ProTL Tutorial

1 Overview of ProTL

ProTL (Probabilistic-CCSL TransLator) [8] is a tool that provides a push-button transformation from PrCCSL (probabilistic extension of clock constraint specification language) [5,6] specifications to UPPAAL-SMC [1] models (stored in *.xml* files). ProTL enables formal verification of constraints/requirements specified as probabilistic temporal and casual *relations* in PrCCSL using UPPAAL-SMC. ProTL offers a textual editor for PrCCSL specifications and allows users to configure parameters for generation of different probabilistic queries, e.g., *Hypothesis Testing*, *Probability Estimation*, *Simulations*. Two translation modes are provided and different output results are generated under different modes:

1. PrCCSL mode: ProTL translates the input PrCCSL encodings/specifications into STA and queries. The resulted model is an *.xml* file containing all the generated STA and queries.
2. PrCCSL+System mode: ProTL generates an *.xml* file by integrating the system model (provided by users) with the translated STA and queries of the input PrCCSL specifications.

2 Preliminary

2.1 Probabilistic Extension of Clock Constraints Specification Language

PrCCSL [5,6] is a probabilistic extension of CCSL [2,7] for formal specification of timing constraints associated with stochastic behaviors. In PrCCSL, a clock represents a sequence of (possibly infinite) instants. An event is a clock and the occurrences of an event correspond to a set of ticks of the clock. A dense clock that represents physical time can be discretized into a discrete/logical clock. *idealClock* is a predefined dense clock whose unit is second. We define a universal clock *ms* based on *idealClock*: *ms* = *idealClock* `discretizedBy` 0.001. *ms* represents a periodic clock that ticks every 1 millisecond in this paper. A step is a tick of the universal clock, i.e., 1 millisecond. An execution of the system model is a `run` where the occurrences of events are clock ticks. The history of a clock c presents the number of times the clock c has ticked prior to the current step.

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

Definition 2 (History). *For $c \in C$, the `history` of c in a run R is a function: H_R^c: $\mathbb{N} \to \mathbb{N}$. $H_R^c(i)$ 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) $H_R^c(0) = 0$; (2) $\forall\ i \in \mathbb{N}^+,\ c \notin R(i) \implies H_R^c(i+1) = H_R^c(i)$ (3) $\forall\ i \in \mathbb{N}^+,\ c \in R(i) \implies H_R^c(i+1) = H_R^c(i)\ +\ 1$.*

CCSL provides two types of clock constraints, *expressions* and *relations*. An *expression* derives new clocks from the already defined clocks. Let C be a set of clocks, $c1, c2 \in C$, `ITE` (if-then-else) *expression*, denoted as β ? $c1$: $c2$, defines a new clock that behaves either as a clock $c1$ or as $c2$ according to the value of the boolean variable β. `Intersection` expression (denoted as $c1$ * $c2$) builds a new clock that ticks whenever both $c1$ and $c2$ tick. `DelayFor` (denoted ref (d) $\rightsquigarrow$ $base$) results in a clock by delaying the *base* clock for d ticks of a reference clock *ref*. `periodicOn` 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*. `Infimum`, denoted `inf`, is defined as the slowest clock that is faster than both $c1$ and $c2$. `Supremum`, denoted `sup`, is defined as the fastest clock that is slower than $c1$ and $c2$. `FilterBy`$(c1\ \alpha(\gamma))$ generates a new clock that ticks from the α^{th} tick of $c1$ and coincides with $c1$ whenever γ is true.

A *relation* in CCSL limits the occurrences among different events/clocks. A *relation* in PrCCSL is satisfied if and only if the probability of *relation* constraint being satisfied is greater than or equal to the probability threshold p. Given k `runs` $= \{R_1, \ldots, R_k\}$, the probabilistic *relations*, i.e., `subclock`, `coincidence`, `exclusion`, `causality` and `precedence` *relations* in PrCCSL are defined as follows:

Definition 3 (Probabilistic Subclock). $c1 \subseteq_p c2 \iff Pr[c1 \subseteq c2] \geqslant p$, *where* $Pr[c1 \subseteq c2] = \frac{1}{k} \sum_{j=1}^{k} \{R_j \models c1 \subseteq c2\}$, *representing 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* i *in* R_j*, s.t.,* $(R_j \models c1 \subseteq c2) \iff (\forall i\ 0 \leqslant i \leqslant n,\ c1 \in R(i) \implies c2 \in R(i))$*. "*$R_j \models c1 \subseteq c2$*" returns 1 if* R_j *satisfies* $c1 \subseteq c2$*, otherwise it returns 0.*

Definition 4 (Probabilistic Coincidence). $c1 \equiv_p c2 \iff Pr[c1 \equiv c2] \geqslant p$, *where* $Pr[c1 \equiv c2] = \frac{1}{k} \sum_{j=1}^{k} \{R_j \models c1 \equiv c2\}$, *which 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.*

Definition 5 (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.*

Definition 6 (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 condition* $\forall i,\ 0 \leqslant i \leqslant n,\ (H_R^{c1}(i) \geqslant H_R^{c2}(i))$ *and* $(H_R^{c2}(i) = H_R^{c1}(i)) \implies (c2 \notin R(i))$ *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.*

Definition 7 (Probabilistic Causality). *The probabilistic* `causality` *relation between* $c1$ *and* $c2$ *(*$c1$ *is the cause and* $c2$ *is the effect), denoted* $c1 \preceq_p c2$*, is satisfied if the following condition holds:*

$$\mathcal{M} \models c1 \preceq_p c2 \iff Pr[c1 \preceq c2] \geqslant p$$

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`*.*

2.2 UPPAAL-SMC

UPPAAL-SMC [1, 3] 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) [4] 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://github.com/`

`kangeu/protl`. The installation can be completed by simply running selecting the installation directory and following the installation instructions.

4 ProTL Interface

Fig. 1 illustrates the interface of ProTL. which consists of five parts:

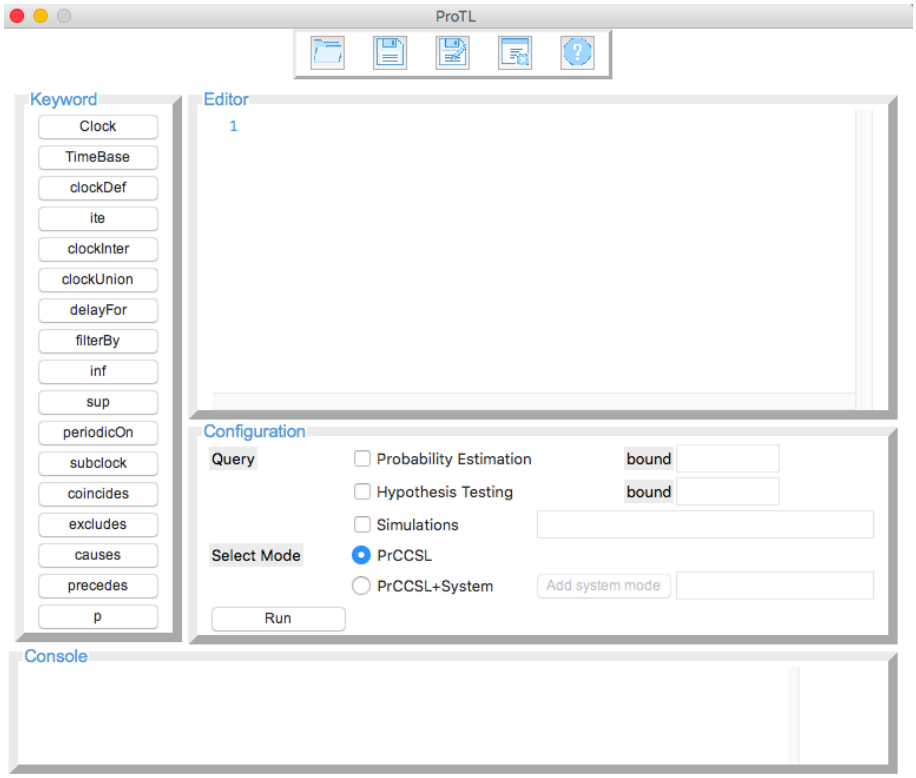


Fig. 1. Interface of ProTL

1. *Tool Bar* (on the top): provides basic functions that enable user to import and save files;
2. *Keyword* (on the left): offers the textural keywords to facilitate users edit the PrCCSL encodings/specifications;
3. *Editor* (on the right): allows users to edit encodings in PrCCSL format;
4. *Configuration* (below the editor): allows users to configure the parameter used for translation;
5. *Console* (on the bottom): shows the errors/warnings information of the input PrCCSL encodings/specifications to help users to refine/fix the encodings.

4.1 Tool bar

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

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

Fig. 2. Buttons in ProTL Tool Bar

1. **Open** button allows users to import the textural file (stored in *.txt* file) that contains PrCCSL specifications/encodings, into *Editor*.
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 *Configuration*.
5. **Help** button directs the user to access the *ProTL Tutorial* [8], in which the detailed information about the usage of the tool can be found.

4.2 Keyword

Keyword provides a set of keywords, i.e., the textural expressions of corresponding PrCCSL operators, which aims to provide the convenience to user for editing PrCCSL encodings/specifications. *Keyword* consists of three parts: clock declarations, PrCCSL relations and expressions:

- **Declaration**: `Clock`, `Timebase`, `clockDef`
- **Expressions**: `periodicOn`, `inf`, `sup`, `delayFor`, `clockInter`, `clockUnion`, `filterBy`, `ite`
- **Relations**: `coincides`, `excludes`, `subclock`, `causes`, `precedes`, `p`

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

4.3 Editor

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

4.4 Configuration

Configuration enables users to specify parameters of queries used in PrCcsl translation, which is illustrated in Fig. 3. *Configuration* consists of three sections: **Query**, **Select mode**, and **Run** button.

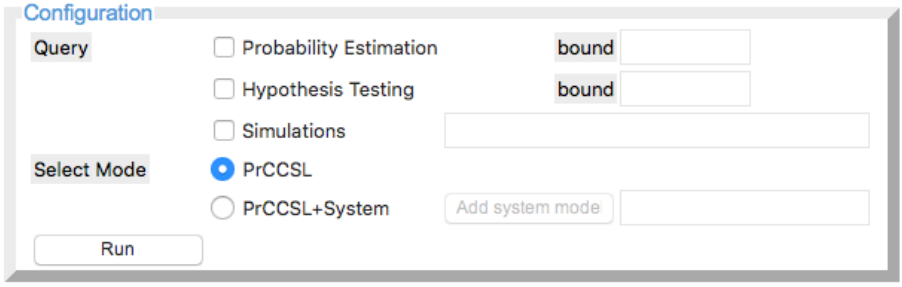


Fig. 3. ProTL Configuration

1. **Query** provides three types of Uppaal-SMC queries, i.e., *Probability Estimation*, *Hypothesis Testing*, and *Simulations* (introduced in Sec. 2.2), with relevant parameters to facilitate users to perform verification and simulation. For generation of *Probability Estimation* and *Hypothesis Testing* query, user needs to give the value for *bound*, which represents the time bound of verification and simulation. The probability threshold of *Hypothesis Testing* is implicitly set as the threshold of corresponding *relations* in PrCcsl. For example, if *bound* is set as 3000 and the probability threshold is 95%, the generated queries corresponding to the checking *property*1 (specified as a propositional formula) is shown in Listing. 1.

```
1  Probability Estimation: Pr[<=3000]([] property1)
2  Hypothesis Testing: Pr[<=3000]([] property1)>=0.95
```

Listing 1. *Probability Estimation* and *Hypothesis Testing* queries

   For generation of *Simulations* query, the input parameters can be specified using a tuple (N, *bound*, *varaibles*): the number of simulations is represented by N, the time bound of simulation is *bound*, and the variable names are listed in *variables*. For instance, if the parameter is set as *(100;3000;v1,v2)* ($v1$ and $v2$ are two variables), the generated simulations query is shown in Listing. 2.

```
1  simulate 100 [<=3000] {v1,v2}
```

Listing 2. *Simulations* query

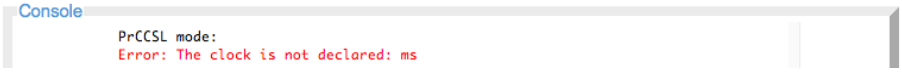
2. **Select Mode** section offers two modes for translation:

(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). The system model need to be loaded prior to the translation by clicking *Add system model* button.

3. **Run** button starts the translation of PrCCSL specifications.

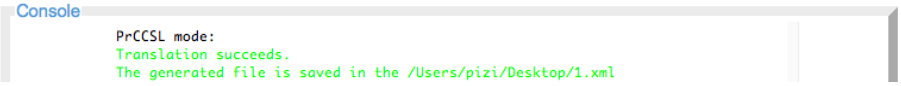
4.5 Console

In ProTL, the error/warning information that regarding to errors of syntax of the encodings or configuration for translation is shown in *console*. Those error information and messages help users refine the encodings. Fig.4 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) Translation status

Fig. 4. Console message

5 Input Language of ProTL

In this section, we presents the syntax of input languages of ProTL, i.e., PrCCSL encodinsg/specifications. We first describe the character set that are supported in the PrCCSL specifications. We then introduce how to 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

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

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. Declaration
   (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 .3.

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

Listing 3. Clock declaration

   (b) "`Timebase`" 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 `TimeBase` is specified with *precisionFactor* and *precisionUnit*. An example of `TimeBase` declaration for a clock *ms* that ticks every 1 milisecond is illustrated in Listing .4.

```
1  TimeBase ms {dimension physicalTime
2  precisionFactor 10 precisionUnit millisecond};
```

Listing 4. `TimeBase` declaration

2. Expression
   (a) "`clockDef`" 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*.

```
1  Clock c;
2  Clock base;
3  c clockDef 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.

```
1  Clock c, c1, c2;
2  c clockDef 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.

```
1  Clock c, c1, c2;
2  c clockDef 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.

```
1  Clock c;
2  Clock base, ref;
3  c clockDef 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.

```
1  Clock c, c1, c2;
2  c clockDef 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.

```
1  Clock c, c1, c2;
2  c clockDef 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.

```
1  Clock c, c1;
2  c clockDef 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.

```
1  Clock c, c1, c2;
2  c clockDef if b then c1 else c2;
```

Listing 12. `ite` expression

3. Relations

(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. 13.

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

Listing 13. `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. 14.

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

Listing 14. `Probabilistic coincides` relation

(c) `Probabilistic exclusion` ($c1 \#_{0.95} c2$) prevents the instants of two clocks from being coincident, as shown in Listing. 15.

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

Listing 15. `Probabilistic exclusion` relation

(d) `Probabilistic precedence` $c1 \prec_{0.95} c2$ specifies that c1 must run faster than c2, as shown in Listing. 16.

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

Listing 16. `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. 17.

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

Listing 17. `Probabilistic causality` relation

Note that the following keywords should not be used as identifier or names when editing PrCCSL encodings/specifications: clock, clockDef, TimeBase, dimension, physicalTime, precisionFactor, precisionUnit, coincides, excludes, causes, precedes, periodicOn, period, if, then, else, inf, sup, on, clockInter, clockUnion, delayFor, filterBy.

6 A Small Example

Consider the following requirement: *Two events* $c1$ *and* $c2$ *can not happen at the same time*. The 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. 18.

```
1  Clock ms;
2  TimeBase ms {dimension physicalTime precisionFactor 1
3  precisionUnit microsecond};
4  Clock c1, c2;
5  c1 excludes(p=0.95) c2;
```

Listing 18. CCSL/PrCCSL specifications of `R1`

Recall that ProTL provides two different translatoion modes. We show the translation process and results under the two different modes. To translate the PrCCSL specifications into STA and queries, we perform the following steps:

– Translation under PrCCSL mode:
  1. Configure parameters in *Query* section. We choose to generate `Probability Estimation` query and set *bound* as 3000. Fig. 5 shows the configuration of mode and parameters under "PrCCSL" mode.
  2. Click `Run` button. If the specifications contain any syntax errors, users can refine the encodings regarding to the information given in the *Console*. If

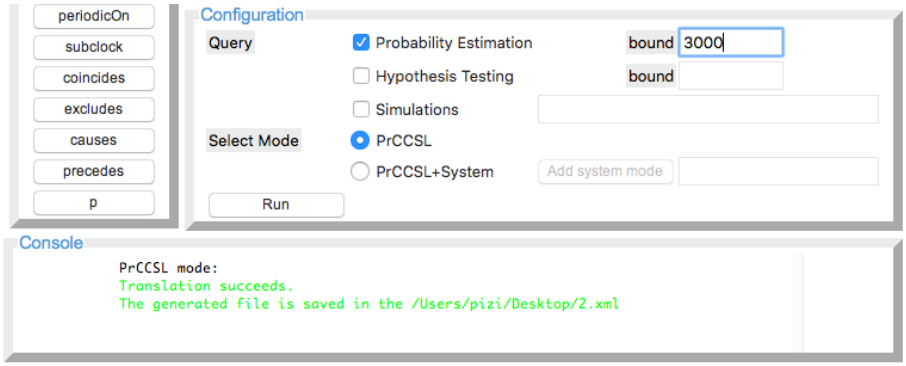


Fig. 5. Configuration of parameters and mode

ProTL succeeds to translate the PrCCSL specifications into STA, it requires users to select a directory in which the generated *.xml* file can be saved.
3. If the above steps are completed successfully, users can open the generated file by UPPAAL-SMC (as presented in Fig. 6). Fig. 6(a) shows the declaration of *broadcast channels* and corresponding variables. Fig. 6(c) shows the five generated STA (processes): The `Universal_ms` process corresponds to `TimeBase` declaration. The ticking of the universal clock *ms* is represented by the *synchronization channel*, i.e., *ms*!. The `Clock3` and `Clock4` processes represent the tick and history of clock $c1$ and $c2$, respectively. `excObs1` process captures the semantics of the `exclusion` *relation*, corresponding to *c1 excludes(p=0.95) c2*. Fig. 6(b) shows the generated `Probability Estimation` query.

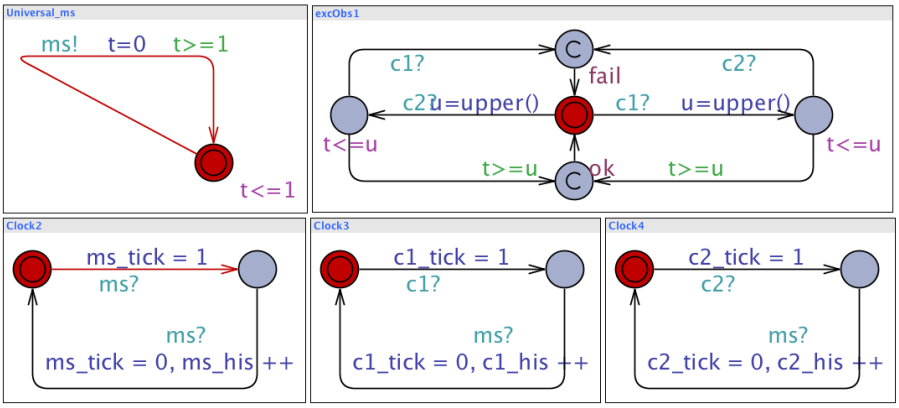
– Translation under PrCCSL+System mode
  1. Choose *PrCCSL+System* mode (see Fig. 7).
  2. Click *Add system model* button to select file of system model.
  3. Click *Run* button. If the specifications contain any syntax errors, users can refine the encodings regarding to the information given in the *Console*. If ProTL succeeds to translate the PrCCSL specifications into STA, it requires users to select a directory in which the generated *.xml* file can be saved.

```
// Place global declarations here.
broadcast chan ms;
int ms_tick;
int ms_his;
broadcast chan c1;
int c1_tick;
int c1_his;
broadcast chan c2;
int c2_tick;
int c2_his;
```

(a) Variable Declaration

```
Pr[<=3000] ([] not excObs1.fail)
```

(b) Probability Estimation Query



(c) STA

Fig. 6. Translated model under PrCCSL mode

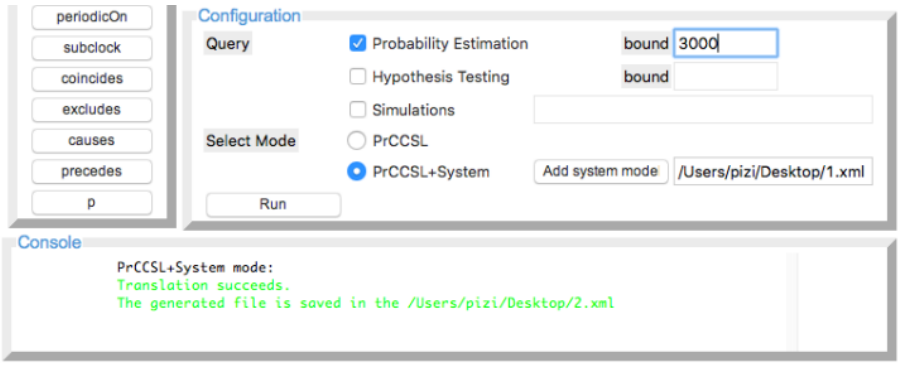


Fig. 7. PrCCSL+System mode

References

1. UPPAAL-SMC. http://people.cs.aau.dk/~adavid/smc/
2. André, C.: Syntax and semantics of the clock constraint specification language (CCSL). Ph.D. thesis, INRIA (2009)
3. 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. EPTCS (2012)
4. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B.: UPPAAL-SMC tutorial. International Journal on Software Tools for Technology Transfer **17**(4), 397 – 415 (2015)
5. Kang, E.Y., Mu, D., Huang, L.: Probabilistic analysis of weakly-hard real-time systems. arXiv preprint arXiv:1807.00003 (2018)
6. 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)
7. Mallet, F., De Simone, R.: Correctness issues on MARTE/CCSL constraints. Science of Computer Programming **106**, 78 – 92 (2015)
8. ProTL. https://github.com/kangeu/protl