Timed Automata Modeling of CCSL Constraints

Jagadish Suryadevara
Mälardalen Real-Time Centre (MRTC)
Mälardalen University, Sweden
jagadish.suryadevara@mdh.se

Ling Yin
Institute of Software Engineering
East China Normal University, Shanghai, China
yinling86@gmail.com

The UML profile MARTE includes CCSL (Clock Constraint Specification Language) for specifying logical (synchronous/asynchronous) as well as chronometric timing constraints. A reference semantics for CCSL has been defined and transformation techniques proposed e.g. CCSL to Promela. In this paper, we present transformation of CCSL into timed automata, to enable verification with UPPAAL modelchecker. Further, we discuss how the transformation approach supports modeling multiple timebases, timebase relationships and corresponding timing constraints.

1 Introduction

Formal semantics and robust transformation techniques are prerequisites for successful application of formal methods in industrial safety-critical applications. For real-time and embedded systems (RTE), UML (Unified Modeling Language) provides MARTE (Modeling and Analysis of Real-Time and Embedded systems) Profile[1]. MARTE includes Clock Constraint Specification Language (CCSL), for specification of logical and physical clock constraints; synchronous, asynchronous, and chronometric time. Also, a reference semantics for a kernel subset of CCSL has been defined [1]. Further, Mallet & Yin have proposed a 'constructive approach' for transforming CCSL into Promela and also proved the correctness by $checkpoint-bisimulation$ approach[3].

In this paper, we describe a transformation technique for CCSL specifications into timed automata, the input language for UPPAAL modelchecker [2]. The approach is similar to the above mentioned transformations for Promela. However, the differences exist due to the flexibility and time modeling features in UPPAAL. For example, in addition to intuitive, visual specifications, it is simpler to model coincident instants using *urgent*(`u`) and $committed$(`c`) locations in UPPAAL. Further, UPPAAL supports modeling chronometric time using *clock* variables and clock *invariants*. Finally, we will also discuss how the proposed transformation can support modeling multiple timebases and timebase relationships as defined in an extension of TADL(Timing Augmented Description Language)[2] [4] .

2 CCSL: Clock Constraint Specification Language

CCSL supports specification of logical and physical timing constraints; while logical constraints address the functionality and associated causality, physical constraints concern the chronometric time. Timing constraints in CCSL are of three kinds: *synchronous*, *asynchronous* and *mixed*. Synchronous constraints rely on *coincidence* relation between the clock instants. For example, '`subclocking`' specifies that each instant of the subclock must coincide with one instant of the superclock. Asynchronous constraints are based on *precedence* relation between clock instants. For example, `s_precedence` specifies that an

[1]Specialization of a subset of UML elements for a given domain.
[2]In the context of EAST-ADL architectural modeling for Automotive domain.

Submitted to:
FTSCS 2012

© J. Suryadevara, L.Yin
This work is licensed under the
Creative Commons Attribution License.

instant of a clock (strictly) precedes the corresponding instant of another clock. Non-strict '`precedence`' is a mixed constraint; specifies that a clock instant precedes or coincides the corresponding instant of another clock. Another example of a mixed constraint: ' `c = a delayedFor n on b`' specifies that c coincides with the n-th tick of b following a tick of a, where a and b are asynchronous.

3 Checkpoint Transition Systems & CCSL

Formally, a CTS (Checkpoint Transition System) is a tuple $T = \{S, A, \rightarrow, I, clp\}$, where S is a finite set of states, $clp : S \rightarrow \{0,1\}$ such that $s \in S$ with $clp(s) = 1$ is called a *checkpoint*. $A = A \cup \tau$, A is a finite set of actions and τ is an invisible action. For each initial state $i \in I \subseteq S$, $clp(i) = 1$. $\rightarrow \subseteq S \times L \times S$ is the set of transitions, $L = \mathcal{P}(A)$. A label $l \in L$ is a set of actions, representing the simultaneously performed actions during that transition.

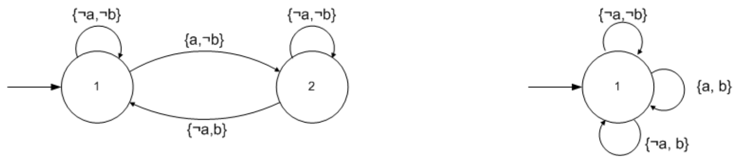


Figure 1: CCSL constraints: (a) `a alternatesWith b` (b) `a subClock b`

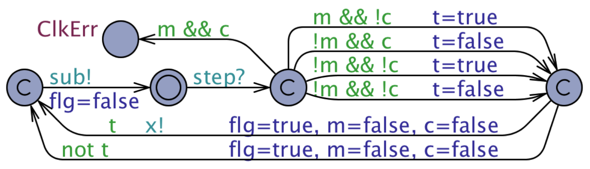
CCSL constraints can be modeled as CTS. For example, the CTS of '`a alternatesWith b`' is shown in Fig. 1. A transition in a CTS represents a valid ticking configuration of the clock constraint. Every state is marked as a checkpoint. If no clock ticks, the constraint is not violated. This intuitive formalism makes it easy for further transformations of CCSL; but, properties can only be verified at states corresponding to the *checkpoints* of the CTS. Further details of CTS & CCSL can be found in [3].

4 Timed Automata Specification of CCSL

A timed automaton is a tuple $< L, l_0, C, A, E, I >$, where L is a set of *locations*, $l_0 \in L$ is the initial location, C is the set of clocks, A is the set of actions, co-actions and the internal τ-action, $E \subseteq L \times A \times B(C) \times 2^C \times L$ is a set of edges between locations with an action, a guard, a set of clocks to be reset, and $I : L \rightarrow B(C)$ assigns clock *invariants* to locations. Further details of timed automata can be found in [2].

For each CCSL constraint, we define a timed automaton based on its CTS representation. It models the activation conditions, using the attributes '`must_tick`' and '`cannot_tick`'. For example, if both are `false`, a non-deterministic choice of 'tick' or 'not-tick' is made globally. Also, asynchronous constraints are state-based, hence the state is encoded and updated locally after each global 'fireable' step that changes the clock configuration. Synchronous constraints are state-less but depend on clock ordering, e.g. for `subclocking`, a subclock 'tick' enforces a superclock 'tick'.

Logical clocks, Instants, Clock ordering, and non-determinism. In Fig. 2 (a), we present timed automata modeling of logical clocks. A clock *instant* is modeled through a global non-deterministic *step*, also a timed automaton shown in Fig. 2 (b), using synchronization channel `fire`. Clock ticks are based on its attribute values `must_tick` (`m`) and `cannot_tick` (`c`); an inconsistency i.e., when both attribute values are *true* leads to the *location* 'ClkErr'. The clock tick itself is denoted by a boolean variable 't' and a (broadcast) synchronization channel 'x'. Thus the logical clocks (timed automata) composed with the global 'triggering' automaton represent the '$firable$' phase for the clocks.



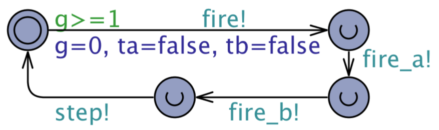
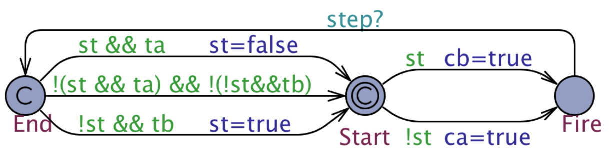


Figure 2: (a) Logical clock. (b) Instance activation and clock ordering

Modeling asynchronous constraints. In Fig. 3(a), we present a timed automata model of an asynchronous constraint `a alternatesWith b` (based on the CTS in Fig.1 (a)). In *enable* phase (location Start to Fire) the attributes '`cannot_tick`' (`c`) may be set i.e. 'ca' or 'cb' for logical clocks 'a' and 'b' based on the state 'st'. In *update* phase (End to Start), after the global $fireable$ phase, the state 'st' is updated based on the current state and the current tick values 'ta' and 'tb' (of 'a' and 'b' resp.).



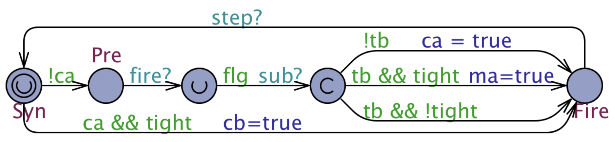


Figure 3: (a) `a alternatesWith b` (b) `a subClock b`

Modeling synchronous constraints. In Fig. 3(b), we present a timed automata model of a synchronous constraint: `a subClock b` (based on the CTS in Fig.1 (b)). It is state-less and depends on the clock ordering; $b < a$. The *sub-enable* phase (location Pre to Fire) occurs 'during' the global fireable phase (and hence so called) and immediately after the super clock 'b' $fires$. The model includes a stricter version of causality between 'a' and 'b' using the boolean value 'tight'; if 'a' cannot tick 'b' also cannot tick (ca ⇒ cb) and if 'b' ticks 'a' must tick as well (tb ⇒ ma).

5 TimeBases and TimeBase Relationships

In automotive domain, the need for explicit modeling of timebases and timebase relationships is identified. For example, in the context of East-ADL architectural modeling, the associated timing specification language TADL (Timing Augmented Description Language) has been extended to support timebases and timebase relationships [4]. The automata modeling of CCSL constraints as described in previous section can be extended to support modeling timebases and timebase relationships. This is due to clock variables in timed automata that can be used for modeling chronometric time.

Modeling Unit, Dimension, TimeBase and TimeBase relationships. In TADL, a TimeBase is defined in terms of a TimeDimension and Time Unit. For example, Eqn.1 below defines a timebase '$ecu1$'

in terms of dimension *universalTime* and *micros* as units. Similarly we can define timebase '*ecu*2'. A timebase relation between *ecu*1 and *ecu*2 is also defined below.

$$\texttt{TimeBase}\ \ ecu1\{\texttt{dimension}\ \ universalTime\ \ \texttt{precisionFactor}\ \ 0.1\ \ \texttt{precisionUnit}\ \ micros\}$$
$$\texttt{TimeBaseRelation}\ \ tbr\{(1.0ms\ \ on\ \ ecu1) = (2.0ms\ \ on\ \ ecu2)\} \tag{1}$$

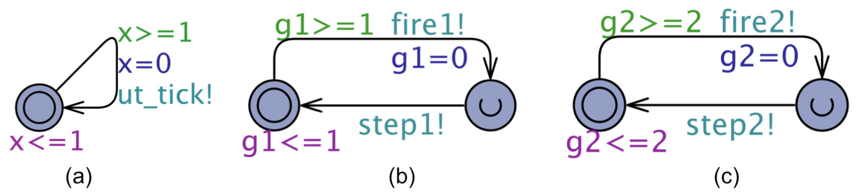


Figure 4: (a) *universalTime* `Dimension` (b) *ECU*1 `TimeBase` (c) *ECU*2 `TimeBase`

In Fig. 4.(a), we model *universalTime* as a timed automata[3] and clock variable x represents the units i.e. *micros* (ms). The timebases *ecu*1 and *ecu*2 are defined as global activation 'clocks', e.g., Fig. 2(b) hiding the (logical) clock ordering, for the timing constraints defined w.r.t timebases *ecu*1 and *ecu*2 respectively. When the automata in Fig.4 are composed, *ecu*1, *ecu*2 timebases are related w.r.t the time dimension `universalTime` as defined by the Eqn. 1 i.e., *ecu*1 timebase is twice faster than *ecu*2 timebase w.r.t the `universalTime`.

6 Conclusions

In this paper, we have presented an approach for timed automata modeling of CCSL timing constraints in MARTE (UML) profile for real-time and embedded systems. As timing properties are crucial in safety-critical systems, the approach paves the way for formal verification with UPPAAL, an efficient modelchecker. However, the approach needs to be validated through an industrial case study. Further, due to the underlying check-point transition system, property specification templates needs to be defined for different classes of properties that can be easily verified for the transformed CCSL specifications.

References

[1] Charles André (2009): *Syntax and Semantics of the Clock Constraint Specification Language (CCSL)*. Rapport de recherche RR-6925, INRIA. Available at `http://hal.inria.fr/inria-00384077`.

[2] Gerd Behrmann, Alexandre David & Kim G. Larsen (2004): *A Tutorial on UPPAAL*. In M. Bernardo & F. Corradini, editors: *International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004. Revised Lectures*, *Lecture Notes in Computer Science* 3185, Springer Verlag, pp. 200–237. Available at `http://doc.utwente.nl/51010/`.

[3] Frédéric Mallet & Ling Yin (2012): *Correct Transformation from CCSL to Promela for verification*. Rapport de recherche RR-7491, INRIA. Available at `http://hal.inria.fr/hal-00667849`.

[4] Marie-Agnés Peraldi-Frati, Arda Goknil, Julian Deantoni & Johan Nordlander (2012): *A Timing Language for Specifying Multi Clock Automotive Systems: The Timing Augmented Description Language*. In: *17th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*. Available at `http://hal.inria.fr/hal-00687562`.

[3]With timed automata modeling, a dimension can also be considered as an implicit timebase.