

# Formal Verification of Dynamic and Stochastic Behaviors for Automotive Systems

Li Huang\*, Tian Liang\* and Eun-Young Kang†

\*School of Data & Computer Science, Sun Yat-Sen University, Guangzhou, China  
{huangl223, liangt37}@mail2.sysu.edu.cn

†The Maersk Mc-Kinney Moller Institute, University of Southern Denmark, Denmark  
eyk@mmmi.sdu.dk

**Abstract**—Formal analysis of functional and non-functional requirements is crucial in automotive systems. The behaviors of those systems often rely on complex dynamics as well as on stochastic behaviors. We have proposed a probabilistic extension of Clock Constraint Specification Language, called PrCCSL, for specification of (non)-functional requirements and proved the correctness of requirements by mapping the semantics of the specifications into UPPAAL models. Previous work is extended in this paper by including an extension of PrCCSL, called PrCCSL\*, for specification of stochastic and dynamic system behaviors, as well as complex requirements related to multiple events. To formally analyze the system behaviors/requirements specified in PrCCSL\*, the PrCCSL\* specifications are translated into stochastic UPPAAL models for formal verification. We implement an automatic translation tool, namely ProTL, which can also perform formal analysis on PrCCSL\* specifications using UPPAAL-SMC as an analysis backend. Our approach is demonstrated on two automotive systems case studies.

**Index Terms**—Automotive Systems, PrCCSL\*, UPPAAL-SMC, ProTL

## I. INTRODUCTION

Model-based development (MBD) is rigorously applied in automotive systems in which the software controllers continuously interact with physical environments. The behaviors of automotive systems often involve complex hybrid dynamics as well as stochastic characteristics. Formal verification and validation (V&V) technologies are indispensable and highly recommended for development of safe and reliable automotive systems [1], [2]. *Statistical model checking (SMC)* techniques have been proposed [3]–[5] to address the reliability of hybrid systems associated with the stochastic and non-linear dynamical features. These techniques for fully stochastic models validate probabilistic properties of controllers in given environments under uncertainty.

EAST-ADL [6], [7] is a concrete example of MBD approach for architectural modeling of automotive systems. The latest release of EAST-ADL has adopted the time model proposed in the Timing Augmented Description Language (TADL2) [8], which expresses and composes the basic timing constraints, i.e., repetition rates, end-to-end delays, and synchronization constraints. TADL2 specializes the time model of MARTE, the UML profile for Modeling and Analysis of Real-Time and Embedded systems [9]. MARTE provides CCSL [10], [11], which is the clock constraint specification language for specification of temporal constraints and functional causality

properties [12]. In CCSL, time can be either chronometric (i.e., associated with physical time) or logical (i.e., related to events occurrences), which are represented by dense clocks and logical clocks, respectively. Dense clocks can be multi-form and attached with various rates, allowing to express the evolution of time-related quantities with different units, e.g., temperature degree and crankshaft angle [13]. The discrete and dense clocks in CCSL enable the specifications of hybrid system behaviors that incorporate both discrete phenomena and continuous dynamics [14].

We have previously proposed a probabilistic extension of CCSL, i.e., PrCCSL [15], [16], to formally specify (non)-functional properties in weakly-hard (WH) context [17], i.e., a bounded number of constraints violations would not lead to system failures when the results of the violations are negligible. However, PrCCSL still lacks expressivity for describing critical system behaviors regarding to: 1) Continuous dynamic behaviors of physical plant, which are typically modeled by ordinary differential equations (ODE) containing functions and derivatives; 2) Discontinuous activities triggered by discrete events. For example, the velocity of an automotive system undertakes instantaneous changes when collisions occur; 3) Stochastic time spans of continuous activities (e.g., execution time of a component, response time for a spontaneous failure); 4) Nondeterministic behaviors regarding to exclusive activities. For instance, during the execution of an automotive system, multiple sensors can forward messages to the controller simultaneously and the order for the controller to receive the messages is nondeterministic.

Supporting formal specifications of dynamic and stochastic behaviors is crucial for effective analysis of automotive systems. In this paper, we propose an extension of PrCCSL, called PrCCSL\*, to enable the specification of the aforementioned system behaviors. To allow the analysis of system behaviors specified in PrCCSL\*, the PrCCSL\* specifications are translated into UPPAAL-SMC models for formal verification. Furthermore, in PrCCSL, requirements are specified as binary clock constraints that describe temporal and causal relations between two events. To support the analysis of complex requirements associated with multiple events, we extend the binary clock constraints into n-ary constraints and provide the corresponding translation patterns in UPPAAL-SMC. The automatic translation from PrCCSL\* specifications

(of system behaviors/requirements) to UPPAAL-SMC models is implemented in a tool called ProTL (Probabilistic CCSL Translator). Formal analysis of the PrCCSL\* specifications can be performed by ProTL using UPPAAL-SMC as an analysis backend. Our approach is demonstrated on two automotive systems case studies.

The paper is organized as follows: Sec. II presents an overview of PrCCSL and UPPAAL-SMC. The definition of PrCCSL\* is presented in Sec. III. Sec. IV describes the mapping rules from PrCCSL\* specifications into stochastic UPPAAL models. The applicability of our approach is demonstrated by performing verification on two automotive systems in Sec. V. Sec. VI and Sec. VII present related works and conclusion.

## II. PRELIMINARY

In this section, we first introduce the notations and formalism in PrCCSL that are employed in the rest of the paper. Then, we give an overview of UPPAAL-SMC, which is utilized as the analysis backend of ProTL.

**PrCCSL** [15] is a probabilistic extension of CCSL (Clock Constraint Specification Language) [10], which specifies temporal and causal constraints with stochastic characteristics in weakly-hard context [17]. In PrCCSL, a clock is a basic element that represents a sequence of (possibly infinite) instants. A clock in PrCCSL can be either dense or discrete/logical. A dense clock represents physical time, which is considered as a continuous and unbounded progression of time instants. The physical time is represented by a dense clock with a base unit. *idealClk* is a predefined dense clock in CCSL, which represents the ideal physical time whose unit is second. A dense clock can be discretized into a discrete/logical clock. For example, a clock *ms* can be defined based on *idealClock*: *ms* = *idealClock* discretizedBy 0.001, i.e., *ms* ticks every 1 millisecond. A logical clock represents an *event* and the clock instants correspond to the event occurrences. A clock represents an instance of `ClockType` characterized by a set of attributes. The keyword `DenseClockType` (`DiscreteClockType`) defines new dense (discrete) clock types.

PrCCSL provides two types of clock constraints to describe the occurrences of different events (logical clocks), i.e., clock *expressions* and clock *relations*. *Expressions* derive new clocks from the already defined clocks. Table I presents a set of clock

*expressions*, where *c1, c2, ref, res, base* are clocks and “ $\triangleq$ ” means “is defined as”. We only list a subset of clock *expressions* here due to page limit and the full set of *expressions* can be found in [10].

A clock *relation* limits the occurrences among different clocks/events, which are defined based on `run` and `history`. A `run` corresponds to an execution of the system model where the clocks tick/progress. The `history` of a clock *c* represents the number of times *c* has ticked currently. A probabilistic *relation* in PrCCSL is satisfied if and only if the probability of the *relation* constraint being satisfied is greater than or equal to the probability threshold  $p \in [0, 1]$ . Given  $k$  runs =  $\{R_1, \dots, R_k\}$ , the probabilistic *relations* in PrCCSL, including `subclock`, `coincidence`, `exclusion`, `precedence` and `causality` are defined in Table II.

**UPPAAL-SMC** [18] performs the probabilistic analysis of properties by monitoring simulations of complex hybrid systems in a given stochastic environment and using results from the statistics to determine whether the system satisfies the property with some degree of confidence. Its clocks evolve with various rates, which are specified with *ordinary differential equations* (ODE). UPPAAL-SMC provides a number of queries related to the stochastic interpretation of Timed Automata (STA) [5] and they are as follows, where  $N$  and *bound* indicate the number of simulations to be performed and the time bound on the simulations respectively: 1) *Probability Estimation* estimates the probability of a requirement property  $\phi$  being satisfied for a given STA model within the time bound:  $Pr[bound] \phi$ . 2) *Hypothesis Testing* checks if the probability of  $\phi$  being satisfied is larger than or equal to a certain probability  $P_0$ :  $Pr[bound] \phi \geq P_0$ . 3) *Probability Comparison* compares the probabilities of two properties being satisfied in certain time bounds:  $Pr[bound_1] \phi_1 \geq Pr[bound_2] \phi_2$ . 4) *Expected Value* evaluates the minimal or maximal value of a clock or an integer value while UPPAAL-SMC checks the STA model:  $E[bound; N](min : \phi)$  or  $E[bound; N](max : \phi)$ . 5) *Simulations*: UPPAAL-SMC runs  $N$  simulations on the STA model and monitors  $k$  (state-based) properties/expressions  $\phi_1, \dots, \phi_k$  along the simulations within simulation bound *bound*: *simulate*  $N [\leq bound] \{\phi_1, \dots, \phi_k\}$ .

## III. EXTENSION OF PRCCSL

To improve the expressiveness of PrCCSL for specifying dynamic and stochastic system behaviors (i.e., the four

TABLE I: Clock Expressions

Expression	Notation	Remarks
ITE (if-then-else)	$res \triangleq b ? c1 : c2$	<i>res</i> behaves either as <i>c1</i> or as <i>c2</i> based on the value of the boolean variable <i>b</i> .
DelayFor	$res \triangleq ref (d) \rightsquigarrow base$	$\forall i \in \mathbb{N}^+$ , the $i^{th}$ tick of <i>res</i> corresponds to the $i^{th}$ tick of <i>ref</i> delayed for <i>d</i> ticks (or units) of <i>base</i> .
FilterBy	$res \triangleq base \blacktriangledown u(v)$	<i>res</i> is generated by filtering the instants of <i>base</i> based on a binary word $w=u(v)$ , where <i>u</i> is <i>prefix</i> and <i>v</i> is <i>period</i> . “( <i>v</i> )” denotes the infinite repetition of <i>v</i> . $\forall i \in \mathbb{N}^+$ , if the $i^{th}$ bit of <i>w</i> is 1, <i>res</i> ticks at the $i^{th}$ tick of <i>base</i> .
PeriodicOn	$res \triangleq base \propto n$	Any two consecutive instants of <i>res</i> are separated by <i>n</i> instants of <i>base</i> .
Infimum	$res \triangleq c1 \wedge c2$	<i>res</i> is the slowest clock faster than <i>c1</i> and <i>c2</i> .
Supremum	$res \triangleq c1 \vee c2$	<i>res</i> is the fastest clock slower than <i>c1</i> and <i>c2</i> .

TABLE II: Relations in PrCCSL

Relation	Notation	Remarks
Probabilistic Subclock	$c1 \sqsubseteq_p c2$	$c1 \sqsubseteq_p c2 \iff Pr[c1 \sqsubseteq c2] \geq p$ , where $Pr[c1 \sqsubseteq c2] = \frac{1}{k} \sum_{j=1}^k \{R_j \models c1 \sqsubseteq c2\}$ is the ratio of runs that satisfies the relation out of $k$ runs. A run $R_j$ satisfies the subclock relation $c1 \sqsubseteq c2$ if “when $c1$ ticks, $c2$ must tick” always holds in $R_j$ .
Probabilistic Coincidence	$c1 \equiv_p c2$	$c1 \equiv_p c2 \iff Pr[c1 \equiv c2] \geq p$ , where $Pr[c1 \equiv c2] = \frac{1}{k} \sum_{j=1}^k \{R_j \models c1 \equiv c2\}$ is the ratio of runs that satisfies the coincidence relation out of $k$ runs. The satisfaction of coincidence relation $c1 \equiv c2$ is established in $R_j$ when the two conditions “if $c1$ ticks, $c2$ must tick” and “if $c2$ ticks, $c1$ must tick” always hold.
Probabilistic Exclusion	$c1 \#_p c2$	$c1 \#_p c2 \iff Pr[c1 \# c2] \geq p$ , where $Pr[c1 \# c2] = \frac{1}{k} \sum_{j=1}^k \{R_j \models c1 \# c2\}$ , indicating the ratio of runs that satisfies the exclusion relation out of $k$ runs. A run $R_j$ satisfies the exclusion relation $c1 \# c2$ if the condition “ $c1$ and $c2$ must not tick at the same time” holds.
Probabilistic Precedence	$c1 \prec_p c2$	$c1 \prec_p c2 \iff Pr[c1 \prec c2] \geq p$ , where $Pr[c1 \prec c2] = \frac{1}{k} \sum_{j=1}^k \{R_j \models c1 \prec c2\}$ , which denotes the ratio of runs that satisfies the precedence relation out of $k$ runs. A run satisfies the precedence relation $c1 \prec c2$ if the history of $c1$ is not less than the history of $c2$ , and $c2$ must not tick when their histories are equal.
Probabilistic Causality	$c1 \preceq_p c2$	$c1 \preceq_p c2 \iff Pr[c1 \preceq c2] \geq p$ , where $Pr[c1 \preceq c2] = \frac{1}{k} \sum_{j=1}^k \{R_j \models c1 \preceq c2\}$ , i.e., the ratio of runs satisfying the causality relation among the total number of $k$ runs. A run $R_j$ satisfies the causality relation $c1 \preceq c2$ if the history of $c2$ is always less than or equal to the history of $c1$ .

types of behaviors mentioned in Sec. I), we present an extension of PrCCSL, called PrCCSL\*, which is augmented with notations of interval-based DelayFor, Clock Action, Probabilistic Clock Action and DenseClockType. Furthermore, to allow specification of complex requirements involving multiple events, the binary clock relations in PrCCSL are extended into n-ary clock relations.

In automotive systems, time delays between events/activities (e.g., message transmissions, execution of computational components) can be stochastic and fluctuate randomly in the environments under uncertainty. Those stochastic time delays can be described as random variables that uniformly distributed between the values of the *best-case time* and the *worst-case time*. To express the stochastic time delay between events, we define an interval-based DelayFor expression.

**Definition 3.1 (Interval-based DelayFor):** Let  $C$  be a set of clocks.  $res, ref, base \in C$ ,  $lower, upper \in \mathbb{N}^+$  and  $upper \geq lower$ , the interval-based DelayFor generates a new clock  $res$  by delaying  $ref$  for random number of units (instants) of  $base$  clock, which is expressed as:

$$res \triangleq ref ([lower, upper]) \rightsquigarrow base$$

where  $lower, upper$  are two positive integers that represent the minimum and maximum time delay. The expression is interpreted as:  $res \triangleq ref(x) \rightsquigarrow base \wedge x \in [lower, upper]$ , i.e.,  $res$  is a clock generated by delaying  $ref$  for  $x$  units of  $base$  and  $x$  is given by (continuous/discrete) uniform probability distribution over  $[lower, upper]$ .

The standard DelayFor operator (see Table I) can be seen as a special interval-based DelayFor when  $lower = upper$ . Since the  $base$  clock of the DelayFor operator can be either dense or discrete, the delay should conform to continuous or discrete probability distribution according to the clock type of  $base$ .

Automotive systems are event-driven systems that react to external/internal events, e.g., triggering of sensors/actuators or arrivals of input signals. Event occurrences can lead to execution of related actions, i.e., functions/operations on variables. The “event-action” relation can be specified by Clock Action.

**Definition 3.2 (Clock Action):** Let  $V$  be a set of variables and  $C$  be a set of logical clocks. *Assign* denotes a sequence of assignments to  $V$  and *Func* represents a set of mathematical functions on  $V$ . Clock action is a function  $A: C \mapsto Assign \cup Func$ . Let  $c \in C$ ,  $A(c)$  represents the set of assignments and functions that are invoked to execute when  $c$  ticks. The clock action of  $c$  is defined as:

$$c \rightarrow \{\lambda_1, \lambda_2, \dots, \lambda_n\}$$

where  $n \in \mathbb{N}^+$  is the number of assignments/functions related to  $c$ ,  $\lambda_i \in Assign \cup Func$  ( $i \in \{1, 2, \dots, n\}$ ) denotes an assignment or a function that is executed when an occurrence of  $c$  is detected.

In other words, the clock action relates a logical clock  $c$  with a set of operations that are performed to change the system behaviors. Note that the assignment of each variable in  $V$  must appear at most once in the clock action, i.e., two or more assignments to the same variable are not allowed.

An event can be associated with multiple actions under uncertainty, i.e., it is uncertain which action out of a set of actions the system will take when the event occurs. In this case, the actions related to the same event can be assigned with probabilities and interpreted as probabilistic alternatives. We enrich clock actions with probabilities and define probabilistic clock action.

**Definition 3.3 (Probabilistic Clock Action):** Let  $C$  be a set of logical clocks and  $P$  be a set of real numbers in  $[0, 1]$ . The probabilistic clock action is a function  $A_p: C \times Assign \cup Func \mapsto P$ . Let  $c \in C$  and  $\Lambda_i \subseteq Assign \cup Func$

( $i \in \{1, 2, \dots, n\}$ ),  $A_p(c, \Lambda_i)$  represents the probability of  $\Lambda_i$  being executed when  $c$  ticks. The probabilistic clock action is represented as:

$$c \rightarrow_p \{(p_1, \Lambda_1), (p_2, \Lambda_2), \dots, (p_n, \Lambda_n)\}$$

where  $n \in \mathbb{N}^+$  is the number of sets of actions related to  $c$ ,  $\Lambda_i$  is a set of assignments/functions, and  $p_i \in P$  represents the probability of actions in  $\Lambda_i$  being executed when  $c$  occurs, i.e.,  $p_i = Pr(A(c) = \Lambda_i)$  and  $\sum_{i=1}^n p_i = 1$ .

We express the probability distribution of the set of actions related to event  $c$  by a list of tuples in the form of “( $p_i, \Lambda_i$ )”. For instance,  $c \rightarrow_p \{(0.2, \{v = 0\}), (0.3, \{v = 1\}), (0.5, \{v = 2\})\}$  means that when  $c$  ticks,  $v$  is assigned the value 0, 1, 2 with probability 0.2, 0.3 and 0.5, respectively.

In automotive systems, variations of physical quantities (e.g., energy consumption, temperature) usually involve continuous dynamics described by ODE, as well as discrete changes activated by physical phenomena. For example, battery consumption of an automotive system increases continuously with a certain rate when the vehicle runs under certain modes (e.g., braking or turning) while undergoes discrete increments when the physical phenomena (e.g., turning on/off switches in circuits) take place. To allow the specification of the continuous/discrete variations of those physical quantities, we utilize dense clocks to represent those quantities and extend the attributes of `DenseClockType`, from which the dense clocks can be instantiated.

**Definition 3.4 (DenseClockType):** Let  $n$  and  $m$  be two positive integers. A dense clock type  $DT$  can be defined based on the following four attributes:

`DenseClockType`  $DT$  {reference  $ref$ , factor  $r$ , offset  $\{(c_1, v_1), \dots, (c_n, v_n)\}$ , reset  $\{e_1, \dots, e_m\}$ };

where

- reference specifies a referential dense clock, i.e.,  $ref$ .
- factor indicates the increase rate of instances of  $DT$  compared to  $ref$ .
- offset represents a set of tuples  $\{(c_1, v_1), (c_2, v_2), \dots, (c_n, v_n)\}$ , where  $\forall i \in \{1, 2, \dots, n\}$ ,  $c_i \in C$  is a logical clock and  $v_i \in \mathbb{R}$  represents a real number. The ticks of  $c_i$  result in the instantaneous changes of instances of  $DT$  by  $v_i$  time units.
- reset represents a set of logical clocks  $\{e_1, e_2, \dots, e_m\}$  whose ticks reset the instances of  $DT$ .

Let  $c$  be an instance of  $DT$  and  $v_c$  be a real number that represents the time value of  $c$ . The factor of  $c$  (denoted by  $r$ ) corresponds to the increase rate of  $v_c$  compared to the reference clock. For example, if the reference clock of  $c$  is `idealClk`, then  $v_c = \int r dt$ , where  $t \in [0, \infty]$  represents the physical time. The `offset` and `reset` of  $c$  describe discrete changes of  $v_c$  triggered by events, i.e.,  $\forall i \in \{1, 2, \dots, n\}$ ,  $c_i \rightarrow \{v_c = v_c + v_i\}$  and  $\forall j \in \{1, 2, \dots, m\}$ ,  $e_j \rightarrow \{v_c = 0\}$ .

By utilizing the operators and notations in `PrCCSL*`, a system can be specified as a `Probabilistic Clock Based System (PCBS)`, in which the continuous and discrete system

behaviors can be described as the evolutions of a set of dense and logical clocks.

**Definition 3.5 (Probabilistic Clock Based System (PCBS)):** A probabilistic clock based system is a tuple:

$$S = \langle T, C_t, C_n, Exp, A, A_p \rangle$$

where

- $T$  is a set of clock types;
- $C_t$  is a set of dense clocks;
- $C_n$  is a set of logical clocks;
- $Exp$  is a set of clock expressions;
- $A$  is a set of clock actions of clocks in  $C_n$ ;
- $A_p$  is a set of probabilistic clock actions of clocks in  $C_n$ .

Clocks in  $C_t$  and  $C_n$  are instances derived from clock types in  $T$ .  $C_t$  and  $C_n$  are two exclusive sets. Clock actions and probabilistic clock actions (see Definition 3.2 and 3.3) are employed to describe the actions activated by events (i.e., logical clocks). Since clock *relations* in `PrCCSL*` are applied in specifying requirements and not utilized in system behaviors specifications, a `PCBS` does not contain any clock *relations*.

In `PrCCSL`, requirements are specified by clock *relations* (see Table II) between two events. To describe the complex requirements associated with multiple events, we extend the binary *relations* into *n*-ary *relations*, which allow to describe the dependencies among a set of events.

**Definition 3.6 (N-ary Relation):** Let  $\mathcal{M}$  be a `PCBS` and  $c_1, c_2, \dots, c_n$  are  $n$  clocks in  $\mathcal{M}$ . An *n*-ary clock relation among clocks  $c_1, c_2, \dots, c_n$ , denoted as  $\sim(c_1, c_2, \dots, c_n)$ , is satisfied over  $\mathcal{M}$  if the following condition holds:

$$\mathcal{M} \models \sim(c_1, c_2, \dots, c_n) \iff \forall i, j : 1 \leq i < j \leq n \Rightarrow \mathcal{M} \models c_i \sim c_j$$

where  $\sim \in \{\subseteq, \equiv, \prec, \preceq, \#\}$ ,  $n \geq 2$  is the number of clocks in the relation.

Note that the  $n$  clocks in the *n*-ary *relations* are partially ordered, e.g.,  $\subseteq(c_1, c_2, c_3)$  and  $\subseteq(c_2, c_1, c_3)$  are different subclock *relations*. Informally, an *n*-ary relation, including coincidence, subclock, exclusion, precedence and causality *relations*, is satisfied if any order-preserving pair (i.e., the order relation  $i < j$  is maintained) of two clocks in the set of  $n$  clocks satisfy the corresponding (binary) clock relation. In other words, an *n*-ary clock relation can be seen as the conjunction of  $\frac{n(n-1)}{2}$  corresponding binary *relations*. For example, the ternary causality relation among  $c_1, c_2$  and  $c_3$  limits that the three binary relations, i.e.,  $c_1 \preceq c_2$ ,  $c_1 \preceq c_3$  and  $c_2 \preceq c_3$ , must be satisfied at the same time.

The probabilistic *n*-ary relation is satisfied if the probability of the corresponding *n*-ary relation being satisfied is greater than or equal to a given probability threshold  $p$ .

**Definition 3.7 (Probabilistic N-ary Relation):** The probabilistic *n*-ary relation among  $c_1, c_2, \dots, c_n$ , denoted as  $\sim_p(c_1, c_2, \dots, c_n)$ , is satisfied if the following condition holds:

$$\mathcal{M} \models \sim_p(c_1, c_2, \dots, c_n) \iff Pr(\sim(c_1, c_2, \dots, c_n)) \geq p$$



where  $\sim \in \{\subseteq, \equiv, \prec, \preceq, \#\}$ ,  $Pr(\sim(c_1, c_2, \dots, c_n)) = \frac{1}{k} \sum_{j=1}^k \{R_j \models \sim_p(c_1, c_2, \dots, c_n)\}$ , which is the probability of the  $n$ -ary *relation* being satisfied, i.e., the ratio of runs satisfying the  $n$ -ary *relation* among the total number of  $k$  runs.

#### IV. TRANSLATION OF PrCCSL\* INTO UPPAAL-SMC MODELS

In PrCCSL\*, an automotive system can be specified as a PCBS and the requirements of the system are specified as clock *relations*. To enable the formal analysis of system behaviors/requirements specified in PrCCSL\*, in this section, we first present how to translate PrCCSL\* elements, including *DenseClockType*, *expressions*, (probabilistic) clock actions and *relations*, into verifiable UPPAAL models. Then, we introduce our developed tool ProTL for supporting automatic translation and formal verification of PrCCSL\* specifications.

**Clock and DenseClockType** In PrCCSL\*, a clock is either a logical clock or a dense clock. A logical clock  $c$  represents an event, which is represented as a

*synchronization channel*  $c!$  in UPPAAL-SMC. The history of  $c$  (which represents the number of times that  $c$  has ticked currently) is

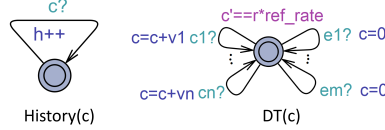


Fig. 1: History and DenseClockType

modeled as the *History(c)* STA (stochastic timed automata) in Fig. 1: whenever  $c$  occurs ( $c?$ ), the value of its history is increased by 1 (i.e.,  $h++$ ).

A dense clock in PrCCSL\* represents the physical time, which is considered as a continuous and unbounded progression of time instants. A dense clock is represented as a “clock” variable in UPPAAL-SMC [18], which is a real type variable increasing monotonically with a certain rate. For instance, the *idealClk* is mapped to a standard *clock* variable whose increase rate is 1 in UPPAAL-SMC.

To describe the continuous and instantaneous variations of physical quantities (e.g., energy consumption, temperature), those quantities are represented as dense clocks instantiated by different *DenseClockTypes*. A *DenseClockTypes*  $DT$  is defined based on the *reference*, *factor*, *offset* and *reset* attributes (see Definition 3.4).  $DT(c)$  STA in Fig. 1 is a generic representation of  $DT$  in UPPAAL-SMC, in which  $c$  is a dense clock instantiated from  $DT$ . The *factor* of  $DT$  (denoted  $r$ ) is the rate of  $c$  compared to the *reference* clock  $ref$ . Let  $ref\_rate$  represent the increase rate of  $ref$  with respect to ideal physical time (i.e., *idealClk*). Thus the increase rate of  $c$  compared to *idealClk* equals to  $r * ref\_rate$ , modeled as the *invariant* “ $c' = r * ref\_rate$ ” in Fig. 1. Thus,  $c$  is changed continuously according to the differential equation “ $c = \int r * ref\_rate dt$ ”, where  $t \in [0, \infty]$  represents the physical time. Moreover, *offset* specifies the instantaneous changes of  $c$  activated by a set of events  $c_1, c_2, \dots, c_n$ . As shown in Fig. 1, each tuple in *offset* corresponds to a self-loop transition where a discrete increment of  $c$  is performed.

*reset* is a set of events whose occurrences reset the time value of  $c$  into 0, modeled as the transitions where  $c$  is *reset*.

**Clock Expressions** generate new clocks based on existed clocks. To model clock *expressions* in UPPAAL-SMC, the resulting clock of the *expression* is represented by a channel variable  $res$  and the semantics of the *expression* is modeled as an STA that determines when  $res$  ticks (via  $res!$ ). For example, the interval-based *DelayFor* *expression* (Definition 3.1), expressed as  $res \triangleq ref \ ([lower, upper]) \rightsquigarrow base$ , is modeled as the STA in Fig. 2. *DelayFor* defines a new clock  $res$  based on a *reference* clock ( $ref$ ) and a *base* clock, i.e.,  $\forall i \in N^+$ , the  $i^{th}$  tick of  $res$  is generated by delaying the  $i^{th}$  tick of  $ref$  for  $[lower, upper]$  units of *base*. The generation process of the  $i^{th}$  tick of  $res$  can be summarized as: when the  $i^{th}$  tick of  $ref$  occurs, after  $[lower, upper]$  time units of *base* clock is elapsed, the  $i^{th}$  tick of  $res$  is triggered. The generation processes of different ticks of  $res$  are independent and can run in parallel.

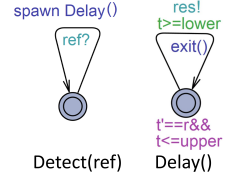


Fig. 2: DelayFor

Therefore, we model the generation of each tick of  $res$  as a spawnable STA [5] (i.e., the *Delay()* STA in Fig. 2), which is dynamically spawned (by *Detect(ref)* STA) whenever  $ref$  occurs ( $ref?$ ), and terminated when the calculation of the current tick of  $res$  is completed (denoted “*exit()*”). Here the *base* in *DelayFor* is a dense clock and  $r$  is the increase rate of *base* compared to *idealClk*. For the STA of *DelayFor* with discrete *base* clock and other clock *expressions*, refer to [19].

#### (Probabilistic) Clock

**Action** (see Definition 3.2 and 3.3) of clock  $c$  are modeled as the STA in Fig. 3. The clock action “ $c \rightarrow \{v[0] = 0, v[1] = 1, v[2] = 2\}$ ” is modeled as the *Action(c, v)* STA: when  $c$  ticks ( $c?$ ), the assignment operations on the integer array  $v$  are executed. The probabilistic clock action of clock  $c$  specifies a set of actions that are performed when  $c$  ticks based on a discrete probability distribution. For instance, the probabilistic clock action “ $c \rightarrow_p \{(p[0], v[0] = 0), (p[1], v[1] = 1), (p[3], v[2] = 2)\}$ ” is modeled as the *pAction(c, p, v)* STA, in which each element of the action is mapped to a probabilistic transition weighted by corresponding probability.

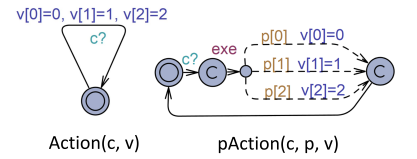
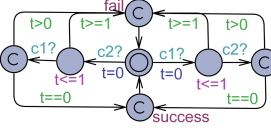
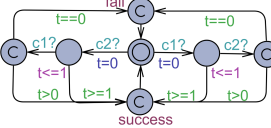
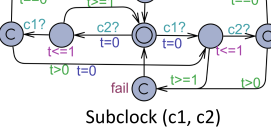
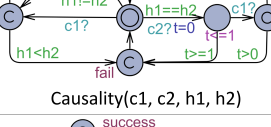
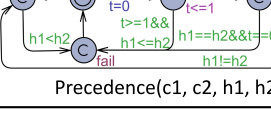


Fig. 3: STA of clock action

Based on the mapping patterns described above, a PCBS (that consists of a set of clocks, clock types, *expressions* and clock actions) can be represented as a network of STA (NSTA), which consists of the STA of corresponding clock type, (probabilistic) clock actions and *expressions*.

**Probabilistic Clock Relations** To represent PrCCSL\* *relations* in UPPAAL-SMC, observer STA that capture the semantics of standard subclock, coincidence, exclusion,

TABLE III: STA of PrCCSL\* Relations

STA	Remarks
 <p>Coincidence(c1, c2)</p>	<p>Coincidence relation <math>c1 \equiv_p c2</math> delimits that two clocks must tick simultaneously. When <math>c1</math> (<math>c2</math>) ticks via <math>c1?</math> (<math>c2?</math>), the STA judges if the other clock, <math>c2</math> (<math>c1</math>) ticks at the same time. If there is no time elapsed between the corresponding occurrences of <math>c1</math> and <math>c2</math> (i.e., "<math>t=0</math>"), the STA transits to <i>success</i> location. Otherwise, it goes to <i>fail</i> location.</p>
 <p>Exclusion (c1, c2)</p>	<p>Exclusion relation <math>c1 \#_p c2</math> limits that two clocks must not occur at the same time. Contrary to the <i>Coincidence</i>(c1, c2) STA, when <math>c1</math> (<math>c2</math>) ticks and if the other clock ticks simultaneously ("<math>t=0</math>"), the STA goes to the <i>fail</i> location.</p>
 <p>Subclock (c1, c2)</p>	<p>Subclock relation <math>c1 \sqsubseteq_p c2</math> states that <math>c2</math> (superclock) must tick when <math>c1</math> (subclock) ticks, which is interpreted as a conditional coincidence relation: when <math>c1</math> ticks, <math>c2</math> must coincide with <math>c1</math>. Similar to <i>Coincidence</i>(c1, c2) STA, when <math>c1</math> (<math>c2</math>) occurs, the <i>Subclock</i>(c1, c2) STA checks whether the other clock also ticks at the same instant. If <math>c1</math> ticks and <math>c2</math> does not occur (denoted "<math>t&gt;0</math>"), the relation is violated and the STA transits to <i>fail</i> location.</p>
 <p>Causality(c1, c2, h1, h2)</p>	<p>Causality relation <math>c1 \preceq_p c2</math> states that <math>c2</math> (effect) must not tick prior to <math>c1</math> (cause). When <math>c1</math> or <math>c2</math> ticks, if the two clocks are coincident (represented by "<math>t=0</math>") or <math>c1</math> ticks faster (denoted "<math>h1 \geq h2</math>"), the relation is satisfied and the STA goes to <i>success</i> location. Otherwise, the STA goes to <i>fail</i> location.</p>
 <p>Precedence(c1, c2, h1, h2)</p>	<p>Precedence relation <math>c1 \prec_p c2</math> states that <math>c1</math> must run faster than <math>c2</math>. When <math>c1</math> or <math>c2</math> ticks, if <math>c2</math> ticks faster ("<math>h1 &lt; h2</math>") or <math>c1</math> and <math>c2</math> are coincident (represented by "<math>h1=h2 \&amp; t=0</math>") the relation is violated and <i>fail</i> location is activated.</p>

precedence and causality relations are constructed. In our earlier work [15], we have shown the translation patterns from clock relations into STA. However, the patterns are given based on discrete time semantics, i.e., the continuous physical time line is discretized into a set of equalized steps. As a result, (in the extreme cases) two clock instants are still considered coincident even if they are one time step apart. In this paper, we refine the STA of relations in [15] to support the translation of relations conformed to continuous time semantics. The refined STA are illustrated in Table III, in which  $t$  represents the time delay between two consecutive instants of clock  $c1$  and  $c2$ .  $c1$  and  $c2$  are simultaneous if  $t$  equals to 0.  $h1$  and  $h2$  represent the histories of  $c1$  and  $c2$ , respectively. Each relation is mapped to an observer STA that contains a "fail" location, which suggests the violation of corresponding relation. Recall the definition of probabilistic relations in Sec. II, the probability of a relation being satisfied is interpreted as a ratio of runs that satisfies the relation among all runs. It is specified as Hypothesis Testing query in UPPAAL-SMC,  $H_0: \frac{m}{k} \geq p$  against  $H_1: \frac{m}{k} < p$ , where  $m$  is the number of runs satisfying the given relation out of all  $k$  runs. As a result, the probabilistic relations are interpreted as the query:  $Pr[bound]([\neg STA_{obs}.fail]) \geq p$ , which means that the probability of the "fail" location of the observer STA

(denoted  $STA_{obs}$ ) never being reached should be greater than or equal to threshold  $p$ .

Requirements associated with multiple clocks can be expressed by n-ary relations (see Definition 3.6). According to the definition, an n-ary relation among  $n$  clocks  $c_1, c_2, \dots, c_n$  is satisfied if any pair of two clocks  $\langle c_i, c_j \rangle$  ( $1 \leq i < j \leq n$ ) satisfies the corresponding binary relation. Since an n-ary clock relation can be seen as the conjunction of  $\frac{n(n-1)}{2}$  binary relations, we construct an STA for the relation of each pair  $\langle c_i, c_j \rangle$  ( $1 \leq i < j \leq n$ ) and an n-ary relation is represented as the synchronization product of the  $\frac{n(n-1)}{2}$  STA. For instance, an n-ary exclusion can be represented as the composition of the  $\frac{n(n-1)}{2}$  Exclusion( $i, j$ ) STA in Fig. 4(a) (similar to the Exclusion( $c1, c2$ ) STA in Table III), which represents the exclusion relation between  $c_i$  and  $c_j$ . The probabilistic n-ary exclusion is specified as:  $Pr[\leq bound]([\ ] forall(i : int[1,n]) forall(j : int[1,n]) (i < j \text{ imply not } Exclusion(i,j).fail)) \geq p$ .

The coincidence, subclock, precedence and causality relations are transitive relations, i.e., if both the relations between  $c_i$  and  $c_j$  and the relation between clock  $c_j$  and  $c_k$  are satisfied, then the relation between  $c_i$  and  $c_k$  is also satisfied. Thus, an n-ary transitive relation can be interpreted as the combination of  $n - 1$  binary

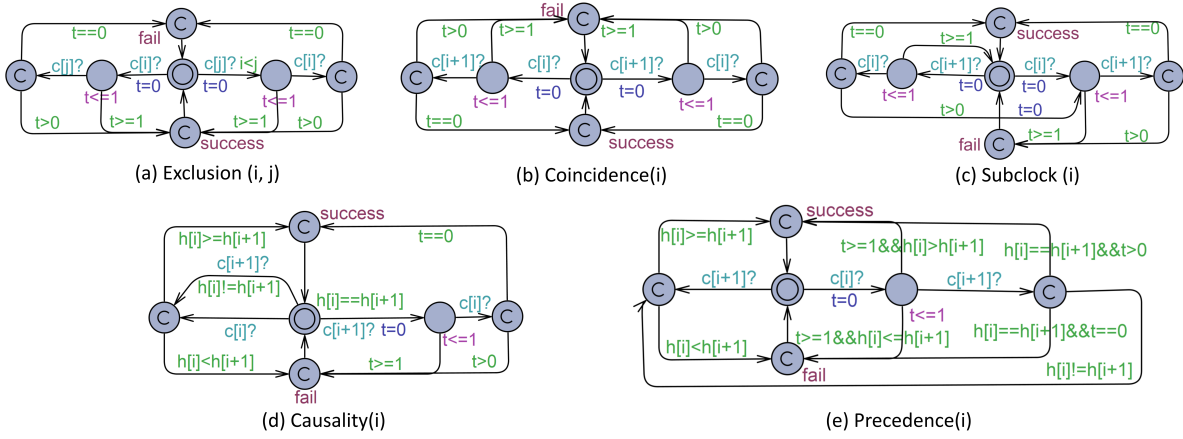


Fig. 4: STA of N-ary Relations

*relations*. For instance, the  $n$ -ary coincidence relation is interpreted as:  $\equiv(c_1, c_2, \dots, c_n) \iff \bigwedge_{i=1}^{n-1} c_i \equiv c_{i+1}$ . As illustrated in Fig. 4(b), Coincidence(i) STA represents the coincidence relation between  $c_i$  and  $c_{i+1}$  (similar to the Coincidence( $c_1, c_2$ ) STA in Table III). The  $n$ -ary coincidence relation can be represented as the composition of the  $n - 1$  Coincidence(i) STA, where  $i \in \{1, 2, 3, \dots, n - 1\}$ . The  $n$ -ary probabilistic coincidence relation can be verified by using the query:  $Pr[\leq \text{bound}]([\ ] \text{forall}(i : \text{int}[1, n - 1])(\text{not Coincidence}(i).\text{fail})) \geq p$ . Similarly, the other three transitive relations, i.e., subclock, precedence and causality relations, can be represented based on the STA in Fig. 4.

**Tool Support** To improve the efficiency and accuracy of translation, we implement a tool ProTL (Probabilistic CCSL Translator) [19], which provides a push-button transformation from PrCCSL\* specifications into UPPAAL-SMC models. Moreover, to enable the formal verification of PrCCSL\* specifications, ProTL brings the capability of verification for the translated UPPAAL-SMC models by employing UPPAAL-SMC as its verification backend. Furthermore, ProTL offers a configuration panel for customizable generation of five types of probabilistic queries (introduced in Sec. II). ProTL can also generate counter-examples that depict the evolution of clocks related to the unsatisfied clock relations, which provide diagnosis information for further refinement of PrCCSL\* specifications.

## V. CASE STUDIES

We show the applicability of ProTL on two automotive systems case studies. We report only the verification on a list of representative requirements for each example and further details can be found in [19].

**Autonomous Vehicle (AV)** [15] reads the road signs, e.g., “speed limits”, “stop” or “right/left turn”, and adjusts its speed and movements accordingly. To achieve traffic sign recognition, a camera is equipped in the vehicle to capture images.

The camera relays the captured images to a sign recognition device periodically. The representative requirements on AV are listed below:

- A1. A periodic acquisition of camera must be carried out every 50ms with a jitter of 10ms.
- A2. If the vehicle detects a left turn sign, it should start to turn left within 300ms.
- A3. The detected image should be computed within [20, 30]ms in order to generate the desired sign type.
- A4. When a traffic sign is recognized, the speed of vehicle should be updated within [50, 150]ms based on the sign type.
- A5. The required environmental information should arrive to the controller within 40 ms, i.e., the input signals (traffic sign type, speed, direction, gear and torque) must be detected by controller within 40 ms.
- A6. The execution time interval from the controller to the actuator must be less than or equal to the sum of the worst case execution time intervals of controller and actuator.
- A7. While the vehicle turns left, the “turning right” mode should not be activated. The events of turning left and right are considered as exclusive and specified as an exclusion constraint.

We specify the system behaviors of AV as a PCBS and requirements as clock relations in PrCCSL\*. The PrCCSL\* specifications of requirements (A1–A7) and the verification results are illustrated in Table IV. Let *camera* represent the triggering event of camera and *camera*[ $i$ ] denote the  $i^{th}$  occurrence of *camera*. A1 can be interpreted as:  $\forall i \in \mathbb{N}^+$ , *camera*[ $i + 1$ ] should occur later than *camera*[ $i$ ] delaying for 40ms but prior to *camera*[ $i$ ] delaying for 60ms, which is specified as a ternary precedence relation in Table IV. The “forall” query in UPPAAL is employed to verify the ternary relation.

In the specification of A2, *detectLeftSign* represents the event that a left turn sign is detected. *startTurnLeft* denotes the event that the vehicle starts to turn left. We construct a new clock *leftSignDe* by delaying *detectLeftSign* for 300ms. A2 can be expressed as a precedence relation between *startTurnLeft* and *leftSignDe*, i.e., *startTurnLeft*

TABLE IV: Verification Results of AV

Req	Spec	Expression	Runs	Result	Time (Min)	Mem (Mb)	CPU (%)
A1	PrCCSL*	$cameraFltr \triangleq camera \blacktriangledown 01(1)$ $cameraDelay40 \triangleq camera(40) \rightsquigarrow ms$ $cameraDelay60 \triangleq camera(60) \rightsquigarrow ms$ $\prec_{0.96}(cameraDelay40, cameraFltr, cameraDelay60)$	142	valid	9.69	8.49	25.08
	UPPAAL	$Pr[\leq 10000]([ ] \text{ forall } i:[1, 2] \neg A1\_Precedence(i).fail) \geq 0.96$					
A2	PrCCSL*	$leftSignDe \triangleq detectLeftSign(300) \rightsquigarrow ms$ $startTurnLeft \prec_{0.95} leftSignDe$	161	valid	8.72	8.30	25.40
	UPPAAL	$Pr[\leq 10000]([ ] \neg A2\_Precedence.fail) \geq 0.95$					
A3	PrCCSL*	$ImgRecDe20 \triangleq ImgRec(20) \rightsquigarrow ms$ $ImgRecDe30 \triangleq ImgRec(30) \rightsquigarrow ms$ $\prec_{0.96}(ImgRecDe20, signType, ImgRecDe30)$	142	valid	10.31	8.51	25.27
	UPPAAL	$Pr[\leq 10000]([ ] \text{ forall } i:[1, 2] \neg A3\_Causality(i).fail) \geq 0.96$					
A4	PrCCSL*	$signTypeDe50 \triangleq signType(50) \rightsquigarrow ms$ $signTypeDe150 \triangleq signType(150) \rightsquigarrow ms$ $\prec_{0.95}(signTypeDe50, updateSpeed, signTypeDe150)$	140	valid	10.25	8.58	25.13
	UPPAAL	$Pr[\leq 10000]([ ] \text{ forall } i:[1, 2] \neg A4\_Precedence(i).fail) \geq 0.95$					
A5	PrCCSL*	$InfIn \triangleq speedIn \wedge posIn \wedge dirIn \wedge signType$ $SupIn \triangleq speedIn \vee posIn \vee dirIn \vee signType$ $InfInDe40 \triangleq InfIn(40) \rightsquigarrow ms$ $SupIn \prec_{0.95} InfInDe40$	140	valid	12.96	8.6	24.86
	UPPAAL	$Pr[\leq 10000]([ ] \neg A5\_Causality.fail) \geq 0.95$					
A6	PrCCSL*	$signTypeDe \triangleq signType(SUM_{WCET}) \rightsquigarrow ms$ $actOut \prec_{0.95} signTypeDe$	140	valid	12.11	8.48	24.24
	UPPAAL	$Pr[\leq 10000]([ ] \neg A6\_Precedence.fail) \geq 0.95$					
A7	PrCCSL*	$turnLeft \triangleq \{inLeft = 1\} ? \text{always} : \text{never}$ $turnRight \triangleq \{inRight\} ? \text{always} : \text{never}$ $turnLeft \#_{0.95} turnRight$	140	valid	8.82	8.40	25.31
	UPPAAL	$Pr[\leq 10000]([ ] \neg A7\_Exclusion.fail) \geq 0.95$					

should occur no later than the occurrence of *leftSignDe*. Similarly, A3–A6 can be specified. In the PrCCSL\* specification of A7, *always* (*never*) represents a clock that always (never) ticks. Based on *always* and *never*, we generate two new clocks *turnLeft* and *turnRight* by using ITE expressions. *turnLeft* (*turnRight*) represents the event that the vehicle is turning left (right). A7 is specified as an exclusion relation between *turnLeft* and *turnRight*.

**Cooperative Automotive System (CAS)** [20] includes distributed and coordinated sensing, control, and actuation over three vehicles which are running in the same lane. A lead vehicle can run automatically by recognizing traffic signs on the road. The follow vehicle must set its desired velocity identical to that of its immediate preceding vehicle. Vehicles should maintain sufficient braking distance to avoid rear-end collision while remaining close enough to guarantee communication quality. The position of each vehicle is represented by Cartesian coordinate  $(x_i, y_i)$ , where  $x_i$  and  $y_i$  ( $i \in \{0, 1, 2\}$ ) are distances measured from the vehicle to the two fixed perpendicular lines, i.e., x-axis and y-axis, respectively. A list of the representative requirements on CAS are given below:

B1. The follow vehicle should not overtake the lead vehicle.

B2. When the lead vehicle adjusts its movement (e.g., brak-

ing) regarding environmental information, the follow vehicle should move towards the lead one within 500ms.

B3. Each vehicle should maintain braking distance, i.e., if the braking distance is insufficient, the follow vehicle should decelerate within a given time.

B4. When the lead vehicle starts to turn left, both the lead and follow vehicle should complete turning and run in the same lane within a certain time.

B5. The velocity of vehicles should update every 30ms, i.e., a periodic acquisition of a speed sensor must be carried out for every 30ms.

B6. The required input signals of the environmental information (speed, position, direction) must be detected by controller within a given time window, i.e., 60ms.

B7. The controller of the follow vehicle should finish its execution within [30, 100]ms.

The specifications and verification results of B1–B7 are shown in Table V. In the PrCCSL\* specification of B1, *runAtXDir* indicates that the vehicles are running at the positive x-direction (“*direction=xDir*”). *overTake* represents the event that the position of follow on x-axis is greater than that of lead vehicle ( $x_1 \geq x_0$ ). B1 limits that *runAtXDir* and *overTake* can not happen at the same time, which can



TABLE V: Verification Results of CAS

Req	Spec	Expression	Runs	Result	Time (Min)	Mem (Mb)	CPU (%)
B1	PrCCSL*	$runAtXDir \triangleq \{direction = xDir\} ? always : never$ $overTake \triangleq \{x_1 \geq x_0\} ? always : never$ $runAtXDir \#_{0.95} overTake$	140	valid	128.34	10.62	25.34
	UPPAAL	$Pr[\leq 10000]([ ] \neg B1\_Exclusion.fail) \geq 0.95$					
B2	PrCCSL*	$brakeDelay500 \triangleq leadBrake(500) \rightsquigarrow ms$ $followBrake \prec_{0.95} brakeDelay500$	140	valid	132.83	8.15	25.17
	UPPAAL	$Pr[\leq 10000]([ ] \neg B2\_Precedence.fail) \geq 0.95$					
B3	PrCCSL*	$notSafe \triangleq \{inConst = true \& \& dist < safeDis\} ? always : never$ $notSafeDe300 \triangleq notSafe(300) \rightsquigarrow ms$ $const2dec \prec_{0.95} notSafeDe300$	140	valid	126.40	10.52	25.29
	UPPAAL	$Pr[\leq 10000]([ ] \neg B3\_Precedence.fail) \geq 0.95$					
B4	PrCCSL*	$leadTurnLeftDe \triangleq leadTurnLeft(500) \rightsquigarrow ms$ $followTurn \prec_{0.95} leadTurnLeftDe$	54	Unsatisfied	60.87	10.66	24.86
	UPPAAL	$Pr[\leq 10000]([ ] \neg B4\_Precedence.fail) \geq 0.95$					
B5	PrCCSL*	$prdClk \triangleq ms \propto 30$ $\equiv_{0.98} (leadSpeedTrig, prdClk, followSpeedTrig)$	145	valid	160.91	10.69	25.05
	UPPAAL	$Pr[\leq 10000]([ ] \text{forall } (i:int[1,2]) \neg B5\_Coincidence(i).fail) \geq 0.98$					
B6	PrCCSL*	$InfIn \triangleq speedIn \wedge posIn \wedge dirIn$ $SupIn \triangleq speedIn \vee posIn \vee dirIn$ $InfInDe \triangleq InfIn(60) \rightsquigarrow ms$ $SupIn \prec_{0.95} InfInDe$	140	valid	188.63	8.92	24.09
	UPPAAL	$Pr[\leq 10000]([ ] \neg B6\_Precedence.fail) \geq 0.95$					
B7	PrCCSL*	$ctrlInDe30 \triangleq ctrlIn(30) \rightsquigarrow ms$ $ctrlInDe100 \triangleq ctrlIn(100) \rightsquigarrow ms$ $\prec_{0.95} (ctrlInDe30, ctrlOut, ctrlInDe100)$	140	valid	8.82	8.4	25.31
	UPPAAL	$Pr[\leq 10000]([ ] \text{forall } (i:int[1,2]) \neg B7\_Causality(i).fail) \geq 0.95$					

be expressed by *exclusion relation*. In the specification of B2, *leadBrake* (*followBrake*) denotes the event that the lead (follow) vehicle starts to brake. *brakeDelay500* is built by delaying *leadBrake* for 500ms. Thus B2 can be specified as a *precedence relation* between *followBrake* and *brakeDelay500*. Similarly, B3–B4 and B6–B7 can be specified. To specify B5, a periodic clock *prdClk* that ticks every 30 ms is generated by using *PeriodicOn expression*. *leadSpeedTrig* and *followSpeedTrig* represent the triggering events of speed sensors of the lead and follow vehicles. B5 is specified as a ternary *coincidence relation*.

The verification results in Table V shows that B1–B3 and B5–B7 are established as valid while B4 is unsatisfied. The invalid property B4 is identified using ProTL which generates a counter-example (CE) presented in Fig. 5. After analyzing CE, the cause of error was found: when the follow vehicle is decreasing its speed and the lead vehicle turns left, the follower keeps speeding down but does not turn left until the deceleration is completed. Based on the CE, the system model are refined: when the follower is under deceleration mode and it detects that the lead vehicle turns left, the follower first turns left and then continues to speed down after turning. After the modification, B4 becomes valid.

## VI. RELATED WORK

In the context of EAST-ADL, efforts on the integration of EAST-ADL and formal techniques were investigated in

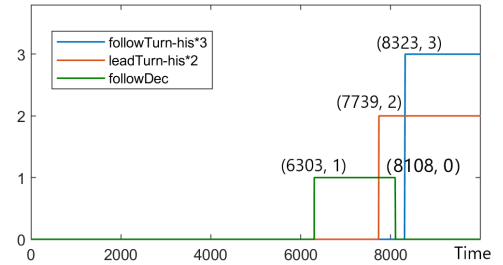


Fig. 5: CE of B4: *leadTurn-his* (*followTurn-his*) represents the history of the clock (event) that the lead (follow) vehicle turns left. At Time=7739, the lead vehicle starts to turn left (*leadTurn-his* becomes 1), while the follow vehicle is under deceleration mode (represented by “*followDec*==1”). The follow does not turn until it finishes the deceleration at Time=8108 and starts to turn left at Time=8323 (*followTurn-his* becomes 1), which violates B4.

several works [21]–[25], which are however, limited to the executional aspects of system functions without addressing dynamic and stochastic behaviors. Kang [26] defined the execution semantics of both the controller and the environment of industrial systems in CCSL which are also given as mapping to UPPAAL models amenable to model checking. Du et al. [27] proposed a probabilistic extension of CCSL, called pCCSL, for specifying the stochastic behaviors of uncertain events in cyber-physical (CPS) and provided the transformation from pCCSL into Stochastic Hybrid Automata. In contrast to our

current work, those approaches lack precise annotations specifying continuous dynamic behaviors in particular regarding different clock rates during execution.

Transformation of CCSL specifications into verifiable formalisms for formal analysis has been investigated in several works [28], [29]. Yin et al. [28] translated CCSL specifications into Promela models amenable to model checking using SPIN model checker. Chen et al. [29] performed formal analysis of timed behaviors specified in CCSL by mapping the specifications into timed Input/Output automata. However, neither tool support for automatic transformation nor probabilistic analysis is provided in those works. Zhang et al. [30] implemented a tool *clzyzer* for formal analysis of CCSL constraints through automated translation from CCSL specifications into SMT formulas amenable to SMT solving. Compared to their approach, we provide the tool support for probabilistic analysis of dynamic and stochastic systems behaviors based on the translation from PrCCSL\* specifications into formal models.

## VII. CONCLUSION

In this paper, we present a tool-supported approach for formal verification of dynamic and stochastic behaviors for automotive systems. To enable the formal specifications of stochastic behaviors and continuous dynamics in automotive systems, we propose an extension of PrCCSL, i.e., PrCCSL\*, which is augmented with notations for descriptions of continuous/discrete variations of physical quantities, stochastic time delays, activations of actions and nondeterministic alternatives. Moreover, to support the specification of complex requirements involved with multiple events, we extend the binary *relations* into *n-ary relations* in PrCCSL\*. To enable the formal verification of system behaviors/requirements specified in PrCCSL\*, we provide the mapping rules to translate PrCCSL\* specifications into verifiable UPPAAL-SMC models. Based on the proposed translation strategies, we implement an automatic translation tool, namely ProTL, which also supports verification of the translated models by leveraging UPPAAL-SMC as an analysis backend. The applicability of our approach and tool is demonstrated by conducting verification of (non)-functional properties on two automotive system case studies.

As ongoing work, formal validation of the correctness of translation rules from PrCCSL\* into stochastic UPPAAL-SMC models is further investigated. Furthermore, new features of ProTL with respect to analysis of UPPAAL-SMC models involving wider range of variable/query types (e.g., *urgent channels* and *bounded integers*) are further developed.

**Acknowledgment.** This work is supported by the EASY project funded by NSFC, a collaborative research between Sun Yat-Sen University and University of Southern Denmark.

## REFERENCES

- [1] "ISO 26262-6: Road vehicles functional safety part 6. Product development at the software level."
- [2] "IEC 61508: Functional safety of electrical electronic programmable electronic safety related systems."
- [3] A. Legay and M. Viswanathan, "Statistical model checking: challenges and perspectives," *STTT*, vol. 17, no. 4, pp. 369 – 376, 2015.
- [4] A. David, D. Du, K. G. Larsen, A. Legay, M. Mikučionis, D. B. Poulsen, and S. Sedwards, "Statistical model checking for stochastic hybrid systems," in *HSB. EPTCS*, 2012, pp. 122 – 136.
- [5] A. David, K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen, "UPPAAL-SMC tutorial," *STTT*, vol. 17, no. 4, pp. 397 – 415, 2015.
- [6] "EAST-ADL specification v2.1.9," MAENAD, Tech. Rep., 2011. [Online]. Available: [https://www.maenad.eu/public/EAST-ADL-Specification\\_M2.1.9.1.pdf](https://www.maenad.eu/public/EAST-ADL-Specification_M2.1.9.1.pdf)
- [7] "MAENAD," <http://www.maenad.eu/>, pp. 2 – 143, 2011.
- [8] H. Blom, L. Feng, H. Lönn, J. Nordlander, S. Kuntz, B. Lisper, S. Quinton, M. Hanke, M.-A. Peraldi-Frati, A. Goknil, J. Deantoni, G. B. Defo, K. Klobedanz, M. Özhan, and O. Honcharova, "TIMMO-2-USE Timing Model, Tools, Algorithms, Languages, Methodology, Use Cases," *TIMMO-2-USE*, Tech. Rep., 2012.
- [9] Object Management Group, "UML profile for MARTE: Modeling and analysis of real-time embedded systems," Tech. Rep., 2011.
- [10] C. André, "Syntax and semantics of the clock constraint specification language (CCSL)," Ph.D. dissertation, INRIA, 2009.
- [11] F. Mallet and R. De Simone, "Correctness issues on MARTE/CCSL constraints," *Science of Computer Programming*, vol. 106, pp. 78 – 92, 2015.
- [12] A. M. Khan, F. Mallet, and M. Rashid, "Combining SysML and MARTE/CCSL to model complex electronic systems," in *ICISE. IEEE*, 2016, pp. 12–17.
- [13] M.-A. Peraldi-Frati, A. Goknil, J. DeAntoni, and J. Nordlander, "A timing model for specifying multi clock automotive systems: The timing augmented description language v2," in *ICECCS. IEEE*, 2012, pp. 230–239.
- [14] J. Liu, Z. Liu, J. He, F. Mallet, and Z. Ding, "Hybrid MARTE statecharts," *Frontiers of Computer Science*, vol. 7, no. 1, pp. 95–108, 2013.
- [15] E. Y. Kang, D. Mu, and L. Huang, "Probabilistic verification of timing constraints in automotive systems using UPPAAL-SMC," in *iFM. Springer*, 2018, pp. 236–254.
- [16] L. Huang and E. Y. Kang, "Formal verification of safety & security related timing constraints for a cooperative automotive system," in *International Conference on Fundamental Approaches to Software Engineering (FASE). Springer*, 2019, pp. 210–227.
- [17] G. Bernat, A. Burns, and A. Llamosi, "Weakly hard real-time systems," *Transactions on Computers*, vol. 50, no. 4, pp. 308 – 321, 2001.
- [18] "UPPAAL-SMC," <http://people.cs.aau.dk/~adavid/smc/>.
- [19] "ProTL," <https://sites.google.com/view/protl>.
- [20] E. Y. Kang, L. Huang, and D. Mu, "Formal verification of energy and timed requirements for a cooperative automotive system," in *SAC. ACM*, 2018, pp. 1492 – 1499.
- [21] A. Goknil, J. DeAntoni, M.-A. Peraldi-Frati, and F. Mallet, "Tool support for the analysis of TADL2 timing constraints using TimeSquare," in *ICECCS*, 2013.
- [22] E. Y. Kang, P. Y. Schobbens, and P. Pettersson, "Verifying functional behaviors of automotive products in EAST-ADL2 using UPPAAL-PORT," in *SAFECOMP. Springer*, 2011, pp. 243 – 256.
- [23] F. Mallet, M.-A. Peraldi-Frati, and C. André, "MARTE/CCSL to execute EAST-ADL timing requirements," in *International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC). IEEE*, 2009, pp. 249 – 253.
- [24] L. Feng, D. Chen, H. Lönn, and M. Törngren, "Verifying system behaviors in EAST-ADL2 with the SPIN model checker," in *ICMA. IEEE*, 2010, pp. 144–149.
- [25] E. Y. Kang, G. Perrouin, and P. Y. Schobbens, "Model-based verification of energy-aware real-time automotive system," in *International Conference on Engineering of Complex Computer Systems (ICECCS). IEEE Computer Society*, 2013, pp. 135 – 144.
- [26] E. Y. Kang and P. Y. Schobbens, "Schedulability analysis support for automotive systems: from requirement to implementation," in *SAC. ACM*, 2014, pp. 1080 – 1085.
- [27] D. Du, P. Huang, K. Jiang, and F. Mallet, "pCCSL: A stochastic extension to MARTE/CCSL for modeling uncertainty in Cyber Physical Systems," *Science of Computer Programming*, vol. 166, pp. 71–88, 2018.
- [28] L. Yin, F. Mallet, and J. Liu, "Verification of MARTE/CCSL time requirements in PROMELA/SPIN," in *ICECCS. IEEE*, 2011, pp. 65 – 74.
- [29] B. Chen, X. Li, and X. Zhou, "Model checking of MARTE/CCSL time behaviors using timed I/O automata," *Journal of Systems Architecture*, vol. 88, pp. 120–125, 2018.

- [30] M. Zhang, F. Mallet, and H. Zhu, “An SMT-based approach to the formal analysis of MARTE/CCSL,” in *FormaliSE*, 2016, pp. 433–449.