $f(tr) = a_1 a_2 ... a_n$, we build a $TS = \langle \mathcal{I}, <, \equiv, \{h\}\rangle$ following several steps:

1) First, let $\mathcal{I} = \mathcal{I}_a \cup \mathcal{I}_b$. In TS we add relations to $<$ such that TS is well-structured in the sense of c_a and c_b:

- whenever $x <_a y$ in c_a, we add $x < y$ in $<$,
- whenever $x <_b y$ in c_b, we add $x < y$ in $<$.

2) Make $f(tr)$ become a trace in TS. To realize it, first we define the mapping

$$g : dom(f(tr)) \to \mathcal{I}_a \cup \mathcal{I}_b$$

as

$$g(i) = c_{a_i}[j]$$

where $j = \sum_{k=1}^{i} \phi_{a_i}^{f(tr)}(k)$, $a_i \in \{a, b\}$. i.e., mapping a_i to the jth element of c_{a_i}, j is the occurrence times in $f(tr)$ before a_i. Then we enrich relations $<$ and $\equiv$ as:

- for $\langle a_i, t_i\rangle$, $\langle a_j, t_j\rangle$ with $t_i < t_j$, we define $g(i) < g(j)$ in TS,
- for $\langle a_i, t_i\rangle$, $\langle a_j, t_j\rangle$ with $t_i = t_j$, we define $g(i) \equiv g(j)$ in TS. (Note that a_i, a_j here must be in different clocks.)

3) The constraint mapping $h : \mathcal{I}_a \to \mathcal{I}_b$ can be built as:

- whenever $x \equiv y$, $x \in \mathcal{I}_a$ and $y \in \mathcal{I}_b$, we have $h(x) = y$. (Note that given such x, there must be just one such y with $x \equiv y$.)

Easy to check that such a TS we built is well-structured and $c_a \subseteq c_b$ is defined over it.

(Base case, $\Leftarrow$). If $tr = a_1a_2...a_n \in \mathbf{Traces}_{TS}(TS)$ for some $TS \in \mathcal{TS}_{c_a \subseteq c_b}$, according to definition of f, it is easy to check that any normal traces in $f^{-1}(tr) \rhd TS$ is in TA $[\![c_a \subseteq c_b]\!]_{CCSL}$.

(Inductive step, $\Rightarrow$). Now suppose $spec = spec_1 \cup spec_2$, and the theorem holds for any constraints in $spec_1$ and $spec_2$ respectively.

If $tr \in \mathbf{Traces}^N_{TA}([\![spec]\!]_{CCSL})$, since

$$[\![spec]\!]_{CCSL} = [\![spec_1]\!]_{CCSL} \otimes [\![spec_2]\!]_{CCSL}$$

, we consider $tr|_{\mathcal{D}(spec_1)} \in \mathbf{Traces}^N_{TA}([\![spec_1]\!]_{CCSL})$ and $tr|_{\mathcal{D}(spec_2)} \in \mathbf{Traces}^N_{TA}([\![spec_1]\!]_{CCSL})$. By the assumption we know there exists $TS_1 = \langle \mathcal{I}_1, <_1, \equiv_1, H_1\rangle$, $TS_2 = \langle \mathcal{I}_2, <_2, \equiv_2, H_2\rangle$ such that $f(tr|_{\mathcal{D}(spec_1)})$ is a trace in TS_1 and $f(tr|_{\mathcal{D}(spec_1)})$ is a trace in TS_2, and TS_1, TS_2 are well-structured with $spec_1$, $spec_2$ defined over them respectively. Consider

$$TS = TS_1 \cup TS_2,$$

easy to check that TS must be a well-structured time structure with $spec$ defined over it.

Fig. 15 gives a picture of the relation between TS, TS_1 and TS_2, where the dashed square indicates the common part of events in $\mathcal{I}_1$ and $\mathcal{I}_2$, the solid squares indicate the events of $\mathcal{I}_1$ and $\mathcal{I}_2$ respectively. From it we can see that in order to make tr become a trace in TS, the only enrichment we need to make is the relations (like r_1) that neither belongs to TS_1 or TS_2 (while relations like r_2, r_3, r_4 have already existed in either TS_1 or TS_2). So for any $\langle x, t_x\rangle$, $\langle y, t_y\rangle$ in tr with $x \in \mathcal{I}_1 - \mathcal{I}_2$, $y \in \mathcal{I}_2 - \mathcal{I}_1$, we have:

1) if $t_x < t_y$, we add $x < y$ in TS,
2) if $t_x = t_y$, we add $x \equiv y$ in TS.

We write the time structure after the enrichment as TS'. It is not hard to show that TS' is still well-structured. This is because from Fig. 15 one can easily see that any triangle relations (like r_1, r_2, r_3) is impossible to be derived as a contradiction in TS'. For example, if $x <_1 z$ in trace $tr|_{\mathcal{D}(spec_1)}$, $z =<_2 y$ in trace $tr|_{\mathcal{D}(spec_2)}$, then there must be $x < y$ in trace tr. And $x < y$ does not affect any constraint mappings since all of them do not concern x and y at the same time.

(Inductive step, $\Leftarrow$). If $tr \in \mathbf{Traces}_{TS}(TS)$ for some $TS \in \mathcal{TS}_{spec}$, we can split $TS = TS_1 \cup TS_2$ so that $tr|_{\mathcal{I}_1} \in \mathbf{Traces}_{TS}(TS_1)$ and $tr|_{\mathcal{I}_2} \in \mathbf{Traces}_{TS}(TS_2)$, with TS_1, TS_2 well-structured and $spec_1$, $spec_2$ defined over them respectively. Thus by assumption for any $tr_1 \in f^{-1}(tr|_{\mathcal{I}_1}) \rhd TS_1$, $tr_2 \in f^{-1}(tr|_{\mathcal{I}_2}) \rhd TS_2$, there are $tr_1 \in \mathbf{Trace}^N_{TA}([\![spec_1]\!]_{CCSL})$ and $tr_2 \in \mathbf{Trace}^N_{TA}([\![spec_2]\!]_{CCSL})$. Since for any $tr' \in f^{-1}(tr) \rhd TS$, we can find such tr_1, tr_2 with $tr'|_{\mathcal{I}_1} = tr_1$ and $tr'|_{\mathcal{I}_2} = tr_2$, from the definition of f, easy to see that $tr' \in \mathbf{Trace}^N_{TA}([\![spec]\!]_{CCSL})$. □

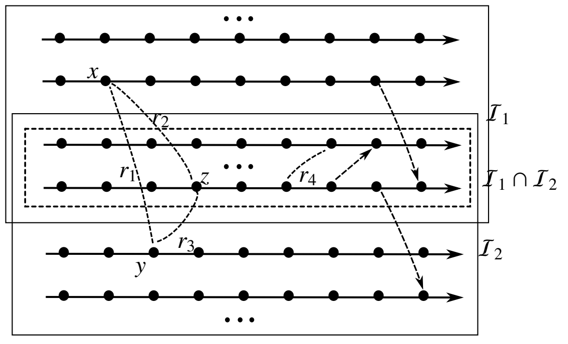


Fig. 15: Relation between TS, TS_1 and TS_2

Note that $f(tr)$ might be a trace in more than one time structures. That is why we consider $f^{-1}(tr) \rhd TS$ instead of $f^{-1}(tr)$. Function f is surjective because TA traces contain time information that indicates at which time an event occurs. However in the time structure TS, though it has time issues (like $Sub\ Clock$ or $Delay\ On$ for dense clock), only the relationship between clocks is reflected.

4.6.2 TA for STeC Process

Intuitively, according to the operational semantics of STeC, each STeC configuration can be seen as a configuration in a TA, each time-only transition between configurations can be seen as a time-only transition between TA configurations, each process transition can be seen as a transition between TA states. In this section we formalize this intuition by first introducing the notion of 'STeC configuration region' where there are only time-only transitions between two configurations, then we correspond each region to a state in a TA. In this way the process transitions correspond exactly to the transitions between states.

One thing should be noticed that the location in STeC turns out to be an atomic proposition in the state of TA (as shown in Definition 4.18). The encoded TA does not lose the capability to express the 'locations' in STeC. In fact, as indicated at the beginning of Section 2.1, the location in STeC is only used for checking the spatio-temporal consistencies at the semantics level. It does not give any more information about the system. What we lose in TA is the restriction of locations: in STeC, the restriction is guaranteed by its semantics, while in TA, such restriction is missing, since we can not build a transition whose firing relies on the satisfaction of propositions. However, the lost of restrictions does not affect that we use TA as models in model checking with CCSL, since CCSL does not include any constraints that concern locations in STeC. It is efficient to require that the expressive power of a STeC process and its encoded TA are the same in the sense of Theorem 4.2.

Definition 4.16 (Configuration Region)**.** *A STeC configuration region is a set of configurations, denoted by $[P, l, t]$ where P is a process and l is a location. It is defined as*

$$[P, l, t] = \{\langle P, l, t'\rangle \mid \langle P, l, t'\rangle \in \mathbf{Conf}_{STeC}(P) \wedge t' \geq t\}$$

such that for any $\langle P, l, t_1\rangle, \langle P, l, t_2\rangle \in [P, l, t]$ with $t_2 > t_1 (\geq t)$, we have

$$\langle P, l, t_1\rangle \xrightarrow{\emptyset}{}^{*} \langle P, l, t_2\rangle$$

, i.e., there are only time-only transitions between $\langle P, l, t_1\rangle$ and $\langle P, l, t_2\rangle$.

Use reg to range over all regions.

The only existence of time-only transitions between $\langle P, l, t_1\rangle$ and $\langle P, l, t_2\rangle$ can be easily proved according to the operational semantics in Table 1, 2 .

Definition 4.17 (Configuration Region Transition System (CRTS))**.** *Based on Definition 4.16 we can define a transition system, called Configuration Region Transition System (CRTS), in which the states are configuration regions in the form of $[P, l, t]$, transitions between regions are in the form of $[P, l, t] \xrightarrow{a} [P', l', t]$ where $a \in \mathbf{Alp}$.*

Given a STeC process P, the CRTS respected to P can be built according to the following rules:

1) *For the initial configuration $\langle P, l^i_P, t^i_P\rangle$ given in advance, we have a state$[P, l^i_P, t^i_P]$.*
2) *For any $\langle P, l, t\rangle \in [P, l, t']$, if there exists a process transition $\langle P, l, t\rangle \xrightarrow{a} \langle P', l', t\rangle$, then we have a state $[P', l', t]$, and a transition $[P, l, t'] \xrightarrow{a} [P', l', t]$.*

The corresponding CRTS of P is denoted as $CRTS_P$.

Now we are ready to translate STeC into TA. The only difference between TA and CRTS is that we need to use TA clocks to express time issues instead of the symbol t.

Definition 4.18 (Translation from STeC to TA)**.** *Given a STeC process P and its corresponding CRTS $CRTS_P$, the TA semantics of P, denoted by $[\![P]\!]_{STeC}$, is a TA*

$$(Q, \Sigma, C, i, Ed, I, AP, L, F)$$

constructed according to following rules:

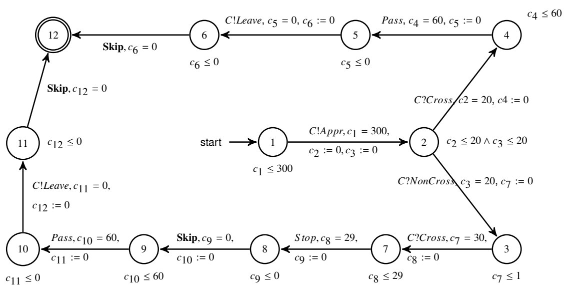
1) *Q is the set of all regions in $CRTS_P$.*
2) *$\Sigma = \mathbf{At}(P)$.*
3) *For each transition $reg_i \xrightarrow{a} reg_j$ in $CRTS_P$, we have a clock denoted as $c(reg_i, a, reg_j)$ in C.*
4) *$i = [P, l^i_P, t^i_P]$ where $\langle P, l^i_P, t^i_P\rangle$ is the initial configuration given in advance.*
5) *For any state $[P, l, t] \in Q$, $L([P, l, t]) = l$.*
6) *F has states of the form $[E, l, t]$.*
7) *For each transition $[P, l, t] \xrightarrow{a} [P', l', t']$ in $CRTS_P$, we have a transition $\langle [P, l, t], a, g, Y, [P', l', t']\rangle$ in Ed where*
   - *$g := c([P, l, t], a, [P', l', t']) = t' - t$,*
   - *$I([P, l, t])$ is in the form of*

     $$... \wedge c([P, l, t], a, [P', l', t']) \leq t' - t \wedge ...,$$

   - *Y consists of all clocks in the form of $c([P', l', t'], a, reg)$ where $reg \in Q$ is an arbitrary state.*
8) *For any $reg \in Q$, $I(reg)$ is constructed as in 7), i.e., the conjunction of the invariants of all transitions from reg.*

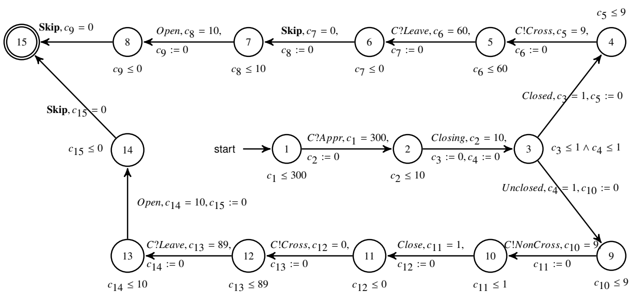
Consider the earlier example $P_{IRCS} = T \bowtie G$, we can compute both $[\![T]\!]_{STeC}$ and $[\![G]\!]_{STeC}$ shown in Fig. 16 and Fig. 17 respectively.

A STeC process is equivalent to its encoded TA in the following sense.



Skip, $c_6 = 0$
$C!Leave$, $c_5 = 0$, $c_6 := 0$
$Pass$, $c_4 = 60$, $c_5 := 0$
$c_4 \leq 60$
$c_6 \leq 0$
$c_5 \leq 0$
Skip, $c_{12} = 0$
$C?Cross$, $c_2 = 20$, $c_4 := 0$
start
$C!Appr$, $c_1 = 300$, $c_2 := 0$, $c_3 := 0$
$c_2 \leq 20 \wedge c_3 \leq 20$
$c_{12} \leq 0$
$c_1 \leq 300$
$C!Leave$, $c_{11} = 0$, $c_{12} := 0$
$C?NonCross$, $c_3 = 20$, $c_7 := 0$
$Pass$, $c_{10} = 60$, $c_{11} := 0$
Skip, $c_9 = 0$, $c_{10} := 0$
$Stop$, $c_8 = 29$, $c_9 := 0$
$C?Cross$, $c_7 = 30$, $c_8 := 0$
$c_{11} \leq 0$
$c_{10} \leq 60$
$c_9 \leq 0$
$c_8 \leq 29$
$c_7 \leq 1$

Fig. 16: The encoded TA of T



$c_5 \leq 9$
Skip, $c_9 = 0$
$Open$, $c_8 = 10$, $c_9 := 0$
Skip, $c_7 = 0$, $c_8 := 0$
$C?Leave$, $c_6 = 60$, $c_7 := 0$
$C!Cross$, $c_5 = 9$, $c_6 := 0$
$c_9 \leq 0$
$c_8 \leq 10$
$c_7 \leq 0$
$c_6 \leq 60$
$Closed$, $c_3 = 1$, $c_5 := 0$
Skip, $c_{15} = 0$
$c_{15} \leq 0$
start
$C?Appr$, $c_1 = 300$, $c_2 := 0$
$Closing$, $c_2 = 10$, $c_3 := 0$, $c_4 := 0$
$c_3 \leq 1 \wedge c_4 \leq 1$
$c_1 \leq 300$
$c_2 \leq 10$
$Unclosed$, $c_4 = 1$, $c_{10} := 0$
$Open$, $c_{14} = 10$, $c_{15} := 0$
$C?Leave$, $c_{13} = 89$, $c_{14} := 0$
$C!Cross$, $c_{12} = 0$, $c_{13} := 0$
$Close$, $c_{11} = 1$, $c_{12} := 0$
$C!NonCross$, $c_{10} = 9$, $c_{11} := 0$
$c_{14} \leq 10$
$c_{13} \leq 89$
$c_{12} \leq 0$
$c_{11} \leq 1$
$c_{10} \leq 9$

Fig. 17: The encoded TA of G

Theorem 4.2. *Given a STeC process P. A bijective function*

$$f : \mathbf{Traces}^N_{TA}([\![P]\!]_{STeC}) \rightarrow \mathbf{Traces}_{STeC}(P)$$

is defined that for any trace

$$tr = \langle a_1, t_1\rangle\langle a_2, t_2\rangle ... \langle a_n, t_n\rangle \in \mathbf{Traces}^N_{TA}([\![P]\!]_{STeC})$$

along with the transitions

$$[P, l^i_P, t^i_P] \xrightarrow{a_1} [P_1, l_1, t_1] \xrightarrow{a_2} ... \xrightarrow{a_n} [P_n, l_n, t_n],$$

we have

$$f(tr) = \langle a_1, l_1, t_1, t_1 - t^i_P\rangle\langle a_2, l_2, t_2, t_2 - t_1\rangle ... \langle a_n, l_n, t_n, t_n - t_{n-1}\rangle$$

in $\mathbf{Traces}_{STeC}(P)$. $[\![P]\!]_{STeC}$ *coincides with P in the sense that f satisfies:*

$$tr \in \mathbf{Traces}^N_{TA}([\![P]\!]_{STeC}) \textit{ iff } f(tr) \in \mathbf{Traces}_{STeC}(P).$$

Proof. From the constructions of CRTS and TA of STeC, it can be easily proved by the induction on the length of STeC processes. □

By Definition 4.18 we can translate a process $P = P_1 \bowtie P_2 \bowtie ... \bowtie P_n$ into a TA by dividing P into configuration regions. It is useful to know that our translation preserves the parallel composition $\bowtie$ in the sense of Proposition 4.1. When we use model checking tools like UPPAAL to build our models, we actually need not to translate the whole P, but each component $P_1, ..., P_n$. Since these tools take lazy-evaluation strategy to deal with parallel composition.

Definition 4.19 (Composition for STeC TA)**.** *Given two TA* $A_1 = (Q_1, S_1 \uplus Ch_1, C_1, i_1, Ed_1, I_1, AP_1, L_1, F_1)$, $A_2 = (Q_2, S_2 \uplus Ch_2, C_2, i_2, Ed_2, I_2, AP_2, L_2, F_2)$ *of STeC processes,* Ch_1, Ch_2 *are set of communicating events of the form* $C * m$ *between* A_1 *and* A_2, *where* $* \in \{!, ?\}$. *The composition of STeC TA, denoted as* $A_1 \odot A_2$, *is defined just as the same as* $A_1 \times A_2$ *in Definition 4.13, except that*

$$S = S_1 \cup S_2 \cup Ch$$

where elements in Ch *are of the form* $C.m$ *with* $C!m$ $(C?m)$, $C?m$ $(C!m)$ *in* Ch_1 *and* Ch_2 *respectively.*

$$Ed = Ed_a \cup Ed_b \cup Ed_c$$

where

$$\begin{aligned} Ed_a = {} & \{\langle\langle q_1, q_2\rangle, s_1, g_1, Y_1, \langle q_1', q_2\rangle\rangle \mid s_1 \in S_1 - Ch_1 \wedge \\ & \langle q_1, s_1, g_1, Y_1, q_1'\rangle \in Ed_1\}. \\ Ed_b = {} & \{\langle\langle q_1, q_2\rangle, s_2, g_2, Y_2, \langle q_1, q_2'\rangle\rangle \mid s_2 \in S_2 - Ch_2 \wedge \\ & \langle q_2, s_2, g_2, Y_2, q_2'\rangle \in Ed_2\} \\ Ed_c = {} & \{\langle\langle q_1, q_2\rangle, C.m, g_1 \wedge g_2, Y_1 \cup Y_2, \langle q_1', q_2'\rangle\rangle \mid \\ & (C \star m \in Ch_1 \wedge C * m \in Ch_2) \wedge \\ & \langle q_1, C \star m, g_1, Y_1, q_1'\rangle \in Ed_1 \wedge \\ & \langle q_2, C * m, g_2, Y_2, q_2'\rangle \in Ed_2\}. \end{aligned}$$

where the pair $(\star, *)$ *could be* $(!, ?)$ *or* $(?, !)$.

Proposition 4.1 (Preservation of Translation)**.** *The translation* $[\![\cdot]\!]_{STeC}$ *preserves operation* $\bowtie$ *with respect to* $\odot$, *that is, given a process* $P = P_1 \bowtie P_2 \bowtie ... \bowtie P_n$, *there is*

$$[\![P]\!]_{STeC} = \bigodot_{i=1...n} [\![P_i]\!]_{STeC}.$$

This property about STeC translation will be useful in Section 5, where we translate each agent P_i into UPPAAL TA, rather than P.

4.6.3 Algorithm Translating STeC and CCSL into TA

Based on Definition 4.14-4.18, we propose Algorithm 1-4, as a general implementation to translate STeC and CCSL into their corresponding TA. They form the base for implementing STeC/CCSL framework in practice and adopting the existing verification methodologies such as UPPAAL, which will be discussed in the next sections.

To avoid the detail of the implementation we only give a general description. We choose mathematical data structures and operations rather than the concrete ones. For example, we use mathematical set and set operations such as union

Algorithm 1 Translation from CCSL into TA

```
procedure CCSL_2_TA($spec$)
    let $R$ be an init empty TA
    for each $Cn$ in $spec$ do
        directly get the TA $R_{Cn}$ of $Cn$ based on Table 3
    Compute $R := \bigotimes_{Cn \in spec} R_{Cn}$
    return $R$
```

$A \cup B$. To save pages sometimes we use English sentence to express one of series of operations in real programs. Like in line 4 of Algorithm 1. The key words follow the traditional meanings in programming langauges, like **if then else**, **for do**, etc. **procedure** just means function, and **let** is the declaration of local variables.

Algorithm 2 Translation from STeC into TA

```
procedure STeC_2_TA($P$)
    let $A := (Q, \Sigma, C, i, Ed, I, AP, L, F)$ be an empty TA
    let $i := [P, l_P^i, t_P^i]$ be the init state
    Build_TA($A$, $i$)
    return $A$
procedure Build_TA($A$, $[P, l, t]$)
    set its prop set $L([P, l, t]) := l$
    set its inv $I([P, l, t]) := \emptyset$
    if $P = E$ then
        $F := F \cup [P, l, t]$
    let $T$:=Trans($[P, l, t]$), $S := \emptyset$
    for each $t = [P, l, t] \xrightarrow{a} [P', l', t']$ in $T$ do
        create a new clock $C := C \cup \{c_t\}$
        let $g := c_t = t' - t$
        $I([P, l, t]) := I([P, l, t]) \wedge c_t \leq t' - t$
        $Ed := Ed \cup \langle [P, l, t], a, g, \emptyset, [P', l', t'] \rangle$
        $S := S \cup [P', l', t']$
    for each $\langle reg, a', g', Y', [P, l, t] \rangle$ in $Ed$ do
        for each $c_{t'}$ where $t' = [P, l, t] \xrightarrow{a''} reg'$ in $C$ do
            $Y' := Y' \cup \{c_{t'}\}$
    for each $s$ in $S$ do
        Build_TA($A$, $s$)
procedure Trans($[P, l, t]$)
    let $T$ be the init empty transition set
    if $P$ is an atomic process then
        $T$:=Trans_Atom($[P, l, t], T$)
    else
        $T$:=Trans_Com($[P, l, t], T$)
    return $T$
```

Algorithm 1 translates CCSL to TA, the computation of ⊗ (at line 5) follows the algorithm in [19]. Here we omit the detail. Algorithm 2 contains three procedures, the main procedure is STeC_2_TA(P), it accepts a STeC process P and returns a TA. Build_TA is for building the TA step by step based on configuration region $[P, l, t]$. The whole process is based on Definition 4.18. Procedure Trans computes the transitions of region $[P, l, t]$, based on the operational semantic rules in Table 1 and 2. Algorithm 3 and 4 give the detail for the cases when P is an atomic or a compositional process.

Algorithm 3 Function Trans_Atom

```
procedure Trans_Atom($[P, l, t], T$)
    if $P = \mathbf{Send}_{(l,t)}^{C}(m)$ then
        $T := T \cup \{[P, l, t] \xrightarrow{C!m} [E, l, t]\}$
    else if $P = \mathbf{Get}_{(l,t)}^{C}(m)$ then
        $T := T \cup \{[P, l, t] \xrightarrow{C?m} [E, l, t]\}$
    else if $P = \alpha_{(l,t)}(l', \delta)$ then
        $T := T \cup \{[P, l, t] \xrightarrow{\alpha} [E, l', t + \delta]\}$
    else if $P = \beta_{(l,t)}(\delta)$ then
        $T := T \cup \{[P, l, t] \xrightarrow{\beta} [E, l, t + \delta]\}$
    else if $P = \mathbf{Skip}_{(l,t)}$ then
        $T := T \cup \{[P, l, t] \xrightarrow{\mathbf{Skip}} [E, l, t]\}$
    return $T$
```

4.7 Model Checking Framework of STeC and CCSL

By now we have all things needed to propose our model checking algorithm to verify CCSL properties in a STeC model. Echoing the beginning of Section 4, we propose an algorithm to check if $P \models_{\overline{(\cdot)}} spec$ given a process P, an observational mapping $\overline{(\cdot)}$ and a specification $spec$.

Theorem 4.3 (Model Checking CCSL Constraints in a STeC Model)**.** *Let P be a STeC process, spec be a CCSL specification, and*

$$\overline{(\cdot)} : S \to \mathcal{D}(spec)$$

be an observational mapping where $S \subseteq \mathbf{At}(P)$. Let

$$\mathcal{M} = [\![P]\!]_{STeC} \times [\![spec]\!]_{CCSL}$$

be the synchronized product of the STeC model and the CCSL specification. Then we have

$$P \models_{\overline{(\cdot)}} spec \text{ iff } \mathcal{M} \models \mathbf{AF}F_{\mathcal{M}},$$

where $F_{\mathcal{M}}$ is the set of accepting states of $\mathcal{M}$. The CTL formula $\mathbf{AF}F_{\mathcal{M}}$ means that ‘every path (starting from the initial state) will finally reach a state in $F_{\mathcal{M}}$’.

AF is a CTL operator. **AF** means that ‘for all paths, finally...’. Refer to [25] for more details.

The form $\mathcal{M} \models \mathbf{AF}F_{\mathcal{M}}$ is decidable by the conventional model checking algorithm for CTL when $\mathcal{M}$ is a finite state

Algorithm 4 Function Trans_Com

```
1:  procedure TRANS_COM($[P, l, t], T$)
2:      if $P = Q; R$ then
3:          let $T'$:=Trans($[Q, l, t]$)
4:          for each $[Q, l, t] \xrightarrow{a} [Q', l', t']$ in $T'$ do
5:              if $a$ = Skip then
6:                  $T$:=$T \cup \{[P, l, t] \xrightarrow{a} [R, l', t']\}$
7:              else
8:                  $T$:=$T \cup \{[P, l, t] \xrightarrow{a} [Q'; R, l', t']\}$
9:      else if $P = Q[]R$ then
10:         let $T'$:=Trans($[Q, l, t]$)$\cup$ Trans($[R, l, t]$)
11:         for each $[Q, l, t] \xrightarrow{a} [Q', l', t']$ in $T'$ do
12:             $T$:=$T \cup \{[P, l, t] \xrightarrow{a} [Q', l', t']\}$
13:         for each $[R, l, t] \xrightarrow{a} [R', l', t']$ in $T'$ do
14:             $T$:=$T \cup \{[P, l, t] \xrightarrow{a} [R', l', t']\}$
15:     else if $P = Q \parallel R$ then
16:         let $T'$:=Trans($[Q, l, t]$)$\cup$ Trans($[R, l, t]$)
17:         for each $[Q, l, t] \xrightarrow{a} [Q', l', t']$ in $T'$ do
18:             $T$:=$T \cup \{[P, l, t] \xrightarrow{a} [Q' \parallel R, l', t']\}$
19:         for each $[R, l, t] \xrightarrow{a} [R', l', t']$ in $T'$ do
20:             $T$:=$T \cup \{[P, l, t] \xrightarrow{a} [Q \parallel R', l', t']\}$
21:     else if $P = Q \trianglerighteq_\delta R$ then
22:         let $T'$:=Trans($[Q, l, t]$)$\cup$ Trans($[R, l, t]$), $flag$=0
23:         for each $[Q, l, t] \xrightarrow{a} [Q', l', t']$ in $T'$ do
24:             if $t' - t < \delta$ then
25:                 $T$:=$T \cup \{[P, l, t] \xrightarrow{a} [Q' \trianglerighteq_\delta R, l', t']\}$
26:                 $flag$:=1
27:         if $flag$=0 then
28:             for each $[R, l, t] \xrightarrow{a} [R', l', t']$ in $T'$ do
29:                 if $t' - t \geq \delta$ then
30:                     $T$:=$T \cup \{[P, l, t] \xrightarrow{a} [R', l', t']\}$
31:     else if $P = Q \trianglerighteq R$ where $R = []_{i \in I} A_i \rightarrow R_i$ then
32:         let $T'$:=Trans($[Q, l, t]$)$\cup$ Trans($[R, l, t]$), $flag$:=0
33:         choose $[R, l, t] \xrightarrow{a} [R_i, l', t']$ in $T'$ with $t_0 = t'$ is
    the minimum
34:         for each $[Q, l, t] \xrightarrow{a} [Q', l', t']$ in $T'$ do
35:             if $t' < t_0$ then
36:                 $T$:=$T \cup \{[P, l, t] \xrightarrow{a} [Q' \trianglerighteq R, l', t']\}$
37:                 $flag$:=1
38:         if $flag$=0 then
39:             $T$:=$T \cup \{[P, l, t] \xrightarrow{a} [R_i, l', t_0]\}$
40:     else if $P = Q \bowtie R$ then
41:         let $T'$:=Trans($[Q, l, t]$)$\cup$ Trans($[R, l, t]$)
42:         for each $[Q, l, t] \xrightarrow{a_1} [Q', l_1, t_1]$ in $T'$ do
43:             if $a_1 \neq C!m(C?m)$ then
44:                 $T$:=$T \cup \{[P, l, t] \xrightarrow{a_1} [Q' \bowtie R, l_1, t_1]\}$
45:             else
46:                 for each $[R, l, t] \xrightarrow{a_2} [R', l_2, t_2]$ in $T'$ do
47:                     if $a_1 = C!m(C?m) \wedge a_2 = C?m(C!m) \wedge$
    $l_1 = l_2 \wedge t_1 = t_2$ then
48:                         $T$:=$T \cup \{[P, l, t] \xrightarrow{C.m} [Q' \bowtie$
    $R', l_1, t_1]\}$
49:         for each $[R, l, t] \xrightarrow{a_2} [R', l_2, t_2]$ in $T'$ do
50:             if $a_2 \neq C!m(C?m)$ then
51:                 $T$:=$T$.add($[P, l, t] \xrightarrow{a_2} [Q \bowtie R', l_2, t_2]$))
52:     return $T$
```

TA. So we often require that *spec* is a safe specification (as indicated in Section 2.2.3).

Fig. 18 gives an intuitive explanation of how our scheme works. Echoing Fig. 12, we now get a better picture of how CCSL clock can observe STeC events by taking synchronized product (×) in their TA models. STeC events synchronize with their observational counterparts in a TA of primitive constraints. The shared clocks between different primitive constraints are synchronized by taking the compositional product (⊗) between CCSL TA.

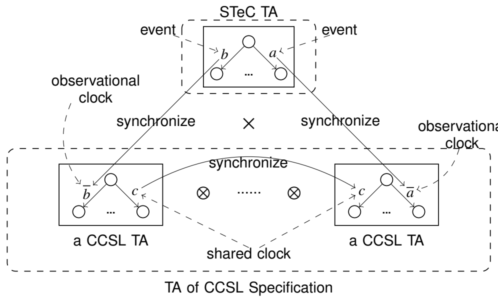


Fig. 18: A detail picture in $\mathcal{M}$

We now give a proof of Theorem 4.3.

Proof of theorem 4.3. (⇒). If $P \models_{\overline{(\cdot)}} spec$, for any $tr \in$ **Traces**$_{STeC}(P)$, from Definition 4.11 the mapping

$$f : dom(tr|_S) \rightarrow \bigcup_{a \in \mathcal{D}_{spec}} I_a$$

is a trace in some selected $TS \in \mathcal{TS}_{spec}$. According to Theorem 4.2 and Theorem 4.1, there is a corresponding trace, denoted by $\widetilde{tr}$, of tr in $\mathbf{Traces}^N_{TA}(\llbracket P \rrbracket_{STeC})$, and a set of corresponding traces, denoted as $\widetilde{f}$, of f in $\mathbf{Traces}^N_{TA}(\llbracket spec \rrbracket_{CCSL})$. When we consider trace $\widetilde{tr}$ running in $\llbracket P \rrbracket_{STeC} \times \llbracket spec \rrbracket_{CCSL}$, there must exist a trace tr' in the TA $\llbracket spec \rrbracket_{CCSL}$ that can make synchronization with $\widetilde{tr}$. Easy to see that tr' must be in $\widetilde{f}$ so it is in the TA $\llbracket spec \rrbracket_{CCSL}$. Therefore $\mathcal{M} \models \mathbf{AF}F_{\mathcal{M}}$.

(⇐). On the other direction, consider a trace $tr \in \mathbf{Traces}^N_{TA}(\mathcal{M})$. Since $\mathcal{M} \models \mathbf{AF}F_{\mathcal{M}}$ so we have $tr|_S \in \mathbf{Traces}^N_{TA}(\llbracket P \rrbracket_{STeC})$ and $tr|_{\mathcal{D}(spec)} \in \mathbf{Traces}^N_{TA}(\llbracket spec \rrbracket_{CCSL})$. From Theorem 4.2 and Theorem 4.1, there is a corresponding trace, denoted by $\widehat{tr|_{\mathcal{D}(spec)}}$, of $tr|_{\mathcal{D}(spec)}$ in a some $TS \in \mathcal{TS}_{spec}$, and a corresponding trace, denoted by $\widehat{tr|_S}$, of $tr|_S$

in $\textbf{Traces}_{STeC}(P)$. Compare $\widehat{tr|_{\mathcal{D}(spec)}}$, $\widehat{tr|_S}$ with $tr|_{\mathcal{D}(spec)}$ and $tr|_S$ we soon find out $\widehat{tr|_{\mathcal{D}(spec)}}$ is exactly the mapping

$$\widehat{tr|_{\mathcal{D}(spec)}} : dom(tr|_S) \rightarrow \bigcup_{a \in \mathcal{D}(spec)} I_a$$

we require. So $P \models_{\overline{(\cdot)}} spec$.

$\square$

4.8 Complexity Analysis of STeC/CCSL Framework

We give a general analysis of the computation complexity of STeC/CCSL framework. We prove that the verification on our proposed framework is almost as efficient as the traditional model checking techniques, since it can be proved that the complexity of the translation process stays below the bound of the complexity of model checking algorithm itself.

Given any STeC process P and specification $spec$, let $n(spec)$ be the number of constraints in $spec$. Let $St(P)$ be the set of configuration regions of P and $Tr(P)$ be the set of transitions between regions of P. Let $pt(P)$ be the set of transitions that start from P. Let $l(P)$ be the length of formula P, defined as: $l(a) = 1$ where $a \in \mathbb{AT}_{STeC}$; $l(Q * R) = 1 + l(Q) + l(R)$ where $* \in \{;, [], \trianglerighteq_\delta, \trianglerighteq, ||, \bowtie\}$. Let $sub(P)$ be the set of all subformulas of P, defined as: $sub(a) = \{a\}$ where $a \in \mathbb{AT}_{STeC}$; $sub(Q * R) = \{Q * R\} \cup sub(Q) \cup sub(R)$ where $* \in \{;, [], \trianglerighteq_\delta, \trianglerighteq, ||, \bowtie\}$.

For CCSL_2_TA($spec$), we can compute the complexity of the composition $\otimes$ of all TA R_{Cn} based on Algorithm 2 in [19] (page 8), where each R_{Cn} can be converted into a finite automata associated with state invariants. The worst case is $O(t^{n(spec)})$, where t is the maximal number of transitions in converted R_{Cn}s. It is a constant as R_{Cn} has fix number of states and transitions according to Table 3. Thus the worse case for CCSL_2_TA($spec$) is $O(n(spec) + t^{n(spec)}) = O(t^{n(spec)})$.

The complexity of STeC_2_TA(P) is a bit complex. Let the computation time of Build_TA($A, [P, l, t]$) and Trans($[P, l, t]$) be $B(P$[2]$)$ and $T(P)$ respectively. Let $P = P_1 \bowtie P_2 \bowtie ... \bowtie P_n$, $pt = max(pt(P_1), pt(P_2), ..., pt(P_n))$, $Tr = max(Tr(P_1), Tr(P_2), ..., Tr(P_n))$. $B(P)$ can be computed as:

$$\begin{aligned} B(P) &= T(P) + |pt(P)| + \Sigma_{P \to P'} B(P') \\ &= \Sigma_{P \to^* P'} T(P') + \Sigma_{P \to^* P'} |pt(P')|. \end{aligned}$$

Easy to see that $\Sigma_{P \to^* P'} |pt(P')| = |Tr(P)|$. For the first sum $\Sigma_{P \to^* P'} T(P')$, we consider two conditions of $T(P)$. If $P = Q * R$ where $* \neq \bowtie$, according to Algorithm 4, the worst condition is $Q \parallel R$, thus we have:

$$T(P) \leq T(Q \parallel R) = T(Q) + T(R) + |pt(Q)| + |pt(R)|.$$

If $P = Q \bowtie R$, we have:

$$T(P) = T(Q) + T(R) + |pt(Q)| \cdot |pt(R)|.$$

So in $B(P)$, $\Sigma_{P \to^* P'} T(P')$ are divided into two parts:

$$\begin{aligned} \Sigma_{P \to^* P'} T(P') &= \Sigma_{P_1 = Q \bowtie R} T(P_1) + \Sigma_{P_2 \neq Q \bowtie R} T(P_2) \\ &\leq \frac{n(n-1)}{2} \cdot pt^2 + \Sigma_{P_2 \neq Q \bowtie R} T(P_2) + \Sigma_{i \in 1...n} T(P_i) \\ &= \frac{n(n-1)}{2} \cdot pt^2 + \Sigma_{P'' \neq Q \bowtie R} T(P''). \end{aligned}$$

In $\Sigma_{P'' \neq Q \bowtie R} T(P'')$, the same P'' only needs to be computed for once. We expand $\Sigma_{P'' \neq Q \bowtie R} T(P'')$ and get:

$$\begin{aligned} \Sigma_{P'' \neq Q \bowtie R} T(P'') &= \Sigma_{S \in sub(P), S \neq Q \bowtie R} T(S) \\ &= \Sigma_{a \in sub(P), a \in \textbf{At}(P)} T(a) + \Sigma_{S \in sub(P), S \neq Q \bowtie R} |pt(S)| \\ &= \Sigma_{a \in sub(P), a \in \textbf{At}(P)} T(a) + \\ &\quad \Sigma_{i \in 1...n} \Sigma_{S \in sub(P_i), S \neq Q \bowtie R} |pt(S)| \\ &\leq |l(P)| + \Sigma_{i \in i...n} |Tr(P_i)| = |l(P)| + n \cdot Tr. \end{aligned}$$

Note that here $\Sigma_{S \in sub(P_i), S \neq Q \bowtie R} |pt(S)| \leq \Sigma_{P_i \to^* S} |pt(S)|$ for $i \in 1...n$. Usually, we can think $|l(P)| \leq |St(P)|$. In the worst case, $Tr(P) = Tr^n$. Thus

$$\begin{aligned} \frac{n(n-1)}{2} \cdot pt^2 + n \cdot Tr &\leq n^2 \cdot Tr^2 + n \cdot Tr \\ &\in O(Tr^n) = O(Tr(P)). \end{aligned}$$

So we have

$$B(P) \in O(2 \cdot |Tr(P)| + |St(P)|) = O(|Tr(P)| + |St(P)|).$$

So the upper bound of STeC_2_TA(P) is $O(|Tr(P)| + |St(P)|)$.

According to [12] we know that the complexity of our model checking algorithm for STeC/CCSL framework is $O(|\mathbf{AF}F_{\mathcal{M}}| \cdot (|St(P)| \cdot |c^{n(spec)}| + |Tr(P)| \cdot |t^{n(spec)}|))$, where let c be the maximum number of states of R_{Cn}s. Thus the total complexity of STeC/CCSL framework is the complexity of the translations adding the complexity of model checking, which is: $O(|t^{n(spec)}| + |Tr(P)| + |St(P)| + |\mathbf{AF}F_{\mathcal{M}}| \cdot (|St(P)| \cdot |c^{n(spec)}| + |Tr(P)| \cdot |t^{n(spec)}|)) = O(|St(P)| \cdot |c^{n(spec)}| + |Tr(P)| \cdot |t^{n(spec)}|)$. The complexity of translation procedures stay within the bound of the complexity of the traditional model checking algorithm.

4.9 Model Checking Framework for Infinite Traces

In previous sections we only deal with processes whose traces are finite, due to the fact that STeC has no recursions defined

[2] We consider P instead of $[P, l, t]$ since the configuration does not have impact on the complexity of the function, and can be easily acquired according to the context of the algorithm. The same as belows, for example in $\Sigma_{P \to^* P'} T(P')$, P' starts from the l, t according to its predecessor P.

in its syntax. However, our theory also can be applied to models with infinite traces, with only a few modifications to make.

In TA theory, according to [23], an infinite trace

$$tr = \langle a_1, t_1\rangle\langle a_2, t_2\rangle...\langle a_n, t_n\rangle...$$

is in TA $A = (Q, \Sigma, C, i, Ed, I, AP, L, F)$ iff there is a transition

$$\langle i, v_A^i\rangle \overset{\emptyset}{\rightarrow}{}^* \xrightarrow{a_1,t_1} \langle q_1, v_1\rangle \overset{\emptyset}{\rightarrow}{}^* \xrightarrow{a_2,t_2} ... \overset{\emptyset}{\rightarrow}{}^* \xrightarrow{a_n,t_n} \langle q_n, v_n\rangle...$$

in A such that for any $N \in \mathbb{N}^+$, there exists a $i \geq N$ such that $q_i \in F$.

In a time structure TS, an infinite trace is defined just as in Section 2.2.3.

Suppose that we have a normal process P of some modelling language called PA, with the set of finite or infinite traces $\mathbf{Traces}_{PA}(P)$, and somehow we can translate it into its equivalent TA $[\![P]\!]_{PA}$ and propose a new definition and theorem similar to Definition 4.18 and Theorem 4.2. Then all the other definitions and theorems of our theory proposed above still work in the case of infinite traces by just replacing $\mathbf{Traces}^N_{TA}(\cdot)$, $\mathbf{Traces}_{TS}(\cdot)$ and $\mathbf{Traces}_{STeC}(\cdot)$ with their counterparts for infinite traces, except for the Theorem 4.3, where the model checking scheme combining PA and CCSL can be dealt with the satisfaction

$$\mathcal{M} = [\![P]\!]_{PA} \times [\![spec]\!]_{CCSL} \models \mathbf{AG}(\mathbf{AF}F_{\mathcal{M}}).$$

$\mathbf{AG}(\mathbf{AF}F_{\mathcal{M}})$ means that for each path in $\mathcal{M}$, it will eventually pass the state in $F_{\mathcal{M}}$ for infinitely many times.

In some model checking tools, like UPPAAL, the CTL formula where the path quantifiers are nested is not allowed (e.g., $\mathbf{AG}(\mathbf{AF}F_{\mathcal{M}})$). To solve this problem we can take an alternative approach by checking

$$\mathbf{AG}(\neg\mathbf{deadlock} \text{ in } \mathcal{M})$$

which means that 'for every path, and every state in each path, there is no deadlock in $\mathcal{M}$'. **deadlock** is a special expression in UPPAAL, to judge if in a state, deadlock can happen. Though they are not equivalent, but in most cases[3], we can use the latter in place of the former. And this is the approach used in [17] for checking safety properties in UPPAAL. However, this formula actually can not replace the formula $\mathbf{AF}F_{\mathcal{M}}$ in the case of finite traces, since in a TA where there is no infinite trace there is always a deadlock. This is also why we do not take this approach for our purpose in this paper.

[3] To be exact, when there is no deadlock in $[\![P]\!]_{PA}$

5 Model Checking in UPPAAL

In this section, we apply our model checking scheme of STeC/CCSL in UPPAAL.

UPPAAL is a verification tool based on TA theory, its modelling language, as an extension of TA with discrete data, offers additional features such as bounded integer variables and urgency states. A UPPAAL system is a network of TA which run concurrently with handshakes of each other through channels. UPPAAL TA offers many variable types besides clocks and channels, like bounded integers, booleans, etc. Channels are for synchronizing parallel automatons. Guards have the same meaning as in TA. There are update functions on edges of UPPAAL TA, which update the values of variables like clocks and integers. [16] gives a full guide.

From the theory proposed last section, given a STeC process P, a CCSL specification $spec$ and a mapping $\overline{(\cdot)}$, we can check $P \models_{\overline{(\cdot)}} spec$ in UPPAAL following several steps listed below:

1) Translate $spec$ into UPPAAL TA, denoted by $[\![spec]\!]_{UPL}$, based on Definition 4.15.
2) Let $P = P_1 \bowtie P_2 \bowtie ... \bowtie P_n$. Translate each P_i into UPPAAL TA, denoted by $[\![P_i]\!]_{UPL}$, based on Theorem 4.2, Definition 4.19 and Proposition 4.1.
3) $[\![spec]\!]_{UPL}$ and each $[\![P_i]\!]_{UPL}$ form a network of UPPAAL TA in UPPAAL, for this network, we check

$$[\![P_1]\!]_{UPL}, ..., [\![P_n]\!]_{UPL}, [\![spec]\!]_{UPL} \models \bigwedge_{i=1...n} A<> P_i.Accept,$$

based on Theorem 4.3.

Here '$A<>$' is a CTL operator in UPPAAL, it has the same meaning as **AF** in Theorem 4.3. '$P_i.Accept$' corresponds the accepting states in $[\![P_i]\!]_{UPL}$. Since in $[\![spec]\!]_{UPL}$ all states are accepting states, so actually we do not need to consider them in the specification.

From the list above we see that we translate $spec$ into $[\![spec]\!]_{UPL}$ as one UPPAAL TA. Readers might wonder why we do not consider translating each $Cn \in spec$ instead, and let UPPAAL do the lazy-evaluation of the composition at runtime, just like what we do for STeC process? The answer is, $[\![Cn]\!]_{CCSL}$ might be a TA with infinite states, though we can make the composition $\otimes$ theoretically to get a finite TA of $spec$ by lazy-evaluation [19], we can not express an infinite TA in UPPAAL!

When we carry out the translation we need to make some modifications since the UPPAAL TA is different from the general TA model. One of the main differences is that in UPPAAL TA, there is only one type of channel, so we must find a way to distinguish the composition $\odot$ between STeC TA and the composition $\times$ between a STeC TA and a CCSL observer. We list the modifications as below:

1) For each event $C!m$ ($C?m$) in $[\![P_i]\!]_{STeC}$, we have the channel $C\_m!$ ($C\_m?$) in $[\![P_i]\!]_{UPL}$.
2) For each event a (not in the form of $C.m$) in $[\![P]\!]_{STeC}$, and its corresponding observational event $\overline{a}$ in $[\![spec]\!]_{CCSL}$, we have a pair $t\_a!$, $t\_a?$ in $[\![P_i]\!]_{UPL}$ and $[\![spec]\!]_{UPL}$ respectively.
3) If an event $C.m$ in $[\![P]\!]_{STeC}$ is observed by CCSL constraints, in all $[\![P_i]\!]_{UPL}$s that contains $C\_m!$, we add an urgent transition right after it with a channel $t\_m!$ being triggered on it. And we have a channel $t\_m?$ in $[\![spec]\!]_{UPL}$.
4) If an event a is either an unobservable event in $[\![P]\!]_{STeC}$ or a generated event in $[\![spec]\!]_{CCSL}$, no labels assigned to their corresponding transitions in UPPAAL TA.

Table 4 gives a graphical illustration for clause 1), 2), 3), 4) given above respectively. '$\cup$' means the urgent state in UPPAAL, a type of state from which any transitions must immediately happen without consuming time.

Clause	Transitions in STeC/CCSL	Transitions in UPPAAL
1)	$\cdots \xrightarrow{C!m/C?m} \cdots$	$\cdots \xrightarrow{C\_m!/C\_m?} \cdots$
2)	$\cdots \xrightarrow{a/\overline{a}} \cdots$	$\cdots \xrightarrow{t\_a!/t\_a?} \cdots$
3)	$\cdots \xrightarrow{C.m} \cdots$	$\cdots \xrightarrow{C\_m!} \textcircled{$\cup$} \xrightarrow{t_m!} \cdots$
		$\cdots \xrightarrow{t\_m?} \cdots$
4)	$\cdots \xrightarrow{a} \cdots$	$\cdots \longrightarrow \cdots$

Table 4: The modifications from TA to UPPAAL TA

From Theorem 4.3 in fact $[\![spec]\!]_{CCSL}$ can only observe the interaction $C.m$ in $[\![P]\!]_{STeC}$, rather than $C!m$ in some $[\![P_i]\!]_{STeC}$. But in 3) above, $[\![spec]\!]_{UPL}$ observes the channel $C\_m!$ instead, since we consider the network $\{[\![P_i]\!]_{UPL}\}_{i=1...n}$ in UPPAAL, not $[\![P]\!]_{UPL}$ as a whole, so there is no $C.m$ in UPPAAL TA. These two means are equivalent because for CCSL, observing $C!m$ or $C.m$ makes no difference.

6 Case Study—Verifying IRCS System in UPPAAL

We apply the techniques introduced in previous sections to the verification of IRCS system (in Section 3) in UPPAAL.

We consider three safety properties.

Property 1: 'The train always passes after the gate is closed and before the gate is opened.'

Its CCSL specification is as follows:

$$spec_1 = \{c_{close} \prec c_{pass}, c_{pass} \prec c_{open},\ c_{close}\ alternatesWith\ c_{open}\}$$

with the observational mapping $\overline{(\cdot)} = \{Close \mapsto close, Pass \mapsto pass, Open \mapsto open\}$.

Property 2: 'If the train stops, it must stop at a time point that is 50s later since the train approaches.'

Its CCSL specification is as follows:

$$spec_2 = \{c_{mayStop} = c_{appr}\ delay\ 50\ on\ IdealClk,\ c_{stop} \subseteq c_{mayStop}\}$$

with the observational mapping $\overline{(\cdot)} = \{C.Appr \mapsto appr, Stop \mapsto stop\}$. $c_{mayStop}$ is a generated clock.

Property 3: 'Once the train approaches, the gate shall close in less than 21 seconds.'

Its CCSL specification is as follows:

$$spec_3 = \{c_{appr\_delay} = c_{appr}\ delay\ 21\ on\ IdealClk,\ c_{appr} \prec c_{close}, c_{close} \prec c_{appr\_delay},\ c_{appr}\ alternatesWith\ c_{appr\_delay}\}$$

with the observational mapping $\overline{(\cdot)} = \{C.Appr \mapsto appr, Close \mapsto close, Closed \mapsto close\}$. c_{appr_delay} is a generated clock. Note that in this specification event *Close*, *Closed* are both observed by the clock c_{close}.

The three specifications can be translated into UPPAAL TA as in the last section shows. Fig. 19, 20, 20 show the UPPAAL TA of $spec_1$, $spec_2$ and $spec_3$ respectively. The UPPAAL TA of the process T and G are different since we have a different mapping $\overline{(\cdot)}$ for each property. Fig. 22 gives an example of train agent $[\![T]\!]_{UPL}$ for **Property 1**.

We verify if these properties hold in UPPAAL by checking two CTL formulas in each case:

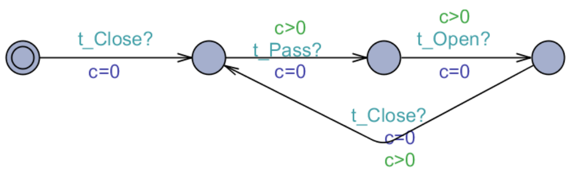


Fig. 19: UPPAAL Model of $spec_1$

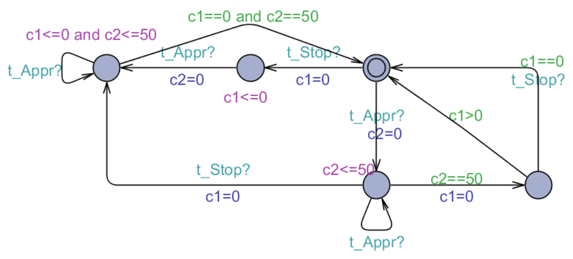


Fig. 20: UPPAAL Model of $spec_2$

$$A <> \ T.Accept,$$
$$A <> \ G.Accept.$$

The final results are listed in Table 5, from which we can see that **Property 3** does not hold since if the gate is open, it will close itself exactly 21 seconds after the train approaches.

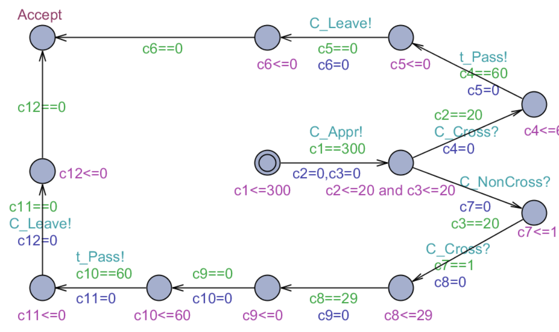


Fig. 22: UPPAAL Model of $[\![T]\!]_{UPL}$ for Property 1

Property	CTL Formula	Result
Property 1	$A <> \ T.Accept$	yes
	$A <> \ G.Accept$	yes
Property 2	$A <> \ T.Accept$	yes
	$A <> \ G.Accept$	yes
Property 3	$A <> \ T.Accept$	no
	$A <> \ G.Accept$	no

Table 5: Verification results in UPPAAL

7 Conclusion and Future Work

In this paper, we propose a STeC/CCSL framework for spatio-temporal systems and explore its verification technique. We build a theory to connect STeC and CCSL in their theories and propose a model checking framework for verifying CCSL properties in STeC models. We propose the theory and algorithms to translate STeC and CCSL into TA and analyze their time complexities. At last we propose the translation from STeC/CCSL into UPPAAL TA and as a case study we show how to verify the IRCS system in UPPAAL.

For the furure work, we mainly have a prospect in two directions:

1. In this paper, we use CCSL to express properties concerning time, but not locations. For some system where spaces are of first important, for example, in a track station system, the tracks can be in four directions, they can cross each other forming a complex track network. In such a system, to give a description of relationships between the locations of trains is important, we may need a kind of primitive language to describe geographical information, not just simple relations like ‘precedence’ and ‘subset’. So our future work may focus on the extension of CCSL to describe geographical relationships of spaces.
2. Since our language is for ITSs, a kind of systems that very much rely on its environment, we may want to consider the uncertainty of environment in systems. For example, in our case study, the traffic condition on the crossing is an uncertain factor of our system, but we ignore it by simply using the internal choice ‘[]’. The next step might focus on the extension of our model to a stochastic model (to extend both STeC and CCSL to a probabilistic version), where uncertain factors can be considered. For the verification of such models in practice, we may need the ‘probabilistic’ version of UPPAAL tool—UPPAAL SMC.

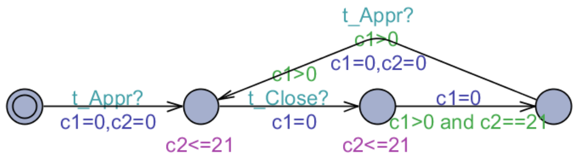


Fig. 21: UPPAAL Model of $spec_3$

8 Related Works

Several approaches have been proposed combining CCSL as a specification language with modelling languages in verification of real-time systems.

MARTE [11], as an extension of UML [26], is a general model-based language for modeling and analysis of real-time and embedded systems. The modelling parts provide support required from specification to detailed design of real-time and embedded features. The analyzing parts offer facilities to annotate models with information required to perform specific analysis. CCSL has recently been introduced into MARTE for testing and run-time verification of crucial safety properties of models [15]. However, MARTE is designed as an unified modeling language for developing large systems in practice. Due to the large size and complexity of models it is hard (and sometimes unrealistic) to directly apply model checking techniques with CCSL. To do it people need to extract behaviour of concern from MARTE models and encode it into a more abstract formal language such as CCS or CSP. Contrary to MARTE, STeC is a formal langauge and aims at modelling systems at a high-level. Its simpler semantics make it easier for composing several distributed agents and analyze the interactive behaviour between them. A trade-off approach may be using STeC to capture an abstract specification while using MARTE to handle the detail designs.

Timed-pNets [27] is a communication behavioural semantic model for distributed systems. Its model is a tree style structure with leaves as distributed agents expressed as LTSs and nodes as communication channels between agents. It supports a hierarchical design pattern either in a top-down or a bottom-up fashion. In Timed-pNets CCSL has been adopted for giving specification of causality relations between events on each leave/node. The correctness of specification on each leave/node can be checked by running simulations on an automated tool called Timed Square [28]. The verification technique used there is based on simulations, rather than our approach which is based on model checking. But we believe our proposed model checking framework about STeC and CCSL can be easily adapted to Timed-pNets.

[29] proposed a clock-based modeling language for hybrid systems. As examples, it was used for modeling several CPSs including IRCS discussed in this paper [30]. This model is based on a hybrid clock theory that can be seen as an extension to CCSL with several clock operators for characterizing dynamic features of hybrid systems. It is a novel approach using clocks and clock relations to model systems, which makes it easy to extract the specification of the control component from the specification of the total system and the desire behaviour of the physical component. Nevertheless, using clock expressions as a formal model leads to the lost of ability to express the compositionality of systems, which is provided by the combinators of formal languages like STeC. It still remains to be seen that such models can be equipped with a strong verification support like automatic proving or model checking. Comparing with our approach, we prefer to use clock theory to express safety properties, rather than modeling the whole system.

[31] investigated simulating and checking an Esterel [32] program against a CCSL specification. CCSL constraints were encoded into Esterel code as an observer of an Esterel program, and model checking takes place in Esterel Studio. This approach is much similar to ours. The only difference is that an Esterel observer for a given clock relation programs the negation of the primitive constraint associated with this relation in order to check possible violations. While in this paper we encode a clock relation into an observer that just behaves as it. The violation checking is by the CTL formula we give in Theorem 4.3. Compared with Esterel, which is a synchronous language for reactive systems, we focus on the verification of CCSL in an asynchronous language—STeC, whose synchronization mechanism between events is different from Esterel.

Another work linking CCSL and an asynchronous language was proposed in [33], where a CCSL specification was encoded into a Promela model. Promela [34] is an asynchronous modelling language for modelling and verifying concurrent processes (e.g., distributed systems). Not like STeC, in Promela both synchronous and asynchronous communication via message can be defined. Promela supports formal verifications by model checker SPIN. By encoding CCSL into a Promela model, CCSL specification can thus be checked by SPIN. However since Promela/SPIN is an untimed verification framework so only logical time issues in CCSL has been taken cared. In this paper we stress both logical and chronometric time issues in CCSL and encode it into a more refined TA model. The TA model equipped with real-time information allows us to specify physical time constraints which is important in analyzing high-level abstract models of ITSs.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (No.61370100 and No.61321064), Shanghai Knowledge Service Platform Project (No.ZF1213), Shanghai Municipal Science and Technology Commission Project (No.14511100400) and Defense Industrial Technology Development Program JCKY 2016212B004-2.

Specially thank Professor Hengyang Wu, who gave us many usable proposals and found out many syntax errors in this paper. Also thank all reviewers for their time to carefully read this paper and give their valuable questions and suggestions.

References

1. Y. Chen, "STeC: A location-triggered specification language for real-time systems." in *ISORC Workshops*. IEEE, 2012, pp. 1–6.
2. H. Wu, Y. Chen, and M. Zhang, "On denotational semantics of spatial-temporal consistency language - STeC," in *TASE*. IEEE, 2013, pp. 113–120.
3. C. A. R. Hoare, "Communicating sequential processes," *Commun. ACM*, vol. 21, no. 8, pp. 666–677, Aug. 1978.
4. R. Milner, *A Calculus of Communicating Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1982.
5. G. M. Reed and A. W. Roscoe, "A timed model for communicating sequential processes," *Theor. Comput. Sci.*, vol. 58, no. 1-3, pp. 249–261, Jun. 1988.
6. W. Yi, "CCS + time = an interleaving model for real time systems." in *ICALP*, ser. Lecture Notes in Computer Science, J. L. Albert, B. Monien, and M. RodrÍłguez-Artalejo, Eds., vol. 510. Springer, 1991, pp. 217–228.
7. L. Cardelli and A. D. Gordon, "Mobile ambients." *Electr. Notes Theor. Comput. Sci.*, vol. 10, pp. 198–201, 1997.
8. R. Milner, J. Parrow, and D. Walker, "A calculus of mobile processes, i," *Inf. Comput.*, vol. 100, no. 1, pp. 1–40, Sep. 1992.
9. C. André and F. Mallet, "Clock constraints in UML/MARTE CCSL," Research Report RR-6540, 2008.
10. L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.
11. "UML profile for MARTE: Modeling and analysis of real-time embedded systems," 2011.
12. C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.
13. "IEEE standard for property specification language (PSL)," *IEEE Std 1850-2005*, pp. 0_1–143, 2005.
14. R. Gascon, F. Mallet, and J. Deantoni, "Logical time and temporal logics: Comparing uml MARTE/CCSL and PSL," in *2011 Eighteenth International Symposium on Temporal Representation and Reasoning*, Sept 2011, pp. 141–148.
15. C. André, F. Mallet, and R. De Simone, "Modeling time(s)," in *Proceedings of the 10th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 559–573.
16. G. Behrmann, R. David, and K. G. Larsen, "A tutorial on UPPAAL." Springer, 2004, pp. 200–236.
17. J. Suryadevara, C. Seceleanu, F. Mallet, and P. Pettersson, "Verifying MARTE/CCSL mode behaviors using UPPAAL," in *11th International Conference on Software Engineering and Formal Methods*, September 2013.
18. Y. Zhang, F. Mallet, and Y. Chen, "Timed automata semantics of spatial-temporal consistency language STeC," in *2014 Theoretical Aspects of Software Engineering Conference, TASE 2014, Changsha, China, September 1-3, 2014*. IEEE, 2014, pp. 201–208.
19. F. Mallet and R. de Simone, "Correctness issues on marte/ccsl constraints," *Science of Computer Programming*, no. 0, pp. –, 2015.
20. C. André, "Syntax and semantics of the clock constraint specification language (CCSL)," Research Report RR-6925, 2009.
21. F. Mallet, *Logical Time @ Work for the Modeling and Analysis of Embedded Systems*. Lambert Academic Publisher LAP, 2011, iSBN: 978-3-8433-9388-1.
22. F. Mallet, J.-V. Millo, and R. de Simone, "Safe CCSL specifications and marked graphs," in *MEMOCODE*. IEEE, 2013, pp. 157–166.
23. R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, Apr. 1994.
24. F. Mallet, "Automatic generation of observers from MARTE/CCSL," in *Rapid System Prototyping (RSP), 2012 23rd IEEE International Symposium on*, Oct 2012, pp. 86–92.
25. M. Huth and M. Ryan, "Logic in computer science: Modelling and reasoning about systems," 1999.
26. J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, The (2Nd Edition)*. Pearson Higher Education, 2004.
27. Y. Chen, Y. Chen, and E. Madelaine, "Timed-pnets: a communication behavioural semantic model for distributed systems," *Frontiers of Computer Science*, vol. 9, no. 1, pp. 87–110, 2015.
28. J. Deantoni and F. Mallet, "Timesquare: Treat your models with logical time," in *TOOLS (50)*, ser. Lecture Notes in Computer Science, C. A. Furia and S. Nanz, Eds., vol. 7304. Springer, 2012, pp. 34–41.
29. H. Jifeng, *A Clock-Based Framework for Construction of Hybrid Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 22–41.
30. B. Xu and L. Zhang, "Formal specification of cyber physical systems: Three case studies based on clock theory," in *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, Aug 2013, pp. 804–811.
31. C. André and F. Mallet, "Specification and verification of time requirements with CCSL and Esterel," in *Proceedings of the 2009 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, ser. LCTES '09. New York, NY, USA: ACM, 2009, pp. 167–176.
32. G. Berry and G. Gonthier, "The esterel synchronous programming language: Design, semantics, implementation," *Sci. Comput. Program.*, vol. 19, no. 2, pp. 87–152, Nov. 1992.
33. L. Yin, F. Mallet, and J. Liu, "Verification of MARTE/CCSL time requirements in Promela/Spin," in *Proceedings of the 2011 16th IEEE International Conference on Engineering of Complex Computer Systems*, ser. ICECCS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 65–74.
34. G. J. Holzmann, "The model checker Spin," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. 23, no. 5, pp. 279–295, 1997.

Yuanrui Zhang is a Phd student in the School of Computer Science and Software Engineering, East China Normal University, China. He received his BS degree in pure and applied mathematics, and his MS degree in computer science. His current research interests are verification of real-time systems, interactive proving theory and its application, formal modelling and verification of cyber-physical systems. Now he is working on verification of CCSL specifications using logical approach.

Frédéric Mallet is a full Professor in the Informatics Department , University of Nice Sophia Antipolis, France. He is also a member of the KAIROS team-project, a joint team between the I3S laboratory (UMR CNRS) and the INRIA research center Sophia-Antipolis Méditerranée. His current research interests focus on modelling, simulation and verification of real-time and embedded systems, model-driven engineering, parallel and distributed computing, computer architecture, modelling and verification of cyber-physical systems. Professor Mallet is one of co-inventors of CCSL language and a contributor to Time Square, a simulation tool for CCSL. He was deeply involved as a voting member of MARTE RTF for the definition of the Time and allocation sub-profiles.

Yixiang Chen is a full Professor in the School of Computer Science and Software Engineering, East China Normal University, China. Where he is coordinating trustworthy software, Internet of things and Human-Cyber-Physical System related research activities. Professor Chen is the director of the MoE Engineering Research Center for Software/Hardware Co-design Technology and Application. He is a Vice-Chairman of Technical Committee for Embedded System China Computer Federation.