ProTL Tutorial

1 Overview of ProTL

ProTL (Probabilistic-CCSL TransLator) [9] is an automatic translation tool that provides a push-button transformation from PrCCSL (probabilistic extension of clock constraint specification language) [6, 7] specifications to UPPAAL-SMC [1] models (stored in *.xml* files). The main features of ProTL can be summarized as follows:

1. An *Editor* that allows one to edit PrCCSL specification of the requirements (e.g., functional properties, timing constraints and energy behaviors, etc) and store the encodings/specifications as *".txt"* files.
2. Automated transformation of PrCCSL specifications into verifiable UPPAAL-SMC models for formal verification. Two translation modes are provided and different outputs (*".xml"* file) are generated when different modes are selected:
   – PrCCSL mode: *Translator* translates PrCCSL specifications into UPPAAL-SMC models and generates an *".xml"* file contains the translated models.
   – PrCCSL+System mode: *Translator* generates models of the input PrCCSL specifications and integrates the models and the UPPAAL-SMC model of system behaviors (imported by users) into an *".xml"* file.
3. A configuration palette for setting parameters used for verification and simulation, e.g., time bound of simulation, number of simulations, etc.
4. Automatic generation of UPPAAL queries based on user-specified parameters, e.g., *Hypothesis Testing* and *Probabilistic Estimation* query.
5. Support for verification and simulation for PrCCSL specifications based on the generated model and queries.

2 Preliminary

2.1 Probabilistic Extension of Clock Constraints Specification Language (PrCCSL)

PrCCSL [6,7] is a probabilistic extension of CCSL (Clock Constraint Specification Language) [2,8], which specifies temporal and causal constraints associated with stochastic characteristics in the context of weakly-hard (WH) [3], i.e., a bounded number of timing constraints violations would not lead to system failures when the results of the violations are negligible. PrCCSL extends the standard CCSL constraints with probability parameters, specifying the boundaries on the number of constraints violations that can be tolerated in the context of WH.

In PrCCSL, a clock represents a sequence of (possibly infinite) instants. A clock represents an instance of `ClockType` characterized by a set of attributes, e.g., *resolution*, *offset* and *maximal value*. A clock in PrCCSL can be either in `DenseClockType` or in `DiscreteClockType`. `DenseClockType` defines new dense/chronometric clock types while `DiscreteClockType` defines new discrete/logical clock types. A dense/chronometric clock represents physical time, which is considered as a continuous and unbounded progression of physical instants. *idealClk* is a predefined dense clock whose unit is second. A discrete/logical clock represents an *event*, i.e., the occurrences of an event correspond to a countable set of discrete ticks/instants of the clock. A dense/chronometric clock can be discretized into a discrete/logical clock.

PrCCSL provides two types of clock constraints, i.e., *expressions* and *relations*, to specify the progression/occurrences of clocks.

Let $c1, c2 \in C$, an *expression* derives new clocks from the already defined clocks [2]:

1. `ITE` (if-then-else) *expression*, denoted as β ? $c1$: $c2$, defines a new clock that behaves either as $c1$ or as $c2$ according to the value of the boolean variable/formula β.
2. `DelayFor` (denoted $ref\ (d) \rightsquigarrow base$) results in a new clock by delaying the reference clock *ref* for d ticks (or d time units) of a *base* clock.
3. `FilterBy` ($c \triangleq base \blacktriangledown u(v)$) builds a new clock c by filtering the instants of a *base* clock according to a binary word w=$u(v)$, where u is the *prefix* and v is the *period*. "*(v)*" denotes the infinite repetition of v. This expression results in a clock c that $\forall\ k\ \in N^+$, if the k^{th} bit in w is 1, then at the k^{th} tick of *base*, c ticks.
4. `Intersection` expression (denoted as $c1 * c2$) builds a new clock that ticks whenever both $c1$ and $c2$ tick.
5. `Union` expression (denoted as $c1 + c2$) generates a new clock that ticks when either $c1$ or $c2$ ticks.
6. `periodicOn` (denoted $base \propto period$) builds a new clock based on a *base* clock and a *period* parameter, s.t., the instants of the new clocks are separated by a number of instants of the *base* clock. The number is given as *period*.
7. `Infimum`, denoted $c1 \wedge c2$, is defined as the slowest clock that is faster than both $c1$ and $c2$.
8. `Supremum`, denoted $c1 \vee c2$, is defined as the fastest clock that is slower than $c1$ and $c2$.

A *relation* limits the occurrences among different events (i.e., logical clocks), which are defined based on `run` and `history`. A `run` corresponds to an execution of the system model where the clocks tick/progress.

Definition 1 (`Run`). *A* `run` *R consists of a finite set of consecutive steps where a set of clocks tick at each step. The set of clocks ticking at step i is denoted as $R(i)$, i.e., for all i, $0 \leqslant i \leqslant n$, if clock* c *ticks at step i, then* $\mathrm{c} \in R(i)$*, where n is the number of steps of R.*

The history of a clock c represents the number of times the clock c has ticked prior to the current step.

Definition 2 (`History`). *For $c \in C$, the* `history` *of c in a run R is a function: H_R^c: $\mathbb{N} \rightarrow \mathbb{N}$. For all instances of step i, $i \in \mathbb{N}$, $H_R^c(i)$ indicates the number of* *times the clock c has ticked prior to step i in run R, which is initialized as 0 at step 0. It is defined as: (1) $H^c_R(0) = 0$; (2) $\forall\ i \in \mathbb{N}^+,\ c \notin R(i) \implies H^c_R(i+1) = H^c_R(i)$; (3) $\forall\ i \in \mathbb{N}^+,\ c \in R(i) \implies H^c_R(i+1) = H^c_R(i)\ +\ 1$.*

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` $= \{R_1, \ldots, R_k\}$, the probabilistic *relations* in PrCCSL, including `subclock`, `coincidence`, `exclusion`, `precedence` and `causality`, are defined as follows:

Probabilistic Subclock: $c1 \subseteq_p c2 \iff Pr[c1 \subseteq c2] \geqslant p$, where $Pr[c1 \subseteq c2] = \frac{1}{k}\sum_{j=1}^{k}\{R_j \models c1 \subseteq c2\}$ is the ratio of `runs` that satisfies the relation out of k `runs`. A `run` R_j satisfies the `subclock` relation between $c1$ and $c2$ "if $c1$ ticks, $c2$ must tick" holds at every step in R_j, i.e., $(R_j \models c1 \subseteq c2) \iff (\forall i\ 0 \leqslant i \leqslant n,\ c1 \in R(i) \implies c2 \in R(i))$. "$R_j \models c1 \subseteq c2$" returns 1 if R_j satisfies $c1 \subseteq c2$, otherwise it returns 0.

Probabilistic Coincidence: $c1 \equiv_p c2 \iff Pr[c1 \equiv c2] \geqslant p$, where $Pr[c1 \equiv c2] = \frac{1}{k}\sum_{j=1}^{k}\{R_j \models c1 \equiv c2\}$ represents the ratio of `runs` that satisfies the coincidence relation out of k `runs`. A `run`, R_j satisfies the `coincidence` relation on $c1$ and $c2$ if the assertion holds: $\forall i,\ 0 \leqslant i \leqslant n,\ (c1 \in R(i) \implies c2 \in R(i)) \wedge\ (c2 \in R(i) \implies c1 \in R(i))$. In other words, the satisfaction of `coincidence` relation is established when the two conditions "if $c1$ ticks, $c2$ must tick" and "if $c2$ ticks, $c1$ must tick" hold at every step.

Probabilistic Exclusion: $c1 \#_p c2 \iff Pr[c1 \# c2] \geqslant p$, 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 on $c1$ and $c2$ if $\forall i,\ 0 \leqslant i \leqslant n,\ (c1 \in R(i) \implies c2 \notin R(i)) \wedge\ (c2 \in R(i) \implies c1 \notin R(i))$, i.e., for every step, if $c1$ ticks, $c2$ must not tick and vice versa.

Probabilistic Precedence: $c1 \prec_p c2 \iff Pr[c1 \prec c2] \geqslant p$, 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` R_j satisfies the `precedence` relation if the two conditions $\forall i,\ 0 \leqslant i \leqslant n,\ (H^{c1}_R(i) \geqslant H^{c2}_R(i))$ and $(H^{c2}_R(i) = H^{c1}_R(i)) \implies (c2 \notin R(i))$ hold, i.e., the history of $c1$ is greater than or equal to the history of $c2$, and $c2$ must not tick when the history of the two clocks are equal.

Probabilistic Causality $\mathcal{M} \vDash c1 \preceq_p c2 \iff Pr[c1 \preceq c2] \geqslant p$, 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 on $c1$ and $c2$ if the condition $\forall i,\ 0 \leqslant i \leqslant n,\ H^{c1}_R(i) \geqslant H^{c2}_R(i)$ holds at every step, i.e., the history of $c2$ is less than or equal to the history of $c1$ at step i.

2.2 UPPAAL-SMC

UPPAAL-SMC [1, 4] 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 ϕ being satisfied for a given STA model within the time bound: $Pr[bound]\ \phi$.
2. *Hypothesis Testing* checks if the probability of ϕ being satisfied is larger than or equal to a certain probability P_0: $Pr[bound]\ \phi\ \geqslant\ P_0$.
3. *Probability Comparison* compares the probabilities of two properties being satisfied in certain time bounds: $Pr[bound_1]\ \phi_1 \geqslant Pr[bound_2]\ \phi_2$.
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 $\phi_1, ..., \phi_k$ along the simulations within simulation bound *bound*: $simulate\ N\ [\leqslant bound]\{\phi_1, ..., \phi_k\}$.

3 Download and Installation

ProTL is available for three types of operating systems: Windows, Mac and Linux. The installation package can be downloaded via `https://sites.google.com/view/protl/`.

The installation of ProTL can be completed by simply (1) selecting the installation directory and (2) following the installation instructions by clicking "next" button.

Note that if ProTL is installed in Disk C, "execute as an administrator" option is required when running the verification/simulation in ProTL.

4 ProTL Interface

ProTL provides a graphical user interface, which can be separated into two parts, i.e., *Translator* and *Verifier*.

4.1 Translator

Translator (shown in Fig.1) provides the service for users to edit textural encodings of PrCCSL specifications and automatically translate PrCCSL encod-

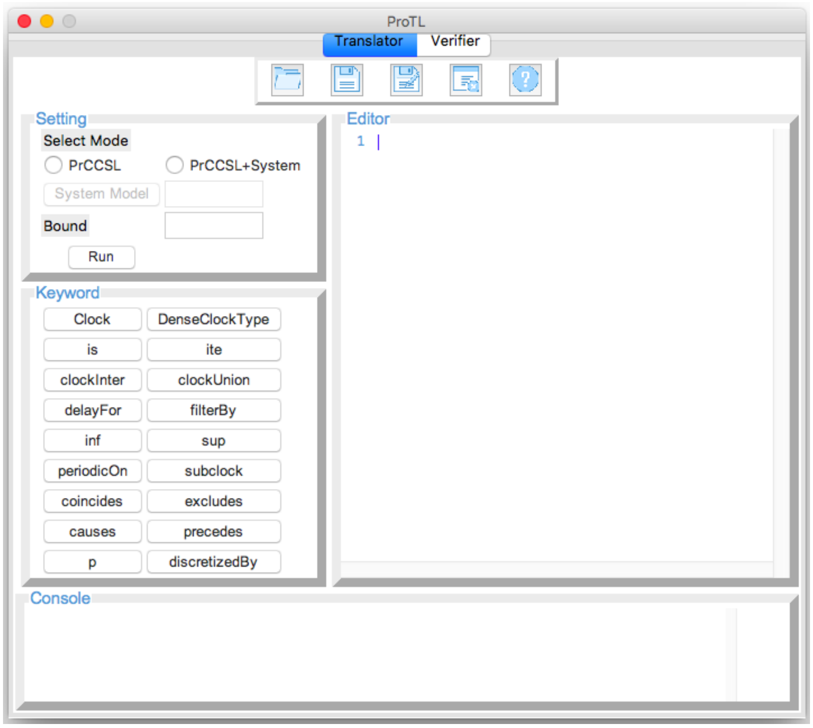


Fig. 1. Translator in ProTL

ings/specifications into corresponding UPPAAL-SMC models. In addition, *Translator* provide the option for users to import the behavioral models in UPPAAL-SMC (".xml") file and combine the generated UPPAAL-SMC model of PrCCSL specification and the imported file into an (".xml") file.

As shown in Fig. 1, *Translator* of ProTL consists of five parts:
1. *Tool Bar* (on the top): provides basic functions that enable user to import and save files;
2. *Setting* (on the upper left corner): allows users to configure the parameters for translation, i.e., *simulation bound*;
3. *Keyword* (below the Setting): offers the textural *keywords* in PrCCSL to facilitate users edit the PrCCSL encodings/specifications;
4. *Editor* (on the right): allows users to edit encodings in PrCCSL format;
5. *Console* (on the bottom): shows the errors/warnings information of the input PrCCSL encodings/specifications to help users to refine/fix the encodings.

Tool bar Fig .2 shows the five buttons in the *Tool Bar*:

1. **Open** button allows users to import the textural file (stored in *.txt* file) that contains PrCCSL specifications/encodings, into *Editor*.

(a) Open (b) Save (c) Save-as (d) Clear (e) Help

Fig. 2. Buttons in Translator Tool Bar

2. **Save** button saves the current PrCcsl encodings/specifications in the *Editor* as a new *".txt"* file in the directory designated by user. If the encodings are imported from existed file, clicking the "save" button overwrites the original file.
3. **Save-as** button saves the PrCcsl specifications as another *".txt"* file in the directory designated by user.
4. **Clear** button removes all encodings in *Editor* and parameters in *Setting*.
5. **Help** button directs the user to access the *ProTL Tutorial* [9], in which the detail of the tool usage can be found.

Setting enables users to specify parameters used in PrCcsl translation, which consists of three sections: **Select mode**, **Bound**, and **Run** button.

1. **Select Mode** offers two modes for translation:
   (a) PrCCSL Mode: ProTL translates PrCcsl specifications into STA (stored them as a *.xml* file) in the specified path. The generated *.xml* file can be read by Uppaal-SMC
   (b) PrCCSL+System Mode: ProTL translates PrCcsl specifications into STA and combines the generated STA with a system modeled as a network of STA (NSTA). Before the running the translation, the system model ("*.xml* file") needs to be imported by clicking *System model* button.
2. **Bound** represents the simulation time bound, which corresponds to the *bound* in *Hypothesis Testing* query. An example of setting *bound* to 3000 is shown in Listing. 1.

```
Hypothesis Testing: Pr[<=3000]([] property1)>=0.95
```

Listing 1. *An example of simulation bound*

3. **Run** button initiates the translation of PrCcsl specifications into Uppaal-SMC models.

Keyword provides a set of keywords (i.e., the textural expressions of corresponding PrCcsl operators) that provide the convenience for editing PrCcsl encodings/specifications, which can be divided into three categories: clock declarations, PrCcsl relations and expressions:

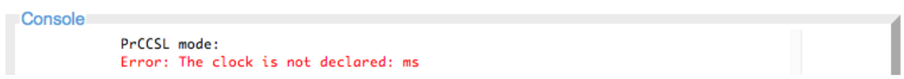
– **Declaration**: `Clock`, `DenseClockType`

– **Expressions**: `is`, `periodicOn`, `inf`, `sup`, `delayFor`, `clockInter`, `clockUnion`, `filterBy`, `ite`, `discretizedBy`
– **Relations**: `coincides`, `excludes`, `subclock`, `causes`, `precedes`, `p`

For further details of how to utilize the *keywords* to specify requirements, refer to Sec. 5.

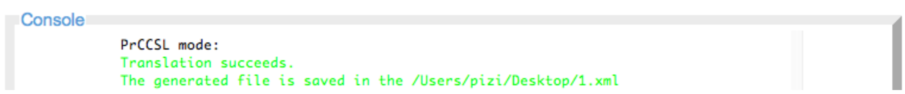
Editor allows user to create, edit specifications/encodings in PrCcsl formalism. For the syntax of the input encodings of ProTL, refer to Sec. 5.

Console In *Translator*, the error/warning information regarding to syntax errors of the encodings or configuration for translation is indicates in *Console*. The error information and warning messages help users refine the encodings. Fig.3 shows three types of messages that would occur during or after the translation, i.e., error messages, warning messages and messages that indicate the status of translation.



(a) Error message

(b) Warning message



(c) Status of translation

Fig. 3. Console messages

4.2 Verifier

Verifier offers the support of automatic generation of probabilistic queries that can be recognized in Uppaal-SMC and can perform verification and simulations on the generated Uppaal-SMC models and queries. Fig. 4 pictures the *Verifier* of ProTL consisting of three parts (from top to bottom): *Query Configuration*, *Query List* and *Console*.

Query Configuration enables specification of parameters for generation of five types queries (introduced in Sec. 2.2), which consists of six sub-sections:

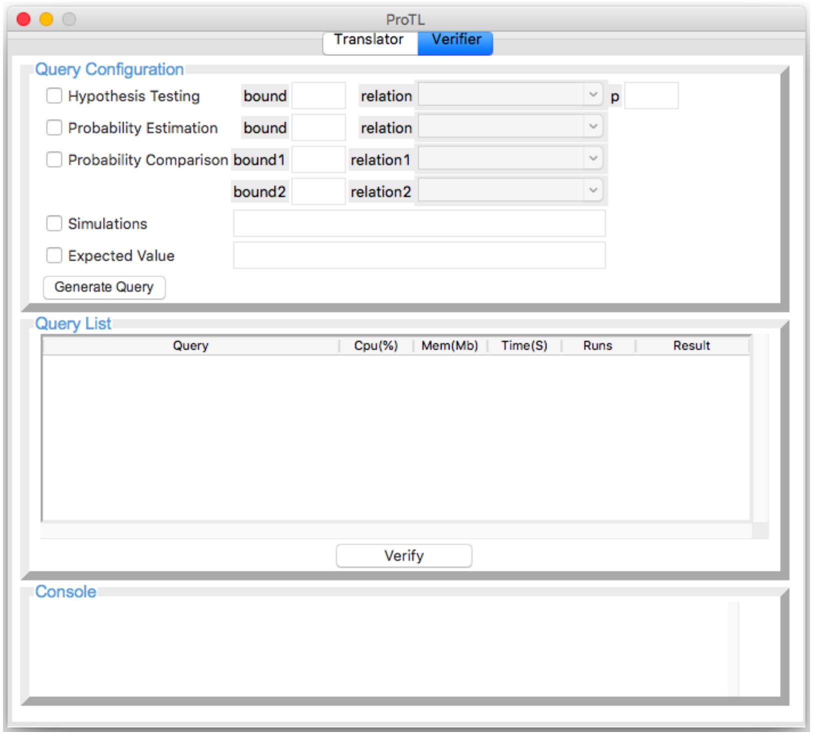


Fig. 4. Verifier in ProTL

Hypothesis Testing, *Probability Estimation*, *Probability Comparison*, *Simulations*, *Expected Value* and *Generate Query* button.

Hypothesis Testing allows users to specify parameters for generating *Hypothesis Testing* query. *bound* is a non-negative integer that represents the time bound of verification/simulation. A combobox *relation* allows users to select a *relation* from all the existed *relations* in the input PrCcsl specifications, i.e., only the *Hypothesis Testing* query of the designated *relation* is generated. p specifies a value of double-type, $p \in [0, 1]$, which represents the probability threshold of a *relation* being satisfied. Similarly, the parameters for generation of *Probability Estimation* query can be specified.

Probability Comparison allows users to specify parameters for *Probability Comparison* query. $bound1$ and $bound2$ are non-negative integers that represents the simulation time bounds for two *relations* selected via $realtion1$ and $realtion2$ comboboxes. *Simulations* enables users to specify *Simulations* query: *simulate* $N\ [\leqslant bound]\{\phi_1, ..., \phi_k\}$. *Expected Value* enables users to specify *Expected Value* query: $E[bound; N](min : \phi)$ or $E[bound; N](max : \phi)$. Furthermore, *Generate Query* button initiates the generation of queries based on the configured parameters.

Query List exhibits the set of generated queries and performs verification/simulation based on queries. Users can select (multiple) queries in *Query List* for verification/simuulation. The verification/simulation of the selected queries can be initiated by clicking the *Verify* button. After the verification/simulation is completed, the verification results, along with the verification performance (analysis time, CPU, memory consumption, etc.), are displayed in the table. Queries can be selected or de-selected using the left-button of mouse or clicking with *Shift* or *Contrl* key pressed (press the *Shift* key to (de-)select multiple queries and the *Contrl* key to (de-)select a single query).

Console illustrates the error/warning information that regarding to errors of verification and simulation. Those error information and messages help users refine the ProTL PrCcsl encodings, system model in Uppaal-SMC.

5 Input Language of ProTL

In this section, we presents the syntax of input languages of ProTL, i.e., PrCcsl encodings/specifications. We first describe the character set that are supported in the PrCcsl specifications. We then introduce how the specifications can be constructed by *tokens*. Finally, the usage a set of *keywords* (including declaration, expression and relations) to encode the specifications is demonstrated.

5.1 Character Set

A subset of the *ASCII* character set is utilized in the PrCcsl specifications:

1. Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
2. Letters: ‘a’ through ‘z’ and ‘A’ through ‘Z’.
3. Punctuation characters: ‘+’, ‘*’, ‘=’, ‘#’, ‘(’, ‘)’, ‘.’, ‘[’, ‘]’, ‘-’.
4. Whitespace characters: space, tab, newline, and carriage return.
5. Line terminators: ‘;’.

5.2 Token

In this section we describe the various tokens recognized by ProTL. A summary of token types is provided in Table. 1.

Table 1. Token types defined in ProTL

<table>
<tr><th>Type</th><th>Format</th><th>Regular Expression</th><th>Examples</th></tr>
<tr><td>Integer (Int)</td><td>A sequence of digits without leading zeros</td><td>0|([1-9][0-9]*)</td><td>0, 10, 33</td></tr>
<tr><td>Double</td><td>An integer followed by a decimal point, followed by a sequence of digits</td><td>(Int+).(Int+)</td><td>32.7, 0.001</td></tr>
<tr><td>Clock</td><td>A sequence of characters</td><td>(‘a’..‘z’|‘A’..‘Z’|‘0’..‘9’|‘[’|‘’]’|‘_’)+</td><td>Ctrig, c2_tt</td></tr>
<tr><td>Parameter (para)</td><td>Integers or operations between integers</td><td>Int|’(’ Int (’+’ Int)* ’)’)</td><td>(WECT+DOM), 22</td></tr>
</table>

5.3 Syntax of PrCCSL

We introduce the *keywords* in PrCCSL and their syntax and semantics. *Keywords* in PrCCSL can be categorized into three types, i.e., *clock declaration*, *expressions* and *relations*.

1. Clock Declarations in ProTL
   (a) "`Clock`" declares a new clock. A clock (i.e., an event) needs to be declared before being used in the PrCCSL *expressions* or *relations*. For example, three new clocks named $c1$, $c2$ and $c3$ can be declared as shown in Listing .2.

```
Clock c1 c2, c3;
```

Listing 2. Clock declaration

The *idealClk* in PrCCSL is an chronometric clock that derives from the *DenseClockType IdealClock*, which is declared in Listing .3.

```
Clock idealClk: IdealClock;
```

Listing 3. idealClk

In the C language, a quick sort function is built in

   (b) "`DenseClockType`" declares a new clock type, which is characterized by a set of attributes, e.g., *reference*, *factor*, *offset* and *reset*. An example of declaration of a new dense clock type is shown in Listing. 13.

```
DenseClockType NewType{
  reference idealClk,
  factor    2,
  offset    {(c1, 0.09), (c2, 7.8)},
  reset     {c3, c4}
  };
  Clock c2: NewType;
```

Listing 4. DenseClockType

2. Expressions in ProTL
   (a) "`is`" defines a new clock based on *expressions*.
   (b) "`periodicOn`" builds a periodic clock that ticks based on a *base* clock and a *period* parameter. An example of using "`periodicOn`" to define a new clock named c is shown in Listing. 5. The instants of c are separated by 50 instants of clock *base*.

```
Clock c;
Clock base;
c is periodicOn base period 50;
```

Listing 5. `periodicOn` expression

(c) "`inf`" represents the infimum *expression*, which defines a new clock c that is the slowest clock that is faster than both $c1$ and $c2$, as shown in Listing. 6.

```
Clock c, c1, c2;
c is c1 inf c2;
```

Listing 6. `inf` expression

(d) "`sup`" represents `supremum` *expression*, which defines the faster clock that is slower than both $c1$ and $c2$, as depicted in Listing. 7.

```
Clock c, c1, c2;
c is c1 sup c2;
```

Listing 7. `sup` expression

(e) "`delayFor`" results in a clock by delaying the *base* clock for a given number of ticks of a reference clock ref. An example of `delayFor` is given in Listing. 8.

```
Clock c;
Clock base, ref;
c is ref delayFor 50 on base;
```

Listing 8. `delayFor` expression

(f) "`clockInter`" denotes `intersection` expression, which defines a new clock that ticks if both two reference clocks tick. An example of defining c as the intersection clock of $c1$ and $c2$ is shown in Listing. 9.

```
Clock c, c1, c2;
c is c1 clockInter c2;
```

Listing 9. `clockInter` expression

(g) "`clockUnion`" represents CCSL union expression, which defined as a new clock ticks if either $c1$ or $c2$ ticks, as shown in Listing. 10.

```
Clock c, c1, c2;
c is c1 clockUnion c2;
```

Listing 10. `clockUnion` expression

(h) "`filterBy`" filters out undesired instants of clock $c1$ and leave the desired ticks. As shown in Listing. 11, c is defined as a clock that ticks from the 2^{nd} tick of $c1$, i.e., the 1^{st} tick of $c1$ is filtered.

```
Clock c, c1;
c is c1 filterBy 2(1);
```

Listing 11. `filterBy` expression

(i) "`ite`" (if-then-else) represents the conditional clock *expression*. As shown in Listing. 12, a conditional clock *expression* defines a clock that behaves either as a clock $c1$ or as another clock $c2$ according to the value of a boolean variable b.

```
Clock c, c1, c2;
c is if b then c1 else c2;
```

Listing 12. `ite` expression

(j) "`discretizedBy`" declares a discrete time unit that represents discrete and totally ordered set of event occurrences (often an event occurrence is called tick) based on physical/logical time. The discrete step of `discretizedBy` is specified with factor which is double number. An example of `discretizedBy` declaration for a clock ms that ticks every 1 milisecond is illustrated in Listing .13.

```
Clock ms;
ms is ideal discretizedBy 0.001.
```

Listing 13. `discretizedBy` expression

3. Probabilistic Relations in ProTL

(a) `Probabilistic subclock` *relation* ($c1 \subseteq_{0.95} c2$) specifies that the `subclock` relation between $c1$ and $c2$ must be satisfied with probability greater than or equal to 95%. "`p`" denotes the probability threshold on the *relations*. The corresponding textual encoding is shown in Listing. 14.

```
Clock c1, c2;
c1 subclock(p=0.95) c2;
```

Listing 14. `Probabilistic coincides` relation

(b) `Probabilistic coincidence` *relation* ($c1 \equiv_{0.95} c2$) specifies that the `coincidence` relation between $c1$ and $c2$ must be satisfied with probability greater than or equal to 95%. The corresponding textual encoding is shown in Listing. 15.

```
Clock c1, c2;
c1 coincides(p=0.95) c2;
```

Listing 15. `Probabilistic coincides` relation

(c) `Probabilistic exclusion` ($c1\#_{0.95}c2$) prevents the instants of two clocks from being coincident, as shown in Listing. 16.

```
Clock c1, c2;
c1 excludes(p=0.95) c2;
```

Listing 16. `Probabilistic exclusion` relation

(d) `Probabilistic precedence` $c1 \prec_{0.95} c2$ specifies that c1 must run faster than c2, as shown in Listing. 17.

```
Clock c1, c2
c1 precedes(p=0.95) c2;
```

Listing 17. `Probabilistic precedence` relation

(e) `Probabilistic causality` $c1 \preceq_{0.95} c2$; represents a relaxed version of precedence relation, allowing the two clocks to tick at the same time. The corresponding encodings is illustrated in Listing. 18.

```
Clock c1, c2;
c1 causes(p=0.95) c2;
```

Listing 18. `Probabilistic causality` relation

Note that the following keywords should not be used as identifier or names when editing PrCcsl encodings/specifications: Clock, is, DenseClockType, idealClk, IdealClock, factor, offset, reference, reset, coincides, excludes, causes, precedes, periodicOn, period, if, then, else, inf, sup, on, clockInter, clockUnion, delayFor, filterBy.

6 A Small Example

Consider the requirement: Two events $c1$ and $c2$ can not happen at the same time. This requirement can be considered as an `exclusion` timing constraint, which can be specified as $c1\#_p c2$. Here, we set the probability threshold p as 95%. The corresponding textural encoding of the above requirement is illustrated in Listing. 19.

```
Clock c1, c2;
c1 excludes(p=0.95) c2;
```

Listing 19. Ccsl/PrCcsl specifications of `R1`

We show the translation process and results under the two different translation modes. To translate the PrCcsl specifications into Uppaal-SMC models and queries, we perform the following steps:
Translation under PrCCSL mode:
1. Configure select the "PrCcsl mode" and set *bound* into 100.
2. Click `Run` button. ProTL requires users to select a directory in which the generated *.xml* file can be stored. The translated Uppaal-SMC model is shown in Fig. 7.
3. Go to *Verifier* to verify the requirement (or open the generated file by Uppaal-SMC). The verification result is presented in Fig. 8
Translation under PrCCSL+System mode

1. Choose *PrCCSL+System* mode (see Fig. 9(a)).

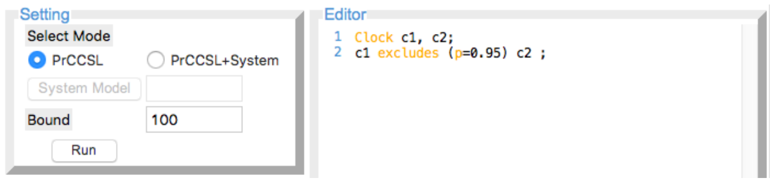


Fig. 5. Translator in ProTL

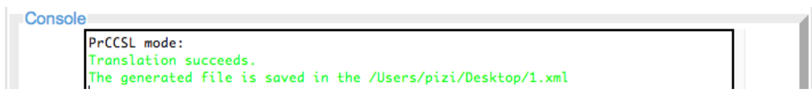


Fig. 6. Translator in ProTL

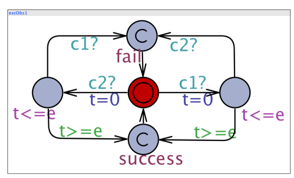
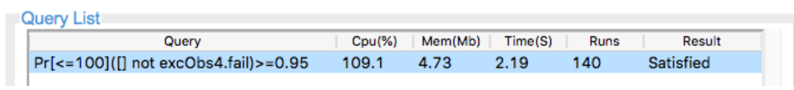
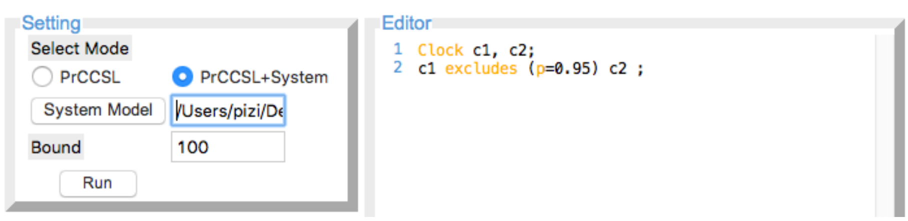


Fig. 7. Translated UPPAAL-SMC model

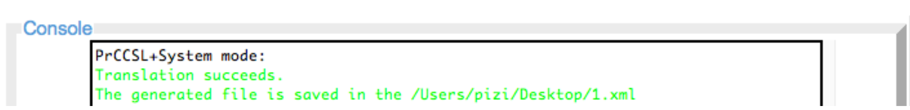


Query	Cpu(%)	Mem(Mb)	Time(S)	Runs	Result
Pr[<=100]([] not excObs4.fail)>=0.95	109.1	4.73	2.19	140	Satisfied

Fig. 8. Translator in ProTL



(a) Editor and Setting



(b) Console Messages

Fig. 9. PrCCSL+System mode

2. Click *System model* button to import the *".xml"* file of system model.
3. Click *Run* button and select a directory in which the generated *".xml"* file can be saved.

References

1. UPPAAL-SMC. http://people.cs.aau.dk/~adavid/smc/ (2013)
2. André, C.: Syntax and semantics of the clock constraint specification language (ccsl). Tech. rep., INRIA (2009)
3. Bernat, G., Burns, A., Llamosi, A.: Weakly hard real-time systems. Transactions on Computers **50**(4), 308 – 321 (2001)
4. David, A., Du, D., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., Sedwards, S.: Statistical model checking for stochastic hybrid systems. In: Hybrid Systems and Biology (HSB). pp. 122 – 136. Hybrid Systems and Biology (HSB), EPTCS (2012)
5. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B.: UPPAAL-SMC tutorial. International Journal on Software Tools for Technology Transfer (STTT) **17**(4), 397 – 415 (2015)
6. Kang, E.Y., Mu, D., Huang, L.: Probabilistic analysis of weakly-hard real-time systems. Tech. rep., School of Data and Computer Science, Sun Yat-Sen University (2018), https://arxiv.org/abs/1807.00003
7. Kang, E.Y., Mu, D., Huang, L.: Probabilistic verification of timing constraints in automotive systems using UPPAAL-SMC. In: International Conference on Integrated Formal Methods (iFM). pp. 236–254. Springer (2018)
8. Mallet, F., Simone, R.D.: Correctness issues on MARTE/CCSL constraints. Science of Computer Programming **106**, 78 – 92 (2015)
9. ProTL. https://sites.google.com/view/protl