Cosmos User Manual

Benoît Barbot

February 8, 2017

This is the user manual for the tool Cosmos version 1.4

1 Tool Usage

1.1 Basic Usage

The primary usage of the tool Cosmos is as follows:

```
Cosmos model property
```

Where the model is specified in the GRML file format or in the `.gspn` file format and the property is expressed either in GRML or in the `.lha` file format. Various options change the behaviours of the simulator and are listed below.

Cosmos is shipped with several examples in the `Examples` directory.

Cosmos can also be called thought the CosyVerif platform.

1.2 Architecture of the Tool

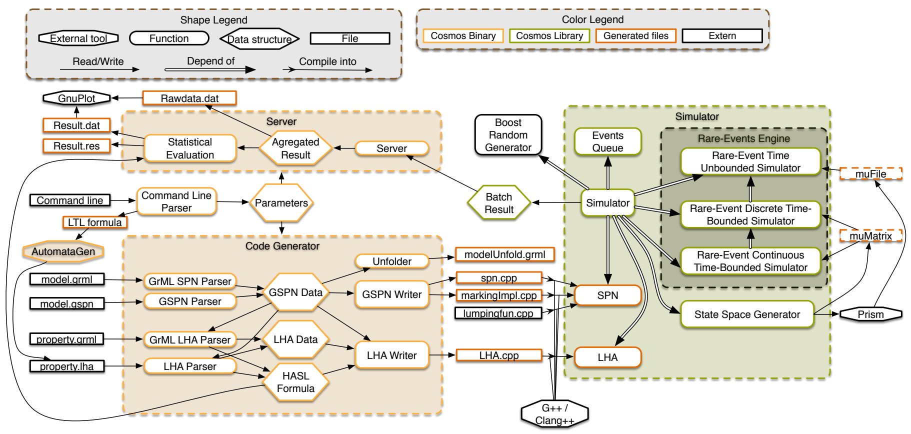

Figure 1: Overview of the architecture of Cosmos

Cosmos is based on the generation of C++ code to produce an efficient simulator from the input model and specification.

From the Petri net given as input the tool produces functions for firing transitions, checking whether a transition is enabled and computing parameters of probability distributions. Tables are also generated that contains for any transitions *t* the set of transitions that might become enabled or disabled due to the firing of *t*. This table is computed by an analysis of the structure of the Petri net and increase the speed of the simulation by testing only small subset of transitions after each firing.

HASL formulas are also transformed into the code. More specifically transitions of the automaton that synchronise with the Petri net use tables to compute the synchronisation. Autonomous transitions of the automaton are implemented by precomputing a system of linear equations and generating function to finish the computation at run time this is possible due to constraints on the expression of the automaton. This also compute exact computation of time for the firing of transitions. Tools dealing with more general hybrid automaton must solve this problem with numerical integration which introduces numerical errors. Computations of integral over the trajectory of the system are also resolved exactly and efficiently thanks to linear constraints.

The generated code is linked with a library containing other parts of the simulator that is not dependent on the model or specification. This library contains: an event heaps to efficiently between the next event in the system, the main synchronisation loop that uses the generated code and a pseudo-random generator to produce new events that are put in the heap.

1.2.1 Model

Models taken by Cosmos as input are variants of Petri nets. The main models supported by Cosmos are stochastic Petri nets with inhibitor arcs and general distributions. The file format of CosyVerif (GRML) allows to specify different variants of Petri nets which are also recognised by Cosmos:

- Plain Petri nets can be read: in this case, an exponential distribution of rate 1 is assigned to each transition.
- Stochastic Petri nets with inhibitor arcs and general distributions.
- Symmetric nets: like for plain Petri nets, exponential distributions of rates 1 are assumed.
- Stochastic Symmetric Nets.

The specification of file formats are reported in Section 2. The `.gspn` file format is the legacy file format of Cosmos. As `.grml` is more standard and easier to extend, it is advised to use `.grml` for new models.

1.2.2 Property

Properties are specified in the Hybrid Linear Automaton Logic (HASL) which, as its name suggests, relies on Linear Hybrid Automaton (LHA) to select paths and on several HASL expressions to specify various performance values.

1.3 Statistical Options

Several options can be used to control the behaviour of the statistical engine. The default behaviour is to simulate trajectories until either

- The maximum number of trajectories is reached.
- The specified width for the confidence interval is reached.

1.3.1 `--width arg` option

This option specifies `arg` as the goal width for the confidence interval. If it is set to 0, trajectories are simulated until the maximum number of trajectories is reached. The default value is 0.001.

1.3.2 `--max-run arg` option

This option specifies `arg` as the maximal number of runs. The default value is $2,000,000$.

1.3.3 `--level arg` option

This option sets the confidence level to `arg`. The default value is 0.99.

1.3.4 `--batch arg` option

This option sets the number of simulations to perform between each statistical test. The default value is 1000. When this value is set to 0, the tool will perform a test at constant time period.

1.3.5 `--njob arg` option

This option sets the number of parallel computations to `arg`. The default value is 1.

1.3.6 `--relative` option

This option allows Cosmos to use a relative confidence interval instead of an absolute one.

1.3.7 `--chernoff arg` option

This option uses Chernoff-Hoeffding bound to compute statistical parameters. `arg` contains one of `width,level,nbrun` which is computed using Chernoff-Hoeffding bound.

1.4 Input Options

1.4.1 `--const X1=c1,X2=c2,...,Xn=cn` option

This option overrides the values of the constants of the model or the property. If the constant was already defined, the value is overwritten, if not, the constant is defined.

1.4.2 `-g, --grml-input` option

This option forces the tool to use the GRML parser for the model and the LHA. This is mostly deprecated as the correct parser is inferred from the extension.

1.4.3 `--HASL-expression arg` option

This allows one to specify additional HASL expressions to the automaton. The syntax of `arg` is the one of HASL expressions in the `.lha` file format and should end with “`;`”.

1.4.4 `--loop arg [--transient arg2]` option

This produces a set of HASL formulas evaluating the mean number of tokens in each place and the mean throughput of each transition until time `arg`. The optional additional argument allows one to specify a transient time where the mean number of tokens and throughput are not recorded. See options `--trace-place` to specify which places and transitions to monitor.

1.4.5 `--sampling t1 t2` option

This produces an automaton which samples, each `t2` time unit, the mean number of tokens in each place for this period, until time `t1` is reached. See options `--output-graph` to format the output in a more manageable way.

1.4.6 `--count-transition` option

This produces a set of HASL formulas counting the number of firing of each transition.

1.4.7 `--formula arg` option

Sends the expression `arg` to the *AutomataGen* tool to produce the LHA used for the simulation. The tool AutomataGen is a work in progress.

1.5 Output formatting Options

The default behaviour of Cosmos for displaying results is as follows: During the simulation a synthetic overview is displayed:

```
Total paths: 28570000  Accepted paths: 22442000  Wall-clock time: 21s  Remaining(approx): 32s  Trajectory per second: 1.4e+06
r:      |< 2.639128431e-08 --[ 0.4999066088    < 0.5000635601    > 0.5002205154    ]-- 0.9999999687 >| width=0.0003139 level=0.99
pi:     |< 0               --[ 3.141245613     < 3.142037101     > 3.142828210     ]-- 4            >| width=0.0015825 level=0.99
pi2:    |< 0               --[ 0.7853114033    < 0.7855092754    > 0.7857070525    ]-- 1            >| width=0.0003956 level=0.99
% of Err: [|||||||||||||||||||||||||||||||||||||||                                                      ] 39%
% of run: [||||||||||||||||||||||||||||                                                                 ] 28%
```

The first line provides information on the speed of simulation. The remaining time is computed differently depending of the statistical procedure used. For fixed-sample-size algorithms, the estimation of the computation time is straightforward. For the Chow-Robbins's method, the size of the confidence interval is assumed to decrease in $1/\sqrt{N}$ with N the number of trajectories, which is quite reliable. For the SPRT, a heuristic is used and the estimation is a lot less reliable. The second to fourth lines present three HASL expressions. The first number is the minimal value observed for this expression, then the lower bound of the confidence interval is displayed, then the estimated value, then the upper bound and then the maximal observed value. The width of the confidence interval is displayed at the end. The last two lines show progress bars indicating the progress of the simulation. The first one indicates the progress to reach the specified confidence interval width limit. This progress is corrected assuming that the width progress in $1/\sqrt{N}$ thus this bar progress at constant speed. The last bar is the number of simulated trajectories compared to the maximal number of trajectories.

When the simulation finishes, results are returned in a key-value format which is easier to parse for tools:

```
Model path: pi.gspn
LHA path: pi.lha
r:
Estimated value: 0.500031454383448
Confidence interval: [0.499932270760877 , 0.50013063800602]
Minimal and maximal value: [1.93365088646926e-08 , 0.999999971987635]
Width: 0.000198367245143327
pi:
Estimated value: 3.14176752504576
Confidence interval: [3.14126745104347 , 3.1422674472686]
Minimal and maximal value: [0 , 4]
Width: 0.000999996225129696
pi2:
Estimated value: 0.78544188126144
Confidence interval: [0.785316862760866 , 0.785566861817149]
Minimal and maximal value: [0 , 1]
Width: 0.000249999056282424
Method: Confidence interval computed sequentially using Chows-Robbin algorithm or SPRT.
Confidence level: 0.99
Total paths: 71569000
Accepted paths: 56213290
Batch size: 1000
Time for simulation: 55.245259s
Total CPU time: 60.068667
Total Memory used: 44.87 MB
Number of jobs: 1
Results are saved in 'Result_pi.res'
Results are saved in 'Result.res'
```

All the simulation parameters are recalled. The additional information is the `Time for simulation` which is the wall clock time used by the simulation. It decreases when the number of parallel processes increases on multiprocessor machine. The `Total CPU time` is the CPU time consumed by the whole computation, it slightly increases when the number of parallel processes increases. The `Total Memory used` line reports the memory used by the whole computation. In most cases, the maximum memory consumption is reached during the compilation of the simulator and highly depends of the compiler used. This output is also stored in a file called `Result.res`.

Several options allow to output additional data from the simulation:

1.5.1 `-v, --verbose arg` option

Sets the verbose level of the tool. The default is 2, with 0 the tool does not write anything on the standard output. The maximum is 6 which generates debug information at each step of the simulation.

1.5.2 `-i,--interactive` option

This option allows one to debug a model by stopping the simulation after each step. This option should not be used together with `--njob n` with $n > 1$. When the simulation is stopped the user is asked either to type `step` to fire the following transition or `fire tr` where `tr` is the name of an enabled transition. This mode is not fully compatible with stochastic symmetric nets as the command `fire` does not allow to specify a binding.

1.5.3 `--output-graph arg` option

Allows one to output results of HASL formula in a blank separated file format. The argument `arg` specifies the name of the file. This is well suited for HASL expressions specifying a graph like `PDF, CDF` or the expression generated by the `--sampling` option. Figure 2 shows a graph generated by the `--sampling 2 0.001` option on the shared memory example. The `--output-graph` option generates a file starting as files:

```
abscissa MeanToken_Access1_low MeanToken_Access1_mean MeanToken_Access1_up          ...
0.005 7.6218e-05 0.000144057 0.000211896 0.00920415 0.00986402 0.0105239 0.000103712 ...
0.015 0.000803674 0.00103768 0.00127168 0.0279092 0.0292027 0.0304962 0.000759281    ...
...
```

Which can easily be plotted as the figure using `gnuplot`. This can also be used for the user defined HASL formulas by using names ending by `$GRAPH$x1$x2$` for HASL expressions where $(x1 + x2)/2$ is the abscissa of the point computed by the expression.

1.5.4 `-d, --output-data arg` option

This option generates a file showing the evolution of the confidence interval along the time. This allows one to observe the speed of convergence. Figure 3 shows a graph plotted from this data.

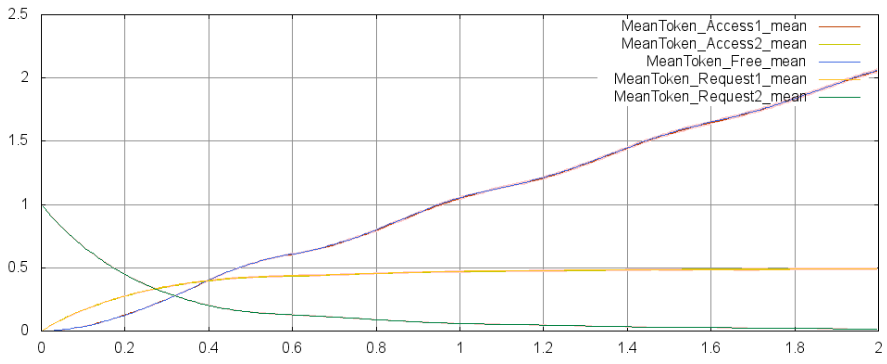

Figure 2: Sampling the trajectories of the shared memory example

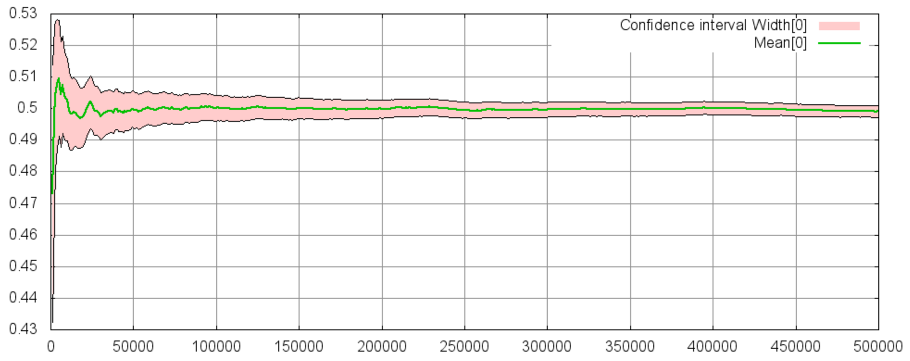

Figure 3: Convergence of the estimation for the shared memory example

1.5.5 `--output-raw arg` option

This option allows one to output the final value of the HASL expressions at the end of each trajectory. This option was introduced for debug purposes and should not be used in normal use. This option is not compatible with `--njob n` when $n > 1$

1.5.6 `--output-trace arg arg2` option

This option stores the traces of the trajectories in file `arg`. Each line contains the global state of the simulator (Petri net + LHA) after a transition. Each trajectory is separated by a line containing `New Path`. Figure 4 contains a trace of one trajectory of the shared memory example. The second argument `arg2` specifies a minimum sampling time. If `arg2` > 0, no consecutive lines in the output file are separated by less than `arg2` time units. This is useful when the trace becomes too big to reduce the size of the generated file.

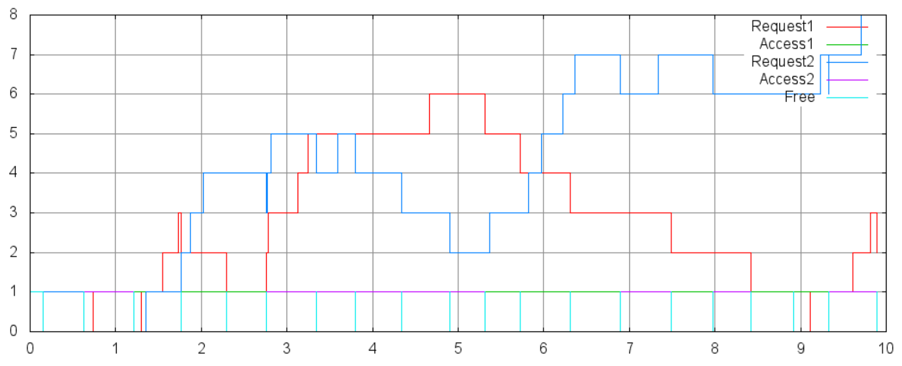

Figure 4: Trace of one simulation for the shared memory example

1.5.7 `--trace-pt arg` option

Specifies which place and transition should be traced in the various output files. `arg` is a list of places and transition names. There are two keywords, `ALL` which is the default, and `COLOR`. The first one allows one to trace all places and transitions, the second, allows one to trace all colour tokens and all bindings of transition for stochastic symmetric nets.

1.5.8 `--gnuplot-driver` option

This option makes Cosmos call `gnuplot` interactively and displays the graph of all outputs specified by a `--output-*` option. All the graphs of this manual have been created using this option.

1.5.9 `--alligator-mode` option

This option removes the output of the progress of the simulation and outputs the progress as a line containing key-value. This mode is useful for interfacing Cosmos with other tools in particular with `alligator`, the server part of CosyVerif. This mode also makes the `--gnuplot-driver` option to write the graph in files instead than on the screen.

1.5.10 `--update-time arg` option

This option sets the time between two updates of the progress of the simulation to be no less than `arg`.

1.6 Miscellaneous Options

1.6.1 `--version` option

Displays the current version of Cosmos.

1.6.2 `-h, --help` option

Displays a help message containing the list of options and quits.

1.6.3 `--seed arg` option

Sets the seed of the simulator to `arg`.

1.6.4 `--local-test` option

This option changes the way enabled transitions are computed. This can improve performance for nets containing transitions with a lot of input arcs.

1.6.5 `--unfold arg` option

This option outputs the model as a place transition net in the GrML file format. This allows ont to unfold symmetric nets but also to convert `.gspn` file to GrML. This option is a work in progress, some expressions on arcs may not be well exported.

1.6.6 `-s, --state-space` option

Does not run the simulator, instead generates the state space and the transition probability matrix. The output format is the one used by `Prism` to import models.

1.6.7 `--prism` option

Exports the state space then run `Prism` on it. At the end of the computation, the vector of probability to reach the accepting state is read and exported.

1.6.8 `--gppcmd arg` option

Sets the C++ compiler to be `arg`.

1.6.9 `--gppflags arg` option

Appends `arg` to the command line calling the C++ compiler.

1.7 Debugging Options

The following options are implemented only for debugging purposes and should not be used in general.

1.7.1 `--tmp-status arg` option

Changes the way Cosmos handle temporary files. The argument `arg` can take four values:

- 0 is the default. A temporary directory is asked to the system and all temporary files are generated in it. The directory is destroyed at the end of the computation.
- 2. The directory name `tmp` in the current `PATH` is used, it is not destroyed at the end. This option is useful to examine generated code.
- 1. Cosmos assumes that the directory `tmp` exists and contains a fully build version of the model and property. The simulator is launched without being rebuilt. At the end of computation, the directory is destroyed.
- 3. Same as 1 but does not destroy the `tmp` directory at the end.

These options allow one to build only once a model and reuse it for several simulations. Only statistical parameters can be changed between simulations as the model is not rebuilding. Use with caution!

1.7.2 `--tmp-path arg` option

Specifies where to put the temporary directory. This option is useless if the `--tmp-status arg` with `arg`> 0 is not specified.

1.7.3 `--bin-path arg` option

Overrides the path to the installation of Cosmos. This path is used by Cosmos to find headers and libraries required to build the model.

1.7.4 `--debug-string` option

Adds debugging information to the generated code. This option is activated and required by a verbose mode greater than 4.

1.8 Rare Events Options

1.8.1 `-r` option

Use importance sampling during the simulation. This option estimates time-unbounded teachability probability. This requires a file name `lumpingfun.cpp` to be in the current directory containing a function mapping marking of the net to marking of the reduce model. A file name `muFile` containing the vector of probability in a reduce model is also required. This file can be produced with the `--prism` option.

1.8.2 `-b arg1 --set-horizon arg2` option

Use time-bounded importance sampling assuming a discrete semantic of the model. `arg1` specify which algorithms to use between 1,2 and 3. 1 is the fastest while 3 consume the least memory. 2 is a trade-off. `arg2` specify the integer time horizon. This option requires the file `lumpingfun.cpp` and `matrixFile` to be in the same directory. This last file can be produced with the option `--state-space` on the reduce model.

1.8.3 `-c -b arg1 --set-horizon arg2` option

This option use time-bounded importance sampling assuming a continuous semantics for the model. In this context `arg2` is a positive real number.

1.8.4 `--epsilon` option

When using continuous importance sampling this option specify the precision of numerical computations.

1.8.5 `--step-continuous` option

When using continuous importance sampling this option specify the width of steps.

2 File Format

2.1 GrML file format

GrML for Graph Markup Language is the graph base language to define models in the CosyVerifplatform. This language is based on XML.

2.2 Generalised Stochastic Petri Net (.gspn)

This file format is used to describe GSPN. First we describe an example:
This GSPN is described by the following text:

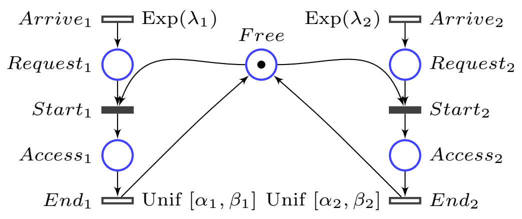

Figure 5: Infinite-state GSPN model of a shared memory system.

```
const double lambda1 = 1;
const double lambda2 = 2;
const double alpha1 = 1;
const double alpha2 = 1;
const double beta1 = 5;
const double beta2 = 5;

NbPlaces = 5;
NbTransitions = 6;

PlacesList = {
   Request_1, Request_2,
   Access_1, Access_2,
   Free
} ;

TransitionsList = {
   Arrive_1,Arrive_2,
   Start_1 ,Start_2,
   End_1   ,End_2
} ;

Marking={
   (Request_1 , 0); (Request_2 , 0) ;
   (Access_1 , 0) ; (Access_2 , 0) ;
   (Free, 1);
};

Transitions={
   (Arrive_1,EXPONENTIAL(lambda1),1,1, SINGLE);
   (Arrive_2,EXPONENTIAL(lambda2),1,1, SINGLE);
   (Start_1,DETERMINISTIC(0),1,1);
   (Start_2,DETERMINISTIC(0),1,1);
   (End_1,UNIFORM(alpha1,beta1),1,1);
   (End_2,UNIFORM(alpha2,beta2),1,1);
};

InArcs={
   (Request_1,Start_1,1); (Free,Start_1,1);
   (Request_2,Start_2,1); (Free,Start_2,1);
   (Access_1,End_1,1);
   (Access_2,End_2,1);
};

OutArcs={
   (Arrive_1,Request_1,1);
   (Arrive_2,Request_2,1);
   (End_1,Free,1);
   (End_2,Free,1);
};
```

Description: The first block is a list of constant definitions, constants can be either `double` or `int`.
Then the number of places and transitions is specified with: `NbPlaces = 5; NbTransitions = 6;`
The list of place names and transition names is given in the `PlacesList` and `TransitionsList` statement.
The initial marking of the net is given as a set of pairs in the `Marking` statement.
The transition distribution is given as a set of tuples like this one:
`(Arrive_1,EXPONENTIAL(lambda1),1,1, SINGLE)` Each tuple contains first the name of the transition then the probability distribution with some parameters, then two positive reals defining the priority and weight of the event generated. For an exponential distribution, the policy of service can be specified by

`SINGLE,INFINITE,MULTIPLE(n)`.
Finally comes the description of the arcs of the net with the `InArcs`,`OutArcs` and `InhibitorsArcs` statements.

2.2.1 Grammar

The complete grammar is:

⟨*accept*⟩ ::= ⟨*GSPN*⟩ 'end of file'

⟨*GSPN*⟩ ::= ⟨*declarations*⟩ ⟨*definitions*⟩

⟨*declarations*⟩ ::= ⟨*Constants*⟩ ⟨*Sizes*⟩ ⟨*Lists*⟩ | ⟨*Sizes*⟩ ⟨*Lists*⟩

⟨*Constants*⟩ ::= ⟨*Constant*⟩ | ⟨*Constant*⟩ ⟨*Constants*⟩

⟨*Constant*⟩ ::= 'const' 'int' ⟨*str*⟩ '=' IntStringFormula ';' | 'const' 'double' ⟨*str*⟩ '=' RealStringFormula ';'

⟨*IntStringFormula*⟩ ::= ⟨*ival*⟩ | ⟨*str*⟩
  | '(' ⟨*IntStringFormula*⟩ ')'
  | ⟨*IntStringFormula*⟩ '+' ⟨*IntStringFormula*⟩
  | ⟨*IntStringFormula*⟩ '-' ⟨*IntStringFormula*⟩
  | ⟨*IntStringFormula*⟩ '*' ⟨*IntStringFormula*⟩
  | ⟨*IntStringFormula*⟩ '^' ⟨*IntStringFormula*⟩
  | FLOOR '(' ⟨*IntStringFormula*⟩ ')'
  | FLOOR '(' ⟨*IntStringFormula*⟩ '/' ⟨*IntStringFormula*⟩ ')'
  | MIN '(' ⟨*IntStringFormula*⟩ ',' ⟨*IntStringFormula*⟩ ')'
  | MAX '(' ⟨*IntStringFormula*⟩ ',' ⟨*IntStringFormula*⟩ ')'

⟨*RealStringFormula*⟩ ::= ⟨*rval*⟩ | ⟨*ival*⟩ | ⟨*str*⟩
  | '(' ⟨*RealStringFormula*⟩ ')'
  | ⟨*RealStringFormula*⟩ '/' ⟨*RealStringFormula*⟩
  | ⟨*RealStringFormula*⟩ '+' ⟨*RealStringFormula*⟩
  | ⟨*RealStringFormula*⟩ '-' ⟨*RealStringFormula*⟩
  | ⟨*RealStringFormula*⟩ '*' ⟨*RealStringFormula*⟩
  | ⟨*RealStringFormula*⟩ '^' ⟨*RealStringFormula*⟩
  | FLOOR '(' ⟨*RealStringFormula*⟩ ')'
  | MIN '(' ⟨*RealStringFormula*⟩ ',' ⟨*RealStringFormula*⟩ ')'
  | MAX '(' ⟨*RealStringFormula*⟩ ',' ⟨*RealStringFormula*⟩ ')'

⟨*Sizes*⟩ ::= ⟨*NbPlaces*⟩ ⟨*NbTransitions*⟩ | ⟨*NbTransitions*⟩ ⟨*NbPlaces*⟩

⟨*NbPlaces*⟩ ::= 'NbPlaces' '=' ⟨*ival*⟩ ';' | 'NbPlaces' '=' ⟨*str*⟩ ';'

⟨*NbTransitions*⟩ ::= 'NbTransitions' '=' ⟨*ival*⟩ ';' | 'NbTransitions' '=' ⟨*str*⟩ ';'

⟨*Lists*⟩ ::= ⟨*PlacesList*⟩ ⟨*TransitionsList*⟩ | ⟨*TransitionsList*⟩ ⟨*PlacesList*⟩

⟨*PlacesList*⟩ ::= 'PlacesList' '=' '' ⟨*PLabels*⟩ '' ';'

⟨*PLabels*⟩ ::= ⟨*str*⟩ | ⟨*PLabels*⟩ ',' ⟨*str*⟩

⟨*TransitionsList*⟩ ::= 'TransitionList' '=' '' ⟨*TLabels*⟩ '' ';'

⟨*TLabels*⟩ ::= ⟨*str*⟩ | ⟨*TLabels*⟩ ',' ⟨*str*⟩

⟨*definitions*⟩ ::= ⟨*PlacesDef*⟩ ⟨*TransitionsDef*⟩ ⟨*InArcs*⟩ ⟨*OutArcs*⟩
  | ⟨*PlacesDef*⟩ ⟨*TransitionsDef*⟩ ⟨*InArcs*⟩ ⟨*OutArcs*⟩ ⟨*Inhibitors*⟩

⟨*PlacesDef*⟩ ::= ’Marking’ ’=’ ” ⟨*PLACES*⟩ ” ’;’

⟨*PLACES*⟩ ::= ⟨*PLACE*⟩ | ⟨*PLACES*⟩ ⟨*PLACE*⟩

⟨*PLACE*⟩ ::= ’(’ ⟨*str*⟩ ’,’ ⟨*IntStringFormula*⟩ ’)’ ’;’

⟨*TransitionsDef*⟩ ::= ’Transition’ ’=’ ” ⟨*TRANSITIONS*⟩ ” ’;’

⟨*Transitions*⟩ ::= ⟨*Transition*⟩ | ⟨*Transitions*⟩ ⟨*Transition*⟩

⟨*Transition*⟩ ::= ’(’ ⟨*str*⟩ ’,’ ⟨*dist*⟩ ’,’ ⟨*Priority*⟩ ’,’ ⟨*Weight*⟩ ’)’ ’;’
| ’(’ ⟨*str*⟩ ’,’ ’EXPONENTIAL’ ’(’ ⟨*RealStringFormula*⟩ ’)’ ’,’ ⟨*Priority*⟩ ’,’ ⟨*Weight*⟩ ’,’ ⟨*Service*⟩ ’)’ ’;’
| ’(’ ⟨*str*⟩ ’,’ ’IMDT’ ’,’ ⟨*Priority*⟩ ’,’ ⟨*Weight*⟩ ’)’ ’;’

⟨*dist*⟩ ::= ⟨*str*⟩ ’(’ ⟨*params*⟩ ’)’

⟨*params*⟩ ::= ⟨*RealStringFormula*⟩ | ⟨*params*⟩ ’,’ ⟨*RealStringFormula*⟩

⟨*Weight*⟩ ::= ⟨*RealStringFormula*⟩

⟨*Priority*⟩ ::= ⟨*RealStringFormula*⟩

⟨*Service*⟩ ::= ’SINGLE’ | ’INFINITE’ | ’MULTIPLE’ ’(’ ⟨*ival*⟩ ’)’ | ’MULTIPLE’ ’(’ ⟨*str*⟩ ’)’

⟨*InArcs*⟩ ::= ’InArcs’ ’=’ ” ⟨*incells*⟩ ” ’;’

⟨*incells*⟩ ::= ⟨*incell*⟩ | ⟨*incells*⟩ ⟨*incell*⟩

⟨*incell*⟩ ::= ’(’ ⟨*str*⟩ ’,’ ⟨*str*⟩ ’,’ ⟨*IntStringFormula*⟩ ’)’ ’;’ | ’(’ ⟨*str*⟩ ’,’ ⟨*str*⟩ ’)’ ’;’

⟨*OutArcs*⟩ ::= ’OutArcs’ ’=’ ” ⟨*outcells*⟩ ” ’;’

⟨*outcells*⟩ ::= ⟨*outcell*⟩ | ⟨*outcells*⟩ ⟨*outcell*⟩

⟨*outcell*⟩ ::= ’(’ ⟨*str*⟩ ’,’ ⟨*str*⟩ ’,’ ⟨*IntStringFormula*⟩ ’)’ ’;’ | ’(’ ⟨*str*⟩ ’,’ ⟨*str*⟩ ’)’ ’;’

⟨*Inhibitors*⟩ ::= ’inhibitor’ ’=’ ” ⟨*inhibcells*⟩ ” ’;’

⟨*inhibcells*⟩ ::= ⟨*inhibcell*⟩ | ⟨*inhibcells*⟩ ⟨*inhibcell*⟩

⟨*inhibcell*⟩ ::= ’(’ ⟨*str*⟩ ’,’ ⟨*str*⟩ ’,’ ⟨*IntStringFormula*⟩ ’)’ ’;’ | ’(’ ⟨*str*⟩ ’,’ ⟨*str*⟩ ’)’ ’;’

2.3 Linar Hybrid Automaton(.lha)

2.3.1 General Expression

⟨*MarkingDepExpr*⟩ ::= ⟨*ConstIdentifier*⟩ | ⟨*DiscreteVarIdentifier*⟩ | ⟨*PlaceIdentifier*⟩
| ⟨*MarkingDepExpr*⟩ ‘`+`’ ⟨*MarkingDepExpr*⟩ | ⟨*MarkingDepExpr*⟩ ‘`-`’ ⟨*MarkingDepExpr*⟩
| ⟨*MarkingDepExpr*⟩ ‘`*`’ ⟨*MarkingDepExpr*⟩
| ‘`Card`’ ⟨*MarkingDepExpr*⟩
| ⟨*MarkingDepExpr*⟩ ‘`[`’ ⟨*ColorList*⟩ ‘`]`’
| ⟨*DomainMarking*⟩

⟨*Expr*⟩ ::= ⟨*ConstIdentifier*⟩ | ⟨*DiscreteVarIdentifier*⟩ | ⟨*VarIdentifier*⟩ | ⟨*PlaceIdentifier*⟩
| ⟨*Expr*⟩ ‘`+`’ ⟨*Expr*⟩ | ⟨*Expr*⟩ ‘`-`’ ⟨*Expr*⟩
| ⟨*Expr*⟩ ‘`*`’ ⟨*Expr*⟩
| ‘`Card`’ ⟨*Expr*⟩
| ⟨*Expr*⟩ ‘`[`’ ⟨*ColorList*⟩ ‘`]`’
| ⟨*DomainMarking*⟩

⟨*ColorList*⟩ ::= ⟨*ColorExpr*⟩ ‘`,`’ ⟨*ColorList*⟩
  | ⟨*ColorExpr*⟩

⟨*ColorExpr*⟩ ::= ⟨*ColorIdentifier*⟩ | ⟨*ColorVarIdentifier*⟩
  | ⟨*ColorExpr*⟩ ‘`++`’ | ⟨*ColorExpr*⟩ ‘`--`’

⟨*LinearExpr*⟩ ::= ⟨*LinearExpr*⟩ ‘`+`’ ⟨*LinearExpr*⟩ | ⟨*LinearExpr*⟩ ‘`-`’ ⟨*LinearExpr*⟩
  | ⟨*MarkingDepExpr*⟩ ‘`*`’ ⟨*VarExpr*⟩

⟨*VarExpr*⟩ ::= ⟨*VarIdentifier*⟩ | ⟨*VarIdentifier*⟩ ‘`[`’ ⟨*ColorList*⟩ ‘`]`’

⟨*DiscreteVarExpr*⟩ ::= ⟨*DiscreteVarExpr*⟩
  | ⟨*DiscreteVarExpr*⟩ ‘`[`’ ⟨*ColorList*⟩ ‘`]`’

⟨*MarkingDepProp*⟩ ::= ‘`true`’ | ‘`false`’ | ‘`not`’ ⟨*MarkingDepProp*⟩
  | ⟨*MarkingDepProp*⟩ ‘`&&`’ ⟨*MarkingDepProp*⟩ | ⟨*MarkingDepProp*⟩ ‘`||`’ ⟨*MarkingDepProp*⟩
  | ⟨*MarkingDepExpr*⟩ ‘$\bowtie$’ ⟨*MarkingDepExpr*⟩
  | ⟨*ColorExpr*⟩ ‘`=`’ ⟨*ColorExpr*⟩ | ⟨*ColorExpr*⟩ ‘`!=`’ ⟨*ColorExpr*⟩

All the expressions are typed with a domain. An expression of a domain D is a function that map to each colour of the domain D an integer or real expression.

Values of the domain “Uncoloured” act as scalar. The construction “Card” takes an expression over a domain and return an expression of the uncoloured domain by taking the sum of the expression over all colors. The construction “Expr[c1,c2]” returns a value in the uncoloured domain equal to the valuation of “Expr” for the colour “c1,c2”.

There are two types of expression. The one described by “Expr” can depend on any quantities defined in the Net or in the automaton whereas “MarkingDepExpr” cannot depend on quantities which are continuous variables. The difference is that “MarkingDepExpr” is constant between two transitions of the automaton whereas “Expr” evolves linearly between two transitions. Both of them can be discontinuous when a transition of the automaton occurs.

2.3.2 Declarative Part and General Expression

⟨*Declaration*⟩ ::= ⟨*VarDeclaration*⟩ ‘`;`’ | ⟨*ConstDeclaration*⟩ ‘`;`’

⟨*ConstDeclaration*⟩ ::= ‘`const`’ ‘`int`’ ⟨*IDENTIFIER*⟩ ‘`=`’ ⟨*INTEGER*⟩
  | ‘`const`’ ‘`double`’ ⟨*IDENTIFIER*⟩ ‘`=`’ ⟨*FLOAT*⟩

⟨*VarDeclaration*⟩ ::= ‘`var`’ ⟨*IDENTIFIER*⟩ (‘`in`’ ⟨*DomainIdentifier*⟩) ?
  | ‘`discvar`’ ⟨*IDENTIFIER*⟩ (‘`in`’ ⟨*DomainIdentifier*⟩) ?
  | ‘`colorvar`’ ⟨*IDENTIFIER*⟩ ‘`in`’ ⟨*DomainIdentifier*⟩

This section defines the identifiers for the different types of variables as well as their domain.

2.3.3 Location Invariant Proposition

⟨*LocInvProp*⟩ ::= ⟨*MarkingDepProp*⟩

Location invariants do not depend on continuous variable value. It ensures that location invariants are constant between two transitions of the automaton.

2.3.4 Flow Expression

⟨*FlowDef*⟩ ::= ⟨*VarExpr*⟩ ‘’’ ‘=’ ⟨*MarkingDepExpr*⟩

As location invariants, flow rates are constant between two transitions of the automaton to ensure that continuous variables are piecewise linear.

2.3.5 Guard Proposition

⟨*GuardProp*⟩ ::= ‘true’ | ⟨*GuardProp*⟩ ‘&&’ ⟨*GuardProp*⟩
  | ⟨*LinearExp*⟩ ‘=’ ⟨*MarkingDepExpr*⟩
  | ⟨*LinExp*⟩ ‘<=’ ⟨*MarkingDepExpr*⟩
  | ⟨*LinearExp*⟩ ‘>=’ ⟨*MarkingDepExpr*⟩
  | ⟨*MarkingDepProp*⟩

⟨*ColorExpr*⟩ ::= ... | ⟨*BindingVarIdentifier*⟩

2.3.6 Update Expression

⟨*UpdateExpr*⟩ ::= ⟨*VarExpr*⟩ ‘:=’ ⟨*Expr*⟩ | ⟨*DiscreteVarExpr*⟩ ‘:=’ ⟨*Expr*⟩
  | ⟨*ColorVarIdentifier*⟩ ‘:=’ ⟨*ColorExpr*⟩

⟨*ColorExpr*⟩ ::= ... | ⟨*BindingVarIdentifier*⟩