PSL vs. CCSL

Régis Gascon,Julien DeAntoni,Frédéric Mallet
EPI Aoste, I3S/INRIA, Sophia Antipolis, France

1 Introduction

The Property Specification Language (PSL) [5] is a language based on temporal logics and regular expressions. It is used for expressing assertions in the design of systems. The Constraint Specification Language (CCSL) has been defined as a companion language for the UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) [4]. It was designed as a specification language to express causal and chronological properties of real-time systems. CCSL has been used to verify time properties of Esterel and VHDL implementations [2, 3]. We give here some hints as what are the main differences between PSL and CCSL.

2 Property Specification Language

PSL has been defined as an IEEE standard [5]. Basically, the language is based on regular expressions and temporal logics. The *Sequential Extended Regular Expressions* (SERE) describe a behavioral properties on a finite part of the execution. They are built over the Boolean algebra defined by:

$$b ::= x \mid b \wedge b \mid b \vee b \mid \neg b$$

where x is a Boolean variable representing a signal of the system. In the following we consider a countably set of Boolean variables VAR. The set of SERE can be defined by the following grammar:

$$r ::= b \mid \{r\} \mid r;r \mid r:r \mid r|r \mid r\&\&r \mid [*0] \mid r[*].$$

The semantics is very similar to regular expressions. According to the definition of PSL, a run is driven by global cycles and at each cycle a subset of signals occur. We represent these runs with words of the form $\sigma : \mathbb{N} \to 2^{\mathrm{VAR}}$ (or $\sigma : \{0, \ldots, |\sigma|-1\} \to 2^{\mathrm{VAR}}$ when σ is finite) that associate to each global cycles a set of signals. This representation is similar to linear models for temporal logics. A SERE is evaluated over a finite subsequence of σ. For $i \leq j \leq |\sigma|$.

Brackets are equivalent to parenthesis. The operators $r_1; r_2$ (concatenation) and $r_1 : r_2$ (fusion) differ by the cycle where the second expression must start. With the fusion operator, the expression r_2 is evaluated at the cycle where r_1 ends and with the concatenation operator, r_2 is evaluated at the next cycle. The expression $r|r$ represents disjunction. The conjunction operator $r_1\&\&r_2$ is such that r_1 and r_2 must holds on paths of the same length. However, classical conjunction (non length-matching) can be expressed with $\{r_1; \mathrm{true}[*]\}\&\&\{r_2; \mathrm{true}[*]\}$. Finally, the operator $r[*]$ is the repetition of r zero or more instances and $[*0]$ represents the empty SERE which means that no cycle occurs in the system (equivalent to the regular expression ϵ).

We will consider this set of expressions as the kernel of SEREs. This kernel is obviously equivalent to regular expressions. In PSL, many additional commodity operators (not presented here) are defined to make specifications more concise. SEREs are the atomic formulas *PSL properties*. The kernel of PSL linear properties is defined by:

$$\phi ::= r \mid \mathrm{next!}\ \phi \mid \phi\ \mathrm{until!}\ \phi \mid \mathrm{not}\ \phi.$$

where r is a SERE. The classical definition of LTL is such that the set of atomic formulas is restricted to Boolean expressions (replace r by b in the definition above). However, the addition of SEREs as atomic formulas does not add expressiveness to the logic. In this paper, we will consider only the linear-time property but PSL also defines a similar CTL variant. PSL also defines syntactic sugar operators for temporal operators (*e.g.,* $\mathrm{always}\ \phi$, $\mathrm{next}\ \phi$, $\phi_1\ \mathrm{until}\ \phi_2$, $\phi_1\ \mathrm{before}\ \phi_2$). next! and until! correspond respectively to the classical LTL operators X and U. The operators next and until are their weak variants.

3 Clock Constraint Specification Language

A *clock* is a totally ordered set of *instants*. A *time structure* is a set of clocks C and relations on instants. The basic relations are *precedence* ($\prec$), *coincidence* ($\equiv$), and *exclusion* ($\#$). For any instants i and j in a time structure, $i \prec j$ means that the only acceptable execution traces are those where i occurs strictly before (precedes) j. $i \equiv j$ imposes instants i and j to be coincident, whereas $i \# j$ forbids the coincidence of the two instants. In this paper, we consider discrete sets of instants only, so that the instants of a clock can be indexed by natural numbers. For a clock c, $c[k]$ denotes its k^{th} instant.

Specifying a full time structure using only instant relations is not realistic since clocks are usually infinite sets of instants. Thus an enumerative specification of instant relations is forbidden. Hence the idea to extend relations to clocks. The Clock Constraint Specification Language (CCSL) has been defined to specify such relations between clocks. As an example, consider the clock relation *precedence* (denoted $\boxed{\prec}$). $a \boxed{\prec} b$, read 'a precedes b' or also 'a is *faster than* b', specifies that for all instants of clock a, its n^{th} instant *precedes* the n^{th} instant of clock b. More precisely: $a \boxed{\prec} b$ means $\forall k \in \mathbb{N}^\star, a[k] \prec b[k]$;

A CCSL specification consists of clock declarations and conjunctions of *clock relations* between *clock expressions*. A clock expression is a declared clock or a new clock defined from existing ones. An example of clock expression is *delay* (denoted $\$$). $a \ \$ \ n$ specifies that a new clock is created and is the exact image of a, delayed for n instants of a. Note that this expression is a simplified version of the expression *defer*, which specifies that a clock can be delayed for a number of instants counted on another clock. For simplicity, we give only the semantics of the *delay*: $a \ \$ \ n$ defines a clock c such that $\forall k \in \mathbb{N}^\star, c[k] \equiv a[k+n]$

A technical report [1] describes the syntax and the semantics of a kernel set of CCSL constraints.

A CCSL specification is a list of definitions and relations seen as a conjunction. A timed structure satisfies the specification iff it satisfies all the relations. CCSL specifications can be simulated by the tool TimeSquare to visualize the execution of a specification and possibly detect deadlocks. To display an execution the timed structured must be totally ordered. We call this a *trace*. A partially ordered timed structure corresponds to several traces since independent instants can be ordered in different ways. As PSL models are totally ordered, it is more convenient in the following to establish correspondence with traces rather than with general time structures.

4 Expressiveness of PSL and CCSL

To compare the expressiveness of PSL and CCSL we must define an equivalence relation between their models. Let T be a time structure over the set of clocks CLK.

As a trace T defines a total ordering of the instants of the clocks in CLK we can define for every instant i the function $\delta(i)$ as the number of distinct instants that strictly precedes i (one identifies the sets of instants that coincide). We say that T is equivalent to a temporal logic model σ iff there exist a bijection $f : \text{CLK} \to \text{VAR}$ such that:

(C) for every $n \in \mathbb{N}$ we have $f(c) \in \sigma(n)$ iff there is $i \in I_c$ such that $\delta(i) = n$.

A CCSL specification and a PSL property are equivalent iff for every model satisfying the specification there is an equivalent model satisfying the property and vice versa.

Most of the CCSL operators can be encoded by equivalent LTL formulas. However, some relations introduce unbounded parameters and cannot be encoded. For instance, the precedence relation cannot be expressed by an equivalent LTL formula since one need to count the respective occurrences of the clocks put in relation. Since PSL is as expressive as LTL, it cannot encode this relation too. The same problem occurs with other CCSL relations not introduced here ($c_1 \wedge c_2$, $c_1 \vee c_2$ in [1]).

Conversely, CCSL cannot express the whole PSL language too. There is no way in CCSL to enforce an occurrence of a clock. So, reachability or liveness properties cannot be expressed in CCSL. For instance, the formula $\mathtt{true\ until}\ p$ that is satisfied by every model where p holds at some position has no equivalent in CCSL. This means that there is no CCSL specification that is satisfied exactly by the time structures where the set of instants of a given clock is non empty.

Also, CCSL cannot express the next and until operators of temporal logics since it cannot ensure that the right part of the until will be reached or that a property is satisfied before the end of the model in the finite case. Note that in CCSL the next operator is not really meaningful when one has a partial ordering of the clocks. We can tackle this problem by imposing that next operators are linked to events represented by sets of clocks. This corresponds to the operators `next_event` defined in PSL.

References

[1] C. André. Syntax and semantics of the clock constraint specication language. Technical Report 6925, INRIA, 2009.

[2] C. André and F. Mallet. Specification and verification of time requirements with CCSL and esterel. In *Int. Conf. on Languages Compilers, and Tools for Embedded Systems (LCTES'09)*, volume 44, pages 167–176, Dublin, Ireland, June 2009. ACM DL.

[3] A. Mehmood Kahn, F. Mallet, C. André, and R. de Simone. IP-XACT components with abstract time characterization. In *Forum on specification, verification & Design Languages, FDL'09*. ECSI, IEEE Computer Press, September 2009.

[4] OMG. *UML Profile for MARTE, v1.0*. Object Management Group, November 2009. formal/2009-11-02.

[5] IEEE standard for Property Specification Language (PSL), IEEE std 1850-2005.