A Logical Approach for the Schedulability Analysis of CCSL

Yuanrui Zhang, Frédéric Mallet, Huibiao Zhu, Yixiang Chen

A Logical Approach for the Schedulability Analysis of CCSL

Yuanrui Zhang*§, Frédéric Mallet†, Huibiao Zhu‡, Yixiang Chen*¶
*MoE Engineering Research Center for Software/Hardware Co-design, East China Normal University
†I3S Laboratory, UMR 7271 CNRS, INRIA, Université Nice Sophia Antipolis
‡Shanghai Key Laboratory of Trustworthy Computing, East China Normal University
Email: §zhangyrmath@126.com ¶yxchen@sei.ecnu.edu.cn (corresponding author)

***Abstract*—The Clock Constraint Specification Language (CCSL) is a clock-based formalism for formal specification and analysis of real-time embedded systems. Previous approaches for the schedulability analysis of CCSL specifications are mainly based on model checking or SMT-checking. In this paper we propose a logical approach mainly based on theorem proving. We build a dynamic logic called 'clock-based dynamic logic' (cDL) to capture the CCSL specifications and build a proof calculus to analyze the schedule problem of the specifications. Comparing with previous approaches, our method benefits from the dynamic logic that provides a natural way of capturing the dynamic behaviour of CCSL and a divide-and-conquer way for 'decomposing' a complex formula into simple ones for an SMT-checking procedure. Based on cDL, we outline a method for the schedulability analysis of CCSL. We illustrate our theory through one example.**

***Index Terms*—CCSL, dynamic logic, schedulability analysis**

I. Introduction

The clock constraint specification language (CCSL) [1] is a formal specification language for specifying the behaviour of real-time embedded systems (RETSs) at high-level. It was firstly defined as an annex of UML/MARTE [2], but later developed as an independent language equipped with a formal semantics [3]. CCSL gives a concrete syntax to deal with logical clocks, made popular by Leslie Lamport [4] and synchronous languages (such as Esterel [5]). CCSL handles 'clocks' as first-class citizens for capturing discrete-time events and logical/chronometrical constraints between events. CCSL is a specification language and not a programming language, it describes a set of possible behaviours and does not attempt to provide a single operational deterministic execution. All the values are ignored to focus only on clock issues. CCSL has been widely used in the specification and analysis of different RTESs, e.g. [6]–[8].

One important aspect in the formal analysis of CCSL is the schedulability analysis of CCSL specifications, whose major goal is to answer whether 'there exists a schedule for a given CCSL specification'. Though the decidability of this problem still remains open [9], previous approaches have been proposed to give a partial solution of this problem [9]–[12]. They either rely on an incomplete transformation into finite automata or an encoding into expensive logic formulae. In this paper, we propose a logical approach for the formal analysis of the schedule problem in CCSL. We propose a variation of dynamic logic called 'clock-based dynamic logic' (cDL) and a modular proof calculus of it. cDL enriches first-order dynamic logic (FODL) [13] with clocks as primitives and inherits a fragment of dynamic formulae from differential temporal logic^2 (dTL^2) [14]. cDL provides a natural way for modelling both the dynamic clock behaviour and the static clock relations of a CCSL specification in a single formalism, and it provides a divide-and-conquer way for analyzing the schedule problem in the form of theorem proving.

In cDL, a CCSL specification SP can be captured as a cDL dynamic formula φ_{SP}. The schedule problem of the specification SP is then reduced to the problem of proving the formula φ_{SP}. In the proof calculus of cDL, formula φ_{SP} is modularly transformed into a set of qunntifier-free linear integer arithmetic (QF-LIA) logic formulae, which prove to be decidable and can be efficiently handled via an SMT-checking procedure [15]. On the other hand, the derivation paths during the proof of φ_{SP} can be analyzed to generate a possible bounded schedule of SP.

The contributions of this paper are mainly twofold:

(i) We construct the cDL and its proof calculus.
(ii) We outline a method for the schedulability analysis of CCSL specifications based on cDL. We focus on the encoding from CCSL specifications into cDL formulae, and how the schedule problem can be solved through the derivation of cDL.

Related Work Previous approaches for solving the schedule problem are mainly based on model checking and SMT-checking techniques. The former technique [10], [11] proposes encoding CCSL specification into a finite transition system where the reachability analysis is made. When a specification corresponds to an infinite transition system (also called 'unsafe' CCSL specification in [16]), a bound needs to be set to make an approximate analysis. The latter technique [9], [12] tackles the problem of expressing 'unsafe' specification. It proposes encoding the CCSL specification into a first-order logic (FOL) formula, which can then be checked through an SMT-checking procedure. However, the FOL formulae there contain quantifiers and so-called undefined functions, which are undecidable and may be very costly for an SMT-solver to solve.

Identify applicable funding agency here. If none, delete this.

Comparing with previous approaches, the advantage of our approach is that cDL calculus offers a nice formalism to capture the dynamic behaviour of CCSL specifications, while providing a 'deduction' approach for reducing a dynamic cDL formula into QF-LIA logic formulae which are decidable and easy to solve for an SMT-solver.

cDL is partially based on the 'CCSL dynamic logic' (CDL) [17], which can capture and verify a simple CCSL specification Rel of a given system p in the form '$[p]Rel$' (which means 'all execution traces of p satisfy a clock relation Rel'). However CDL fails in handling the schedule problem of CCSL in the form of '$\langle p\rangle\Box\varphi$' (introduced in Sect. IV), whose verification must rely on a different proof calculus that contains the dynamic formula $\langle p\rangle\phi\sqcap\Box\varphi$ from dTL^2. The rest of this paper is organized as follows: Sect. II introduces the preliminaries for CCSL. Sect. III gives an example which will be used throughout the paper to explain our viewpoints. Sect. IV defines the syntax and semantics of cDL. In Sect. V, we propose the proof calculus of cDL and analyze its soundness, completeness and automaticity. Sect. VI proposes a method for the schedulability analysis of CCSL in cDL. Sect. VII concludes this paper and discusses possible future works.

II. Preliminaries of CCSL

The CCSL presented here is based on [9], [18].

Logical clock A logical clock actually models a sequence of occurrences of an event in RTESs over a discrete time model. A clock $c : \mathbb{N}^+ \to \{0, 1\}$ is defined as an infinite sequence, where each $c(i)$ ($i \in \mathbb{N}^+$) can be 'tick' (represented as 1) or 'idle' (represented as 0). We use $\mathcal{C}$ to denote a finite set of clocks.

Schedule A schedule $\sigma : \mathbb{N} \to (\mathcal{C} \to \{0, 1\})$ is an infinite sequence that captures the state of all clocks $\eta : \mathcal{C} \to \{0, 1\}$ at each instant. $\mathbb{N} ::= \mathbb{N}^+ \cup \{0\}$. We also denote η by the set of ticked clocks $\alpha^\eta ::= \{c \mid \eta(c) = 1\}$ since they are one-to-one correspondent. If $\alpha = \{c\}$, simply write it as c. We stipulate that $\sigma(0) = \emptyset$, indicating at the beginning, no clock ticks.

Configuration A configuration $H_\sigma : \mathbb{N} \to (\mathcal{C} \to \mathbb{N})$ keeps track of the number of times each clock has ticked $h : \mathcal{C} \to \mathbb{N}$ at each instant in schedule σ, where H_σ is defined as:

$$H_\sigma(i, c) ::= \begin{cases} 0, & \text{if } i = 0 \\ H_\sigma(i-1, c) + 1, & \text{if } i > 0,\ c \in \sigma(i) \\ H_\sigma(i-1, c), & \text{if } i > 0,\ c \notin \sigma(i) \end{cases} .$$

Clock Constraint Clock relations describe binary relationships between clocks, whose syntax is defined by:

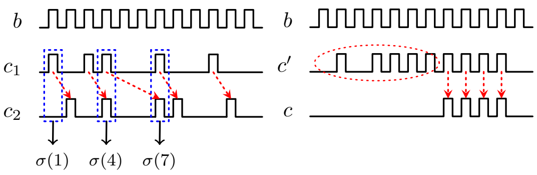
$$Rel ::= c_1 \prec c_2 \mid c_1 \preceq c_2 \mid c_1 \subseteq c_2 \mid c_1 \# c_2,$$

where c_1, c_2 are arbitrary clocks. The semantics of clock relations $\sigma \vDash_{ccsl} Rel$ is defined in Fig. 2. 'Precedence' means that c_1 always ticks faster than c_2; 'Causality' expresses that c_1 ticks not slower than c_2; 'Subclock' says that c_1 can only tick if c_2 ticks; 'Exclusion' means that c_1, c_2 can not tick at the same instant.

Clock definition defines new clock behaviour by composing the old clocks in different ways. A clock definition is of the form: $Cdf ::= c \triangleq E$ where E is a clock expression defined by the following grammar:

$$E ::= c_1 + c_2 \mid c\$n \mid c_1 \vee c_2 \mid ...,$$

where c_1, c_2 are arbitary clocks, $n \geq 1$. Above we only list 3 clock expressions used in this paper, for other clock expressions refer to [9], [18]. The semantics of clock definitions $\sigma \vDash_{ccsl} Cdf$ are defined in Fig. 2. 'Union' defines the clock that ticks iff either c_1 or c_2 ticks; 'Delay' defines the clock that ticks when c' ticks but is delayed for n ticks of c'; 'Supremum' defines the fastest clock that is slower than both c_1 and c_2.



(a) $c_1 \prec c_2$ (b) $c \triangleq c'\$n$ ($n = 5$)

Fig. 1. A possible schedule of CCSL constraints

Precedence:	$\sigma \vDash_{ccsl} c_1 \prec c_2$ iff $\forall i \in \mathbb{N}^+. \left\{\begin{array}{l} (H_\sigma(i, c_1) = 0 \wedge H_\sigma(i, c_2) = 0) \vee \\ H_\sigma(i, c_1) > H_\sigma(i, c_2) \end{array}\right\}$
Causality:	$\sigma \vDash_{ccsl} c_1 \preceq c_2$ iff $\forall i \in \mathbb{N}^+.\{H_\sigma(i, c_1) \geq H_\sigma(i, c_2)\}$
Subclock:	$\sigma \vDash_{ccsl} c_1 \subseteq c_2$ iff $\forall i \in \mathbb{N}^+.\{c_1 \in \sigma(i) \to c_2 \in \sigma(i)\}$
Exclusion:	$\sigma \vDash_{ccsl} c_1 \# c_2$ iff $\forall i \in \mathbb{N}^+.\{c_1 \notin \sigma(i) \vee c_2 \notin \sigma(i)\}$
Union:	$\sigma \vDash_{ccsl} c \triangleq c_1 + c_2$ iff $\forall i \in \mathbb{N}^+. \left\{\begin{array}{r} c \in \sigma(i) \leftrightarrow (c_1 \in \sigma(i) \vee \\ c_2 \in \sigma(i)) \end{array}\right\}$
Delay:	$\sigma \vDash_{ccsl} c \triangleq c'\$n$ iff $\forall i \in \mathbb{N}^+.\{H_\sigma(i, c) = max(H_\sigma(i, c') - n, 0)\}$
Supremum:	$\sigma \vDash_{ccsl} c \triangleq c_1 \vee c_2$ iff $\forall i \in \mathbb{N}^+.\{H_\sigma(i, c) = min(H_\sigma(i, c_1), H_\sigma(i, c_2))\}$

Fig. 2. Semantics of CCSL

Example Fig. 1 (a) illustrates the clock relation $c_1 \prec c_2$, where clock b is used as a 'base clock' that ticks at every time instant. b = 111111111111.... $\mathcal{C} = \{c_1, c_2\}$. c_1 = 101100100100..., c_2 = 010100110010.... The schedule $\sigma = \emptyset\{c_1\}\{c_2\}\{c_1\}\{c_1, c_2\}\emptyset\emptyset\{c_1, c_2\}\{c_2\}\emptyset\{c_1\}\{c_2\}\emptyset....$ The configuration H_σ satisfies that $H_\sigma(0, c_1) = 0$, $H_\sigma(1, c_1) = 1$, $H_\sigma(2, c_1) = 1$, $H_\sigma(3, c_1) = 2$. Fig. 1 (b) illustrates the clock definition $c \triangleq c'\$n$ (when $n = 5$).

Clock Specification & Free Clock Given a set of clock constraints C, $\sigma \vDash_{ccsl} C$ is defined s.t. $\sigma \vDash_{ccsl} cn$ for all $cn \in C$. A CCSL specification is a triple $SP ::= \langle \widetilde{Cdf}, \widetilde{Rel}, \mathcal{F}\rangle$, where $\widetilde{Cdf}$ is a set of clock definitions, $\widetilde{Rel}$ is a set of clock relations. A 'free clock' does not appear on the left side of any clock definitions $c \triangleq E$. $\mathcal{F}$ is the set of all free clocks appeared in $\widetilde{Cdf} \cup \widetilde{Rel}$. $\sigma \vDash_{ccsl} SP$ is defined s.t. $\sigma \vDash_{ccsl} \widetilde{Rel}$ and $\sigma \vDash_{ccsl} \widetilde{Cdf}$. We use $\mathcal{C}(SP)$ to note all clocks appeared in SP.

Schedule Problem Given a CCSL specification $SP = \langle \widetilde{Cdf}, \widetilde{Rel}, \mathcal{F}\rangle$, the schedule problem is to ask whether

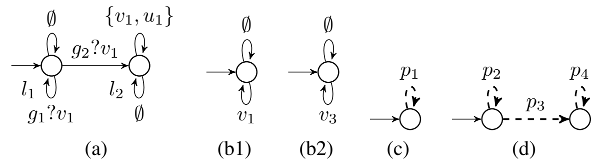
'there exists a schedule σ of $\mathcal{C}(SP)$ s.t. $\sigma \vDash_{ccsl} SP$'.

In a CCSL specification, there always exists a schedule if we allow the absence of all clocks (i.e., $\emptyset$) at any instant (e.g., an empty schedule $\sigma = \emptyset\emptyset\emptyset...$ satisfies any CCSL specification).

So in the schedule problem, we always talk about schedules σ that satisfy $\sigma(i) \neq \emptyset$ for any $i > 0$.

Clock-labeled Transition System (cLTS) The clock behaviour can be captured as a special type of finite transition systems: cLTS [18]. A cLTS is a tuple $\mathcal{A} = \langle L, T, l_0, \mathcal{C}\rangle$ where L is a set of locations, l_0 is the initial location. $T \subseteq L \times (G \times (\mathcal{C} \rightarrow \{0,1\})) \times L$ is a set of transitions, with $(l, g?\alpha, l') \in T$ (also denoted as $l \xrightarrow{g?\alpha} l'$) means that transition from l to l' is fired when guard g is true and all clocks in $\alpha \subseteq \mathcal{C}$ tick. Guard g is of the form $g ::= h(c_1, c_2) \triangle k$, where $k \in \mathbb{Z}$, $\triangle \in \{<, \leq, >, \geq, =\}$. $h(c_1, c_2) ::= h(c_1) - h(c_2)$. A schedule $\sigma = \alpha_1\alpha_2...\alpha_i...$ is accepted by a cLTS iff from the initial location l_0 there is a path $l_0 \xrightarrow{g_1?\alpha_1} l_1 \xrightarrow{g_2?\alpha_2} ... \xrightarrow{g_i?\alpha_i} l_i...$ in this cLTS, where g_i is any satisfiable guard of α_i (if there is). We use $Sch(\mathcal{A})$ to denote the set of schedules accepted by $\mathcal{A}$.

In cLTS we use a 'compositional transition' $[l, g_1?\alpha_1 \oplus ... \oplus g_n?\alpha_n, l']$ as a shorthand to express the set of transitions $(l, g_1?\alpha_1, l')$,...,$(l, g_n?\alpha_n, l')$ with the same locations l, l'.



$g_1 = h(v_1, u_1) < 4$, $g_2 = h(v_1, u_1) = 4$
$p_1 = \emptyset \oplus v_1 \oplus v_3 \oplus \{v_1, v_3\}$, $p_2 = \emptyset \oplus g_1?v_1 \oplus g_1?\{v_1, v_3\}$
$p_3 = \emptyset \oplus g_2?v_1 \oplus g_2?\{v_1, v_3\}$, $p_4 = \emptyset \oplus \{v_1, u_1\} \oplus \{v_1, u_1, v_3\}$

Fig. 3. An example of cLTSs

The synchronous product of $\mathcal{A}_1$,...,$\mathcal{A}_n$ is denoted by $\mathcal{A}_1 \parallel ... \parallel \mathcal{A}_n$. Intuitively, n cLTSs synchronize only on their common clocks when they all agree on whether the common clocks tick or not. Formally, let $\mathcal{A}_i = \langle L_i, T_i, l_{0,i}, \mathcal{C}_i\rangle$ $(1 \leq i \leq n)$, then $\mathcal{A}_1 \parallel ... \parallel \mathcal{A}_n$ is defined as a tuple $\langle L, T, l_0, \mathcal{C}\rangle$ where (1) $L = L_1 \times ... \times L_n$; (2) $(\langle l_1, ..., l_n\rangle, \bigwedge_{i=1}^{n} g_i? \bigcup_{i=1}^{n} \alpha_i, \langle l'_1, ..., l'_n\rangle) \in T$ iff $\langle l_i, g_i?\alpha_i, l'_i\rangle \in T_i$ for $1 \leq i \leq n$ and $\alpha_j \cap \mathcal{C}_k = \alpha_k \cap \mathcal{C}_j$ for any $1 \leq j \leq k \leq n$; (3) $l_0 = \langle l_{0,1}, ..., l_{0,n}\rangle$; (4) $\mathcal{C} = \bigcup_{i=1}^{n} \mathcal{C}_i$. Refer to [18] for more explicit explanations.

Example Fig. 3 (a) shows the cLTS of constraint $u_1 \triangleq v_1\$5$, where $L = \{l_1, l_2\}$, the initial location is l_1, $\mathcal{C} = \{u_1, v_1\}$. There are 5 transitions in T: $(l_1, \emptyset, l_1)$, $(l_1, g_1?v_1, l_1)$, $(l_1, g_2?v_1, l_2)$, $(l_2, \{v_1, u_1\}, l_2)$, $(l_2, \emptyset, l_2)$. Let $\sigma = v_1v_1v_1v_1\{v_1, u_1\}\{v_1, u_1\}...$, then $\sigma \in Sch(\mathcal{A})$. Fig. 3 (c) shows a synchronous product $\mathcal{A}_{b1} \parallel \mathcal{A}_{b2}$ where $\mathcal{A}_{b1}$, $\mathcal{A}_{b2}$ (Fig. 3 (b1), (b2)) are cLTSs of free clocks v_1, v_3. The dashed arrow represents a compositional transition $[l_0, p_1, l_0]$.

Proposition 2.1: Given a specification $SP = \langle \widetilde{Cdf}, \widetilde{Rel}, \mathcal{F}\rangle$, let $\mathcal{A}$ be the synchronous product of all clock behaviours (i.e., all definitions in $\widetilde{Cdf}$ and all free clocks in $\mathcal{F}$), then for any schedule σ of $\mathcal{C}(SP)$, $\sigma \vDash_{ccsl} \widetilde{Cdf}$ iff $\sigma \in Sch(\mathcal{A})$.

III. An Illustrative Example

In this section we consider an illustrative example from [18] which will be used through out this paper. As Fig. 4 shows, a component of a practical application contains two inputs $in1$, $in2$, three computations $step1$, $step2$, $step3$, two buffers and an output out. $step1$, $step2$ and $step3$ are three independent modules that run concurrently. $step1$ (resp. $step2$) needs the input from $in1$ (resp. $in2$) for computation and produces a result in the buffer. $step3$ needs the intermediate results from both $step1$ and $step2$ for computation and returns the result to the output out. The component continuously receives inputs and produces outputs in a streaming fashion.

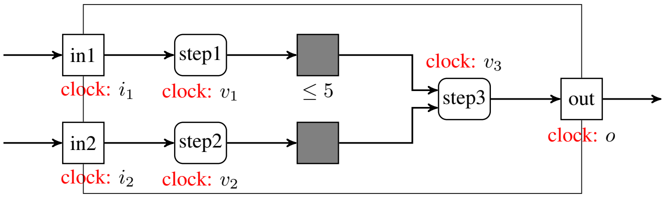


Fig. 4. A component of an application

By assigning each action with a clock, CCSL can be used to capture a high-level specification of this application. As a simple case, consider two basic specifications SP_1, SP_2 in the following table:

	$\widetilde{Cdf}$	$\widetilde{Rel}$	$\mathcal{F}$
SP_1	$u_1 \triangleq v_1\$5$	$v_1 \prec v_3$, $v_3 \preceq u_1$	v_1, v_3
SP_2	$u_1 \triangleq v_1\$5$, $u_2 \triangleq v_1 \vee v_2$	$v_1 \prec v_3, v_3 \preceq u_1, i_1 \preceq v_1,$ $i_2 \preceq v_2, u_2 \prec v_3, v_3 \preceq o$	$i_1, i_2, v_1,$ v_2, v_3, o

where SP_1 specifies a basic relation between $step1$ and $step3$: $step3$ must occur later than $step1$ $(v_1 \prec v_3)$, but before the buffer reaches its maximum capacity, which is 5 outputs of $step1$ $(v_3 \prec u_1)$. Clock u_1 is newly defined which is delayed by clock v_1 with 5 ticks $(u_1 \triangleq v_1\$5)$. SP_2 defines a more refined specification by adding more clock constraints in set $\widetilde{Cdf}$, $\widetilde{Rel}$. It declares all dependent relationships between all actions. Refer to [18] for more complex specifications of this example.

In this paper, we will show how our proposed logic framework can capture and analyze the schedule problem of SP_1.

IV. Syntax and Semantics of cDL

A. The Syntax of cDL

In order to capture the behaviour model of CCSL clocks (cLTS) in logic, we introduce a program model called 'clock program model' (CPM) based on the regular program model of FODL [13], [19].

Definition 4.1 (Syntax of CPM): The syntax of CPM is defined in BNF form as follows:

$$p ::= \alpha \mid g?\alpha \mid \varepsilon \mid p;p \mid p \oplus p \mid p^* \mid p^\omega.$$

The intuitive meaning of each sentence is as follows:

(1) α is the set of ticked clocks, g is the guard defined in cLTS (Sect. II). We also call α an 'event' in CPM. Each event consumes a unit of time. $g?\alpha$ means 'at current time, if g is true, then α executes, else the program halts'. The judgement of g does not consume any time.

(2) ε represents an 'empty program', it neither does anything nor consumes time. $;, \oplus, *$ are the sequence, non-deterministic choice and finite loop operators that are directly inherited from FODL. $p;q$ means the program first executes p, and after p terminates, it executes q. $p\oplus q$ means the program either executes p, or executes q, it is a non-deterministic choice. p^* means the program executes p for a finite number of times. ω is the infinite loop operator. p^ω means that program p executes for infinite number of times and never terminates.

cDL extends FODL with CPM as its program model and borrows the dynamic formula of the form $\langle p\rangle\phi \sqcap \Box\phi$ from dTL^2 [14] in order to express the schedule problem of CCSL.

Definition 4.2 (Syntax of cDL Formula): The cDL formula ϕ is defined in BNF form as follows:

$$\phi ::= tt \mid e \leq e \mid \mathfrak{d} \mid \neg\phi \mid \phi\wedge\phi \mid \forall x.\phi$$

where

$$e ::= x \mid h(c) \mid \eta(c) \mid k \mid e+e \mid e\cdot e,$$

$$\mathfrak{d} ::= \langle p\rangle\phi \sqcap \Box\phi \mid \neg\mathfrak{d} \mid \langle p\rangle\mathfrak{d}.$$

tt is the boolean true. e is an integer arithmetic expression. x is a general variable in the domain $\mathbb{Z}$. Use Var to denote a set of general variables. h, η are defined in Sect. II. Because clock itself does not appear in the cDL formula alone, we can see $h(c), \eta(c)$ as special variables related to clock $c\in\mathcal{C}$. Use $Var(\mathcal{C})$ to denote the set of all 'clock-related variables' $h(c), \eta(c)$ in $\mathcal{C}$. $k\in\mathbb{Z}$ is a constant. $\mathfrak{d}$ is a dynamic formula. $\langle p\rangle\mathfrak{d}$ describes a 'state property' of a clock program, meaning 'after some execution of p, $\mathfrak{d}$ holds'. $\phi\sqcap\Box\varphi$ describes a 'state & temporal' property of a clock program. It consists of a state formula ϕ and a temporal formula $\Box\phi$, with a conjunction operator $\sqcap$ linking the both. $\langle p\rangle\phi\sqcap\Box\varphi$ means that there exists some execution trace of p that satisfies $\Box\varphi$, and after the trace terminates (if it can), ϕ also holds. For non-terminating traces, $\phi\sqcap\Box\varphi$ just means $\Box\varphi$.

Example 4.1: The behaviour of clocks u_1, v_1, v_3 in SP_1 (whose cLTS corresponds to Fig. 3 (d)) can be captured as a CPM $p_{sp1} = p_2^*;(\varepsilon\oplus p_3;(p_4^*;\varepsilon\oplus p_4^\omega))\oplus p_2^\omega$, the schedule problem of SP_1 can then be captured as a dynamic formula $I_{sp1} \to \langle p_{sp1}\rangle ff \sqcap \Box(\varphi_{sp1}\wedge\varphi_\emptyset)$, which means 'under the initial condition I_{sp1}, there exists an infinite trace of p_{sp1} satisfying $\Box(\varphi_{sp1}\wedge\varphi_\emptyset)$. More details will be given in Sect. VI.

Formula '$[p]$...' is the dual formula of '$\langle p\rangle$...', which means 'all execution traces of p satisfy ...'. $\phi\sqcup\Diamond\varphi$ is the dual of formula $\phi\sqcap\Box\varphi$, in which $\sqcup$ is a disjunction operator. $[p]\phi\sqcup\Diamond\varphi$ means for each trace of p, it either satisfies $\Diamond\varphi$, or terminates and satisfies ϕ. $[p]\phi\sqcup\Diamond\varphi$ can be expressed as $\neg\langle p\rangle\neg\phi\sqcap\Box\varphi$. Other logical connectives and terms such as ff (falsehood), $\phi_1\vee\phi_2$, $\phi_1\to\phi_2$, $\exists x.\phi$, e_1-e_2, e_1/e_2, $e_1=e_2$, $e_1<e_2$,... can be expressed as the formulae in Def. 4.2.

In cDL, we call a variable X 'bounded' if (1) $X\in Var$ and X is in the scope of some quantifier $\forall X$, or (2) $X\in Var(\mathcal{C})$. A substitution $\phi[e/X]$ replaces every non-bounded X of ϕ with expression e. An 'admissible substitution' guarantees the meaning of a formula does not change. $\phi[e/X]$ is 'admissible' iff there exists no variable Y s.t. (1) Y is in e; and (2) Y is bounded in $\phi[e/X]$. Unless specially mentioned, all substitutions in this paper are admissible.

B. The Semantics of cDL

Kripke Frame & Trace The semantics of cDL is based on the Kripke frame (S, val) [19], where S is a set of states, val interprets a program as a set of traces on S and a logic formula as a subset of S. A trace tr is a finite or infinite sequence of states. Given a finite trace $tr_1 = s_1s_2...s_n$ and a (possibly infinite) trace $tr_2 = u_1u_2...u_n...$, we define: $tr_1\cdot tr_2 ::= s_1s_2...s_nu_2u_3...$ provided that $s_n = u_1$. Given any tr_1, tr_2, we define $tr_1\circ tr_2 ::= \begin{cases} tr_1\cdot tr_2, & \text{if } tr_1 \text{ is finite} \\ tr_1, & \text{otherwise}\end{cases}$. Given two sets of traces S_1, S_2, $S_1\circ S_2$ is defined as: $\{tr_1\circ tr_2 \mid tr_1\in S_1, tr_2\in S_2\}$. $tr(i)$ denotes the i^{th} element of trace tr, $i\geq 0$. tr_b denotes the first element of trace tr, $tr_b = tr(0)$. tr_e denotes the last element of trace tr, provided that tr is finite.

Definition 4.3 (State and Evaluation in cDL): Given a set of clocks $\mathcal{C}$ and a set of variables Var, a state s in cDL is defined as a total function as follows: (i) s maps each variable $h(c)\in Var(\mathcal{C})$ to a value in domain $\mathbb{N}$. (ii) s maps each variable $\eta(c)\in Var(\mathcal{C})$ to a value in domain $\{0,1\}$. (iii) s maps each variable $x\in Var$ to a value in domain $\mathbb{Z}$. Given an expression e and a state s, an evaluation $Eval_s(e)$ is defined as: (1) If $e=a$, where $a\in\{x, h(c), \eta(c)\}$, then $Eval_s(a) ::= s(a)$. (2) If $e=k$, then $Eval_s(k) ::= k$. (3) If $e = e_1\triangle e_2$, where $\triangle\in\{+,\cdot\}$, then $Eval_s(e) ::= Eval_s(e_1)\triangle Eval_s(e_2)$.

According to Def. 4.3, we can see that there exists a natural correspondence between traces in cDL and schedules in CCSL.

Proposition 4.1 (Relation between traces and schedules): Given a $\mathcal{C}$ and a Var, each schedule σ of $\mathcal{C}$ corresponds to a set of infinite traces Tr^σ, in which each element $tr\in Tr^\sigma$ satisfies that for all clock $c\in\mathcal{C}$ and $i\in\mathbb{N}^+$, there is:

(i) $tr(i)(\eta(c)) = 1$ iff $c\in\sigma$;

(ii) $tr(i)(h(c)) = H_\sigma(i,c)$.

For each infinite trace tr, there must be a σ s.t. $tr\in Tr^\sigma$.

Intuitively, except for $\sigma(0)$, variable $h(c), \eta(c)$ at each state of a trace $tr\in Tr^\sigma$ exactly record the information of schedule σ at each instant $i>0$.

Definition 4.4 (Semantics of CPM): Given a $\mathcal{C}$ and a Var, for any CPM p, the semantics is given as a Kripke frame (S, val) defined as follows:

(i) $val(\varepsilon) := S$.

(ii) $val(\alpha) := \{ss' \mid s, s'\in S;$ for any $c\in\alpha, s'(h(c)) = s(h(c))+1\wedge s'(\eta(c)) = 1;$ for any $d\in\mathcal{C}-\alpha, s'(h(d)) = s(h(d))\wedge s'(\eta(d)) = 0;$ for any $x\in Var, s'(x) = s(x)\}$.

(iii) $val(g?\alpha) ::= \{ss' \mid s\in val(g), ss'\in val(\alpha)\}$.

(iv) $val(p;q) ::= val(p)\circ val(q)$.

(v) $val(p\oplus q) ::= val(p)\cup val(q)$.

(vi) $val(p^*) ::= val(\varepsilon)\cup\bigcup_{n\geq 1}\underbrace{val(p)\circ ...\circ val(p)}_{n}$.

(vii) $val(p^\omega) ::= \underbrace{val(p)\circ val(p)\circ ....}_{\infty}$

We use '$\equiv$' to represent the semantical equivalence between two programs, i.e., $p\equiv q$ iff $val(p) = val(q)$. The semantics of each CPM corresponds to a set of traces. ε defines the set of all traces with length 1. Event α defines a transition from a state s to a state s'. Intuitively, at current instant, if clock c ticks ($c \in \alpha$), the variable $h(c)$ is increased by 1 and the variable $\eta(c)$ is set to 1; if clock c does not tick ($c \notin \alpha$), $h(c)$ does not change, and $\eta(c)$ is set to 0. Other variables in both s and s' are kept the same. Traces satisfying $g?\alpha$ are exactly those traces satisfying α adding that their beginning states must satisfy g. $val(g)$ will be given below in Def. 4.5. Each trace of $p; q$ is formed by concatenating a trace of p and a trace of q. Each trace of $p \oplus q$ is either a trace of p or a trace of q. The traces of program p^* are defined as all finite traces of the form s, or $tr_1 \circ tr_2 \circ ... \circ tr_n$ where $n \geq 1$, $tr_i \in val(p)$ is finite ($i \in \mathbb{N}^+$). The traces of p^ω consists of all infinite traces of the form $tr_1 \circ tr_2 ...$ where each $tr_i \in val(p)$ is finite ($i \in \mathbb{N}^+$), or of the form $tr_1 \circ tr_2 \circ ... \circ tr_n$ where $n \geq 1$, $tr_1, ..., tr_{n-1} \in val(p)$ is finite, but $tr_n \in val(p)$ is infinite.

Definition 4.5 (Semantics of cDL Formula): Given a $\mathcal{C}$ and a Var, the semantics of cDL formula is given as a Kripke frame (S, val) defined as follows:

(i) $val(tt) ::= S$.
(ii) $val(e_1 \leq e_2) ::= \{s \mid Eval_s(e_1) \leq Eval_s(e_2)\}$.
(iii) $val(\langle p \rangle \phi \sqcap \Box \varphi) ::= \{s \mid$ there is a $tr \in val(p)$ s.t. $s = tr_b, tr \vDash \Box\varphi$ and $tr_e \in val(\phi)$ if tr_e exists$\}$.
(iv) $val(\neg\phi) ::= \{s \mid s \notin val(\phi)\}$.
(v) $val(\phi \wedge \varphi) ::= val(\phi) \cap val(\varphi)$.
(vi) $val(\forall x.\phi) ::= \{s \mid$ for any $v_0 \in \mathbb{Z}, s \in val(\phi[v_0/x])\}$.

The semantics of each cDL formula corresponds to a set of states. (iii), (iv) are similar to the definition in dTL^2 [14]. In cDL, $tr \vDash \Box\varphi$ is defined as: every state s in tr ($s \neq tr_b$) satisfies $s \in val(\varphi)$. The dual formula $\Diamond\varphi$ of $\Box\varphi$ is defined as: $tr \vDash \Diamond\varphi$ iff there exists a state s in tr ($s \neq tr_b$) that satisfies $s \in val(\varphi)$. Different from dTL^2, in cDL a trace tr satisfying a temporal formula is from the second state of the trace. This stipulation is more nature for expressing clock constraints (see Example 6.2) since a schedule σ satisfying a clock constraints is from the second element of σ (Fig. 2). For a state property ϕ, we only consider its truth for terminating traces. (v), (vi) are similar to the definition in FODL [19], except that the semantics of CPM is based on traces.

V. Proof Calculus of cDL

Sequent Calculus & Rule We use Gentzen's sequent [20] as the logical argumentation for the proof calculus of cDL. A sequent has the form: $\Gamma \Rightarrow \Delta ::= \bigwedge_{\phi \in \Gamma} \phi \rightarrow \bigvee_{\varphi \in \Delta} \varphi$, where Γ, Δ are two finite multi-sets of logic formulae. A sequent $\Gamma \Rightarrow \Delta$ means that 'every formula in Γ holds can conclude that at least one of formulae in Δ holds'. When Γ or Δ is empty, we use $\cdot$ to denote it. A rule in sequent calculus is of the form: $\dfrac{\Gamma_1 \Rightarrow \Delta_1 \quad ... \quad \Gamma_n \Rightarrow \Delta_n}{\Gamma \Rightarrow \Delta}$, which means that if $\Gamma_1 \Rightarrow \Delta_1$, ..., $\Gamma_n \Rightarrow \Delta_n$ are all valid, so is $\Gamma \Rightarrow \Delta$. Each $\Gamma_i \Rightarrow \Delta_i$ in the upper part is called a 'premise', while $\Gamma \Rightarrow \Delta$ in the lower part is called 'conclusion'. We use $\dfrac{\Gamma' \Rightarrow \varphi \Rightarrow \Delta'}{\Gamma \Rightarrow \phi \Rightarrow \Delta}$ to represent a pair of sequent rules: $\dfrac{\Gamma', \varphi \Rightarrow \Delta'}{\Gamma, \phi \Rightarrow \Delta}$ and $\dfrac{\Gamma' \Rightarrow \varphi, \Delta'}{\Gamma \Rightarrow \phi, \Delta}$, i.e., ϕ, φ can be on both sides of the sequent. Sometimes we also write $\dfrac{\varphi}{\phi}$ to represent $\dfrac{\Gamma \Rightarrow \varphi \Rightarrow \Delta}{\Gamma \Rightarrow \phi \Rightarrow \Delta}$ (when Γ, Δ are kept unchanged). We call Γ, Δ the 'context' of formula ϕ in sequent $\Gamma \Rightarrow \phi, \Delta$ or $\Gamma, \phi \Rightarrow \Delta$.

Node & Proof Tree The derivation of a sequent forms a 'proof tree', where each sequent is a node, denoted by $\zeta = \langle \nu, \tau \rangle$, where ζ is the node name, ν is a vector of the child nodes of ζ, τ is a rule name. In a proof tree, a node $\zeta = \langle (\zeta_1, ..., \zeta_n), `(r)' \rangle$ is defined iff there is a derivation $\dfrac{\zeta_1 : \Gamma_1 \Rightarrow \Delta_1 \quad ... \quad \zeta_n : \Gamma_n \Rightarrow \Delta_n}{\zeta : \Gamma \Rightarrow \Delta}{\scriptstyle (r)}$, where (r) is the name of the rule, $\zeta_1, ..., \zeta_n$ is in sequence from left to right. We call node $\langle \nu, \tau \rangle$ a 'leave node' if $\nu = \emptyset$. If a leave node is obtained from a termination rule (rule (o), (ax), introduced below in Fig. 5 (c)), we also call it a 'valid node', denoted as $\surd$.

A. Proof Rules

Our main contribution for the proof calculus of cDL is rules for special primitives (α, $g?\alpha$, ε, p^ω) in cDL (Fig. 5 (a)). Other rules in cDL are either directly inherited or can be derived from the proof system of FODL [19], dTL^2 [14] and FOL (Fig. 5 (b) (c)).

In Fig. 5 (a), rule (α) says that to prove that under any context Γ, Δ, some trace of α satisfies $\phi \sqcap \Box\varphi$, iff to prove $\phi \wedge \varphi$ holds under the context after the execution of α. Variables in V are updated with new values according to α, while their old values are stored in V'. $\alpha = \{c_1, ..., c_n\}$, $\mathcal{C} - \alpha = \{d_1, ..., d_m\}$. $V = (h(c_1), ..., h(c_n), \eta(c_1), ..., \eta(c_n), \eta(d_1), ..., \eta(d_m))$ is a set of variables whose values change as the execution of α. $V' = (x_1, ..., x_n, y_1, ..., y_n, z_1, ..., z_m)$ is a set of new variables (w.r.t. Γ, $\langle \alpha \rangle \phi \sqcap \Box\varphi$, Δ) corresponding to V. $\Gamma[V/V']$ represents the context obtained by doing the substitution $\phi[V/V']$ for each formula ϕ in Γ, where $\phi[V/V']$ is the shorthand of $\phi[h(c_1)/x_1]...[h(c_n)/x_n][\eta(c_1)/y_1]...[\eta(d_m)/z_m]$. The vector equation $(x_1, ..., x_n) = (e_1, ..., e_n)$ means $x_1 = e_1, ..., x_n = e_n$.

Example 5.1: Consider sequent $h(c_1) = 0, \eta(c_1) = 0, h(c_2) = 0, \eta(c_2) = 0 \Rightarrow \langle c_1 \rangle \Box h(c_1) \geq h(c_2)$, by applying rule (α), we obtain the derivation:

$$\dfrac{\left\{\begin{matrix} x_1 = 0, y_1 = 0, \\ z_1 = 0, \end{matrix}\right\} \begin{matrix} h(c_1) = x_1 + 1, \eta(c_1) = 1, \\ \eta(c_2) = 0 \end{matrix} \Rightarrow h(c_1) \geq h(c_2)}{h(c_1) = 0, \eta(c_1) = 0, h(c_2) = 0, \eta(c_2) = 0 \Rightarrow \langle c_1 \rangle \Box h(c_1) \geq h(c_2)}{\scriptstyle (\alpha)},$$

where x_1, y_1, z_1 keep the old values of $h(c_1)$, $\eta(c_1)$, $\eta(c_2)$ respectively.

Rule $(\langle \varepsilon \rangle \sqcap)$ holds because we stipulate that the first element of any trace is unrelated to the temporal formula $\Box\varphi$ in the definition of $tr \vDash \Box\varphi$. Rule $(g?)$ moves the guard g outside of the dynamic part '$g?\alpha$' as a static formula g. In rule $(\langle \omega \rangle \sqcap)$ and $([\omega] \sqcup)$, the state property ϕ never works since an infinite loop program never terminates. Rule $(\langle \omega \rangle \sqcap)$ says that the conclusion holds if we can find an invariant *Inv* s.t.: (1) *Inv* holds under the current context Γ, Δ; (2) under any context, if *Inv* holds, then there exists a trace of p satisfying $\Box\varphi$ and

(a) Rules for special primitives in cDL

$$\frac{\begin{array}{c}\Gamma[V'/V],(h(c_1),...,h(c_n))=(x_1+1,...,x_n+1),\\(\eta(c_1),...,\eta(c_n))=(1,...,1),\\(\eta(d_1),...,\eta(d_m))=(0,...,0)\Rightarrow\phi\wedge\varphi\Rightarrow\Delta[V'/V]\end{array}}{\Gamma\Rightarrow\langle\alpha\rangle\phi\sqcap\Box\varphi\Rightarrow\Delta}\;(\alpha)\qquad\frac{\phi}{\langle\varepsilon\rangle\phi\sqcap\Box\varphi}\;(\langle\varepsilon\rangle\sqcap)\qquad\frac{g\wedge\langle\alpha\rangle\phi\sqcap\Box\varphi}{\langle g?\alpha\rangle\phi\sqcap\Box\varphi}\;(g?)$$

$$\frac{\zeta_1:\Gamma\Rightarrow Inv,\Delta\quad\zeta_2:\cdot\Rightarrow Inv\rightarrow\langle p\rangle Inv\sqcap\Box\varphi}{\zeta:\Gamma\Rightarrow\langle p^\omega\rangle\phi\sqcap\Box\varphi,\Delta}\;(\langle\omega\rangle\sqcap)$$

$$\frac{\begin{array}{c}\Gamma\Rightarrow\exists x.Inv(k),\Delta\quad\cdot\Rightarrow\forall x>0.(Inv(x)\rightarrow[p]Inv(x-1)\sqcup\Diamond\varphi)\\\cdot\Rightarrow(\exists x\leq 0.Inv(x))\rightarrow[p]\Diamond\varphi\end{array}}{\Gamma\Rightarrow[p^\omega]\phi\sqcup\Diamond\varphi,\Delta}\;([\omega]\sqcup)$$

(b) Rules mainly inherited from FODL and dTL^2

$$\frac{(\langle p\rangle\phi\sqcap\Box\varphi)\vee(\langle q\rangle\phi\sqcap\Box\varphi)}{\langle p\oplus q\rangle\phi\sqcap\Box\varphi}\;(\oplus)\qquad\frac{\langle p\rangle(\langle q\rangle(\phi\sqcap\Box\varphi)\sqcap\Box\varphi)}{\langle p;q\rangle\phi\sqcap\Box\varphi}\;(\langle;\rangle\sqcap)\qquad\frac{\phi\vee\langle p;p^*\rangle(\phi\sqcap\Box\varphi)}{\langle p^*\rangle(\phi\sqcap\Box\varphi)}\;(\langle *^n\rangle)$$

$$\frac{\Gamma\Rightarrow Inv,\Delta\quad\cdot\Rightarrow Inv\rightarrow[p]Inv\sqcup\Diamond\varphi\quad\cdot\Rightarrow Inv\rightarrow\phi}{\Gamma\Rightarrow[p^*]\phi\sqcup\Diamond\varphi,\Delta}\;([*]\sqcup)$$

$$\frac{\zeta_1:\Gamma\Rightarrow\exists x.Inv(x)\wedge\varphi,\Delta\quad\zeta_2:\cdot\Rightarrow\forall x>0.(Inv(x)\rightarrow\langle p\rangle Inv(x-1)\sqcap\Box\varphi)\quad\zeta_3:\cdot\Rightarrow(\exists x\leq 0.Inv(x))\rightarrow\phi}{\zeta:\Gamma\Rightarrow\langle p^*\rangle\phi\sqcap\Box\varphi,\Delta}\;(\langle *\rangle\sqcap)$$

(c) Rules of first-order logic

$$\frac{\models_{cdl}\bigwedge_{\phi\in\Gamma}\phi\rightarrow\bigvee_{\varphi\in\Delta}\varphi}{\Gamma\Rightarrow\Delta}\;(o)\qquad\frac{}{\Gamma,\phi\Rightarrow\phi,\Delta}\;(ax)\qquad\frac{\Gamma\Rightarrow\phi,\Delta\quad\Gamma,\phi\Rightarrow\Delta}{\Gamma\Rightarrow\Delta}\;(cut)\qquad\frac{\Gamma,\neg\phi\Rightarrow\Delta}{\Gamma\Rightarrow\phi,\Delta}\;(\neg r)\qquad\frac{\Gamma\Rightarrow\neg\phi,\Delta}{\Gamma,\phi\Rightarrow\Delta}\;(\neg l)$$

$$\frac{\Gamma\Rightarrow\phi,\Delta\quad\Gamma\Rightarrow\varphi,\Delta}{\Gamma\Rightarrow\phi\wedge\varphi,\Delta}\;(\wedge r)\qquad\frac{\Gamma,\phi,\varphi\Rightarrow\Delta}{\Gamma,\phi\wedge\varphi\Rightarrow\Delta}\;(\wedge l)\qquad\frac{\Gamma\Rightarrow\phi[x'/x],\Delta}{\Gamma\Rightarrow\forall x.\phi,\Delta}\;(\forall r)\qquad\frac{\Gamma,\forall x.\phi,\phi[e/x]\Rightarrow\Delta}{\Gamma,\forall x.\phi\Rightarrow\Delta}\;(\forall l)$$

Fig. 5. Proof Calculus of cDL

after p terminates, Inv holds. Rule $([\omega]\sqcup)$ is similar to $(\langle\omega\rangle\sqcap)$, where x indicates the number of repetitions of p before every trace of p satisfying $\Diamond\varphi$.

In Fig. 5 (b), rule $(\oplus)$ expresses that some trace of $p\oplus q$ satisfies ρ iff some trace of p or some trace of q satisfies ρ. Rule $(\langle;\rangle\sqcap)$ means that some trace of $p;q$ satisfies $\phi\sqcap\Box\varphi$ iff some trace of p satisfies $\Box\varphi$, and if it terminates, there is some following trace of q satisfies $\phi\sqcap\Box\varphi$. Rule $(\langle *^n\rangle)$ unwinds the loop program into a sequential one. Rule $([*]\sqcup)$ and $(\langle *\rangle\sqcap)$ proceed the proof by eliminating the loop operator $*$. They are similar to rule $(\langle\omega\rangle\sqcap)$ and $([\omega]\sqcup)$, but contain the terminating conditions (e.g. ζ_3 in rule $(\langle *\rangle\sqcap)$) for the state property ϕ. Inv is the loop invariant. In rule $(\langle *\rangle\sqcap)$, a number x is need to indicate the number of repetitions of p before p terminates.

In Fig. 5 (c), rule (o) is an 'oracle' rule indicating the termination of the proof, where all formulae in Γ,Δ must be QF-LIA logic formulae. Rule (o) means that to prove the validity of the conclusion, we check the validity of the QF-LIA logic formula in the premise. This process can be handled through an SMT-checking procedure, which is independent from the cDL proof calculus. (ax) is another termination rule. We omit the details of other traditional FOL rules.

B. Soundness, Completeness and Automaticity of cDL

The soundness of rules in Fig. 5 (b) (c) is directly from the proof calculus of FODL [19] and dTL^2 [14]. The soundness of rules in Fig. 5 (a) can be obtained directly from the semantics of cDL. Here we omit the detailed discussion of them.

Generally, like FODL, cDL is not complete due to Gödel's incompleteness theorem [21]. A sub-logic of cDL without operator ω is relatively complete to arithmetic FOL due to the relative completeness of dTL^2 [14]. However, it still remains open whether cDL is relatively complete to arithmetic FOL. The main reason is that for rule $(\langle\omega\rangle\sqcap)$ and $([\omega]\sqcup)$, we still do not know that for each CPM p, if there exists an invariant Inv s.t. the falsehood of the premise implies the falsehood of the conclusion, which is the key for proving the relative completeness of cDL.

FODL and dTL^2 is generally semi-automatic because loop invariant is undecidable in a program model that includes Presburger arithmetic theory. However, CPM only contains very simple arithmetic expressions (clock event α) and conditions (clock guard g). It is still not clear for us whether the invariant in cDL is generally decidable or not.

VI. A Method for Schedulability Analysis of CCSL in cDL

In this section we discuss how to describe and analyze the CCSL schedule problem in cDL calculus built in previous sections.

A. Encoding CCSL Specifications into cDL

The encoding of a CCSL specification $SP=\langle\widetilde{Cdf},\widetilde{Rel},\mathcal{F}\rangle$ can be accomplished in two steps:

i. Modeling the dynamic behaviour of all clocks $\mathcal{C}(SP)$ as a CPM p_{sp}. This can be done by encoding the synchronous product of all clock definitions in $\widetilde{Cdf}$ and all free clocks in $\mathcal{F}$;
ii. Encoding all static clock relations in $\widetilde{Rel}$ as a temporal formula $\Box\varphi_{sp}$.

In step i, the encoding from cLTS into CPM turns out to be a standard process by Brzozowski [22] for translating a type of transition system into the language that expresses it, where cLTS is expressed as an set of algebraic equations, and the latter is then solved by applying Arden's rule [23].

Proposition 6.1 (Arden's rule in CPM): Given any CPMs p,q (where $q\not\equiv\varepsilon$), $X\equiv q^*;p\oplus q^\omega$ is a solution of $X\equiv p\oplus q;X$ in CPM.

Prop. 6.1 is straightforward since CPM can be seen as an omega algebra [24]. Here we omit the proof of it.

Algo. 1 gives the main idea of the encoding from cLTS to CPM. Each compositional transition in cLTS corresponds to

Algorithm 1 Encoding cLTS into CPM

```
procedure CLTS_2_CPM($\mathcal{A} = \langle L, T, l_0, \mathcal{C}\rangle$)
    Build a set of equations from $\mathcal{A}$:
        $l_1 \equiv \varepsilon \oplus p_{11}; l_1 \oplus a_{12}; l_2 \oplus ... \oplus p_{1n}; l_n$    (1)
        $l_2 \equiv \varepsilon \oplus p_{21}; l_1 \oplus a_{22}; l_2 \oplus ... \oplus p_{2n}; l_n$    (2)
        ...
        $l_n \equiv \varepsilon \oplus p_{n1}; l_1 \oplus a_{n2}; l_2 \oplus ... \oplus p_{nn}; l_n$    ($n$)
    where $p_{ij}$ ($1 \le i \le j \le n$) is of the form $a_1 \oplus ... \oplus a_o$, with $a_k$
    ($1 \le k \le o$) in the form $\alpha$ or $g?\alpha$. In each equation, $l \equiv ... \oplus p; l' \oplus ...$
    iff there exists a compositional transition $[l, p, l'] \subseteq T$.
    for each $k$, $k = n, n-1, ..., 2, 1$ do
        substitute $l_{k+1}, ..., l_n$ in equation ($k$).
        transform equation ($k$) into the form $l_k \equiv p \oplus q; l_k$.
        By Prop. 6.1, obtain $l_k \equiv q^*; p \oplus q^\omega$ from $l_k \equiv p \oplus q; l_k$.
    return $l_1$
```

a relation $l \equiv p; l'$ in the equations. All such transitions from a state are linked by choice operator $\oplus$. Each equation (k) contains an empty program ε corresponds to that the state l_k is an accepting state for any schedules that pass by it.

Example 6.1: In SP_1, the behaviour of all clocks: $\mathcal{A}_{sp1}$ (Fig. 3 (d)), is the synchronous product of $u_1 \triangleq v_1\$5$ (Fig. 3 (a)) and free clocks v_1, v_3 (Fig. 3 (b1), (b2)). In Algo.1, from $\mathcal{A}_{sp1}$, we can build equations:

$$l_1 \equiv \varepsilon \oplus p_2; l_1 \oplus p_3; l_2 \quad (1)$$
$$l_2 \equiv \varepsilon \oplus p_4; l_2 \quad (2)$$

In (2) by Prop. 6.1 we obtain $l_2 \equiv p_4^*; \varepsilon \oplus p_4^\omega$. Substitute l_2 in (1) and by Prop.6.1 we obtain

$$p_{sp1} = l_1 \equiv p_2^*; (\varepsilon \oplus p_3; (p_4^*; \varepsilon \oplus p_4^\omega)) \oplus p_2^\omega.$$

According to Algo. 1, Prop. 6.1 and Prop. 4.1, it is easy to see that there is a natural connection between the semantics of a cLTS and its corresponding CPM.

Proposition 6.2 (Relation between cLTS and CPM): Let $\mathcal{A}$ be a cLTS and $p_{\mathcal{A}}$ be its corresponding CPM obtained from Algo. 1, then $\{tr \mid \text{there is a } \sigma \in Sch(\mathcal{A}) \text{ s.t. } tr \in Tr^\sigma\}$ is the set of all infinite traces accepted by $p_{\mathcal{A}}$.

Prop. 6.2 says that the behaviour of a cLTS exactly corresponds to the 'infinite behaviour' of its corresponding CPM obtained from Algo. 1. So if we can find an infinite trace of a CPM, we then find a schedule of its corresponding cLTS.

For step ii, the following proposition declares the relation between clock relations and temporal formulae in cDL.

Proposition 6.3 (Encoding clock relations as temporal formulae): Given a set of clock relations $\widetilde{Rel}$, we can build a temporal formula as: $\varphi_{\widetilde{Rel}} ::= \Box \bigwedge \hbar(Rel)$ s.t. $\sigma \models_{ccsl} \widetilde{Rel}$ iff $tr \models \varphi_{\widetilde{Rel}}$ for any σ and each $tr \in Tr^\sigma$. $\hbar(Rel)$ is defined as:

Rel	$\hbar(Rel)$	Rel	$\hbar(Rel)$
$c_1 \prec c_2$	$(h(c_1) = 0 \wedge h(c_2) = 0) \vee$ $(h(c_1) > h(c_2))$	$c_1 \preceq c_2$	$h(c_1) \ge h(c_2)$
$c_1 \subseteq c_2$	$\eta(c_1) = 1 \to \eta(c_2) = 1$	$c_1 \# c_2$	$\eta(c_1) = 0 \vee \eta(c_2) = 0$

Prop. 6.3 can be directly proved by Prop. 4.1, Def. 4.5 and the semantics of CCSL relations (Fig. 2).

Example 6.2: The clock relations $\{v_1 \prec v_3, v_3 \preceq u_1\}$ of SP_1 can be expressed as

$$\Box\varphi_{sp1} = \Box(\hbar(v_1 \prec v_3) \wedge \hbar(v_3 \preceq u_1)).$$

B. Solving the Schedule Problem

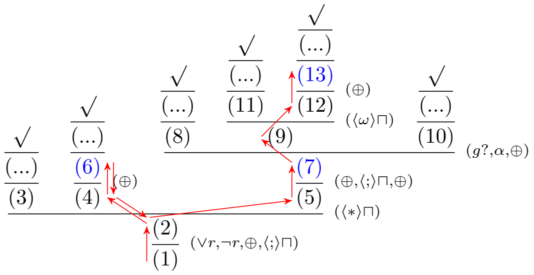
The next proposition states that the schedule problem stated in Sect. II can be verified by proving a cDL formula.

Proposition 6.4 (Schedule problem in cDL): Given a CCSL specification $SP = \langle \widetilde{Cdf}, \widetilde{Rel}, \mathcal{F}\rangle$, the schedule problem of SP (stated in Sect. II) holds iff the cDL formula

$$\phi_{SP} = I \to \langle p_{sp}\rangle(\mathit{ff} \sqcap \Box(\varphi_{sp} \wedge \varphi_\emptyset))$$

is valid, where p_{sp}, φ_{sp} are obtained from SP through step (i), (ii) in Sect. VI-A, $I = \bigwedge_{c \in \mathcal{C}(SP)}(h(c) = 0 \wedge \eta(c) = 0)$, $\varphi_\emptyset = \bigvee_{c \in \mathcal{C}(SP)} \eta(c) = 1$.

Prop. 6.4 is direct from Prop. 4.1, Def. 4.4, 4.5, Prop. 6.2, 6.3 and Prop. 2.1. I represents the initial environment of clock-related variables. The function of the falsehood ff in ϕ_{SP} is to filter all finite traces in p_{sp}, since the schedules we consider only correspond to infinite traces. Formula $\varphi_\emptyset$ means 'at least one clock ticks at an instant', which is used to avoid the schedules that contains $\emptyset$ at any instant (see Sect. III).



(1) $\cdot \Rightarrow I_{sp1} \to \langle(p_2^*; q) \oplus p_2^\omega\rangle\Box\varphi$, (2) $I_{sp1} \Rightarrow \langle p_2^*\rangle(\langle q\rangle\Box\varphi \sqcap \Box\varphi), \langle p_2^\omega\rangle\Box\varphi$,
(3) $\exists k.Inv_1(k)$, (4) $\cdot \Rightarrow \forall k > 0.(Inv_1(k) \to \langle p_2\rangle Inv_1(k-1) \sqcap \Box\varphi)$
(5) $\cdot \Rightarrow (\exists k \le 0.Inv_1(k)) \to \langle q\rangle\Box\varphi$
(6) $k > 0, Inv_1(k) \Rightarrow \langle g_1?v_1\rangle Inv_1(k-1) \sqcap \Box\varphi, \Delta_1$
(7) $Inv_1(0) \Rightarrow \langle g_2?\{v_1, v_3\}\rangle(\langle q'\rangle\Box\varphi \sqcap \Box\varphi), \Delta_2$, (8) $Inv_1(0) \Rightarrow g_2, \Delta_2$
(9) $\Gamma_1 \Rightarrow \langle p_4^\omega\rangle\Box\varphi, \Delta_3$, (10) $\Gamma_1 \Rightarrow \varphi, \Delta_4$, (11) $\Gamma_1 \Rightarrow Inv_2, \Delta_3$
(12) $\cdot \Rightarrow Inv_2 \to \langle p_4\rangle Inv_2 \sqcap \Box\varphi, \Delta_3$, (13) $Inv_2 \Rightarrow \langle\{v_1, v_3, u_1\}\rangle Inv_2 \sqcap \Box\varphi, \Delta_5$

$q = \varepsilon \oplus p_3; (p_4^*; \varepsilon \oplus p_4^\omega)$, $q' = p_4^*; \varepsilon \oplus p_4^\omega$, $\varphi = \mathit{ff} \sqcap (\varphi_{sp1} \wedge \varphi_\emptyset)$
$Inv_1(k) = (h(v_1, u_1) = 4 - k) \wedge h(v_1, v_3) \ge 0 \wedge h(v_3, u_1) \ge 0$
$Inv_2 = (h(v_1, u_1) = 5 - k) \wedge h(v_1, v_3) \ge 0 \wedge h(v_3, u_1) \ge 0$

Fig. 6. Derivation of formula $I_{sp1} \to \langle p_{sp1}\rangle \mathit{ff} \sqcap \Box(\varphi_{sp1} \wedge \varphi_\emptyset)$

If ϕ_{SP} is valid, the derivation of ϕ_{SP} actually provides a 'hint' of what the schedules of SP may 'look like'. Essentially, the successful proof tree (where each leave node is a valid node $\surd$) of ϕ_{SP} itself can be seen as a special 'transition system' that captures the behaviour of all schedules of SP. By analyzing this proof tree, one can generate a 'bounded schedule', i.e., a finite prefix of a schedule of SP.

The generating procedure is illustrated as a coarse algorithm in Algo. 2, where ζ_0 is a successful proof tree, n is a bound for schedule. In Algo. 2, we use ':=' to mean assignment and '=' to represent logical equality. Several commands are separated by the semicolon ';' in a single line. Starting from the root node ζ_0, procedure *Gen_Sch* traverses each node of the tree and continuously updates the bounded schedule σ according to the rule at each node. Only 4 rules need to be considered: $(g?)$, (α), $(\langle * \rangle \sqcap)$ and $(\langle \omega \rangle \sqcap)$. At line 7, we say '$g$ is a target of rule $(g?)$' if the event being dealt with is of the form $g?\alpha$. Line 10 is similar. At line 11, node $\zeta_1, \zeta_2, \zeta_3$ correspond to the child nodes of rule $(\langle * \rangle \sqcap)$ respectively (see rule $(\langle * \rangle \sqcap)$ in Fig. 5 (b)). Similar for line 16. $|\sigma|$ is the length of σ as a finite sequence.

Algorithm 2 Generating a bounded schedule from proof tree

```
 1: procedure GEN_SCH($\zeta_0 = \langle \nu_0, \tau_0 \rangle$, $\sigma$, $n$)
 2:     $\Xi := \{\zeta_0\}$ /*nodes remained to be analyzed*/
 3:     while $\Xi \neq \emptyset \wedge |\sigma| \leq n$ do
 4:         take a $\zeta = \langle \nu, \tau \rangle$ out of $\Xi$
 5:         if $\nu = \emptyset$  then continue /*a leave node*/
 6:         if $\tau =$'$(g?)$'$\wedge \sigma \not\models g$ then /*stop in analysis of this branch*/
 7:             continue /*$g$ is the 'target' of rule $(g?)$*/
 8:         else if $\tau =$'$(\alpha)$' then
 9:             put all nodes of $\nu$ in $\Xi$; $\sigma := \sigma\alpha$;
10:             continue /*$\alpha$ is the 'target' of rule $(\alpha)$*/
11:         else if $\tau =$'$(\langle * \rangle \sqcap)$' then /*set $\nu = (\zeta_1, \zeta_2, \zeta_3)$*/
12:             put $\zeta_3$ in $\Xi$;
13:             while $k \neq 0 \wedge |\sigma| \leq n$ do /*$k$ is a witness of '$\exists x.Inv(x)$'
    in $\zeta_1$*/
14:                 $\sigma' :=$ GEN_SCH($\zeta_2, \sigma$); $\sigma := \sigma\sigma'$; $k := k - 1$
15:             continue
16:         else if $\tau =$'$(\langle \omega \rangle \sqcap)$' then /*set $\nu = (\zeta_1, \zeta_2)$*/
17:             while $|\sigma| \leq n$ do
18:                 $\sigma' :=$ GEN_SCH($\zeta_2, \sigma$); $\sigma := \sigma\sigma'$
19:             continue
20:         else put all nodes of $\nu$ in $\Xi$; continue
21:     return $\sigma$
```

Example 6.3: We consider the schedule problem of SP_1. According to Prop. 6.4, it can be expressed as a cDL formula: $I_{sp1} \rightarrow \langle p_{sp1} \rangle ff \sqcap \Box(\varphi_{sp1} \wedge \varphi_{\emptyset})$, where $I_{sp1} = \bigwedge_{c \in \{v_1, v_3, u_1\}} h(c) = 0 \wedge \eta(c) = 0$, $\varphi_{\emptyset} = \bigvee_{c \in \{v_1, v_3, u_1\}} \eta(c) = 1$. p_{sp1}, φ_{sp1} has been given in Example 6.1, 6.2.

Fig. 6 shows a rough derivation of this formula, where each step abstractly represents one or more derivations, with the rules being applied listed on the right. e.g., from node (1), by applying rule $(\vee r)$, $(\neg r)$, $(\oplus)$ and rule $(\langle ; \rangle \sqcap)$ in sequence, we obtain node (2). Except for the first step, we omit all FOL rules in other steps. The derivation starts from the root (1), and terminates if (i) all leave nodes are valid $(\surd)$; (ii) one of the leave nodes is not valid. We omit the detail of each branch using (...). Fig. 6 (below part) shows the detail content of each node. $p_1 - p_4$ has already been given in Fig. 3. Due to the limit of space, we omit the details of the contexts Γ_1, $\Delta_1 - \Delta_5$.

The proof succeeds with all branches finally terminate. Let $\zeta_0 = (1)$, $\sigma = \emptyset$, $n = 7$, by calling procedure Gen_Sch we can obtain a bounded schedule of SP_1 as: $v_1 v_1 v_1 v_1 \{v_1, v_3\}\{v_1, v_3, u_1\}\{v_1, v_3, u_1\}$. The red path in Fig. 6 shows the process of running Algo. 2. In rule $(\langle * \rangle \sqcap)$, $k = 4$ is a witness of '$\exists k.Inv_1(k)$' at node (3). Node (6), (7) and (13) are 3 crucial nodes where the sets of ticked clocks v_1, $\{v_1, v_3\}$, $\{v_1, v_3, u_1\}$ are generated respectively.

VII. Conclusion and Future Work

In this paper we proposed a logical method for the schedulability analysis of CCSL specifications. We mainly focused on the construction of a dynamic logic cDL and its proof calculus. Based on cDL, we made a schedulability analysis of CCSL and shown how the derivation works through an example.

The future work may focus on the implementation of our cDL calculus in a theorem prover like Isabelle. We are also interested in analyzing the relative completeness of cDL and the decidability of the loop invariants in CPM.

References

[1] F. Mallet, "Clock constraint specification language: specifying clock constraints with UML/MARTE." *ISSE*, vol. 4, no. 3, pp. 309–314, 2008.

[2] OMG, "UML profile for MARTE: Modeling and analysis of real-time embedded systems," OMG, Tech. Rep., June 2011, formal/11-06-02.

[3] C. André, "Syntax and Semantics of the Clock Constraint Specification Language (CCSL)," INRIA, Research Report RR-6925, 2009.

[4] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[5] G. Berry and G. Gonthier, "The Esterel synchronous programming language: design, semantics, implementation," *Sci. Comput. Program.*, vol. 19, no. 2, pp. 87–152, 1992.

[6] J. Peters, R. Wille, N. Przigoda, U. Khne, and R. Drechsler, "A generic representation of ccsl time constraints for uml/marte models." in *DAC*. ACM, 2015, pp. 122:1–122:6.

[7] E.-Y. Kang and P.-Y. Schobbens, "Schedulability analysis support for automotive systems: from requirement to implementation." in *SAC*. ACM, 2014, pp. 1080–1085.

[8] H. Yu, J.-P. Talpin, L. Besnard, T. Gautier, H. Marchand, and P. L. Guernic, "Polychronous controller synthesis from marte ccsl timing specifications." in *MEMOCODE*. IEEE, 2011, pp. 21–30.

[9] M. Zhang, F. Mallet, and H. Zhu, "An SMT-based approach to the formal analysis of MARTE/CCSL." in *ICFEM '16*. Springer, 2016, pp. 433–449.

[10] L. Yin, J. Liu, Z. Ding, F. Mallet, and R. de Simone, "Schedulability analysis with ccsl specifications." in *APSEC (1)*. IEEE Computer Society, 2013, pp. 414–421, 978-1-4799-2143-0.

[11] M. Zhang, F. Dai, and F. Mallet, "Periodic scheduling for MARTE/CCSL: Theory and practice," *Sci. Comput. Program.*, vol. 154, pp. 42 – 60, 2018.

[12] M. Zhang and Y. Ying, "Towards SMT-based LTL model checking of clock constraint specification language for real-time and embedded systems." in *LCTES '17*. ACM, 2017, pp. 61–70.

[13] D. Harel, *First-Order Dynamic Logic*, ser. LNCS. Springer, 1979, vol. 68.

[14] J.-B. Jeannin and A. Platzer, "dtl2: Differential temporal dynamic logic with nested temporalities for hybrid systems." in *IJCAR*, ser. Lecture Notes in Computer Science, vol. 8562. Springer, 2014, pp. 292–306.

[15] C. Barrett, P. Fontaine, and C. Tinelli, "The SMT-LIB Standard: Version 2.6," Department of Computer Science, The University of Iowa, Tech. Rep., 2017, available at `www.SMT-LIB.org`.

[16] F. Mallet, J.-V. Millo, and R. de Simone, "Safe CCSL specifications and marked graphs," in *11th ACM/IEEE Int. Conf. on Formal Methods and Models for Codesign*. IEEE, 2013, pp. 157–166.

[17] Y. Zhang, H. Wu, Y. Chen, and F. Mallet, "Embedding CCSL into Dynamic Logic: A Logical Approach for the Verification of CCSL Specifications," in *FTSCS 2018*, Gold Coast, Australia, Nov. 2018.

[18] F. Mallet and R. de Simone, "Correctness issues on MARTE/CCSL constraints." *Sci. Comput. Program.*, vol. 106, pp. 78–92, 2015.

[19] D. Harel, D. Kozen, and J. Tiuryn, "Dynamic logic." *SIGACT News*, vol. 32, no. 1, pp. 66–69, 2001.

[20] G. Gentzen, "Untersuchungen über das logische schließen," Ph.D. dissertation, NA Göttingen, 1934.

[21] K. Gödel, "ber formal unentscheidbare sätze der principia mathematica und verwandter systeme," *Monatshefte fr Mathematik und Physik*, vol. 38, no. 1, pp. 173–198, 1931.

[22] J. A. Brzozowski, "Derivatives of regular expressions." *J. ACM*, vol. 11, no. 4, pp. 481–494, 1964.

[23] D. N. Arden, "Delayed-logic and finite-state machines," in *SWCT (FOCS)*. IEEE Computer Society, 1961, pp. 133–151.

[24] M. R. Laurence and G. Struth, "Omega algebras and regular equations." in *RAMICS*, ser. Lecture Notes in Computer Science, vol. 6663. Springer, 2011, pp. 248–263.