# User manual

http://www-dssz.informatik.tu-cottbus.de/DSSZ/
Fuzzy Petri nets and coloured fuzzy Petri nets in Snoopy
January 31, 2021

**George Assaf, Monika Heiner and Fei Liu**
Snoopy@informatik.tu-cottbus.de

http://www-dssz.informatik.tu-cottbus.de
Data Structures and Software Dependability
Computer Science Department
Brandenburg University of Technology Cottbus-Senftenberg

# Contents

# List of Figures

# List of Tables

This document explains the procedure of modelling and simulating $\mathcal{FPN}$ [1] and $\mathcal{FPN}^{\mathcal{C}}$ in *Snoopy*; please compare Figure 1 . Please note that the same steps for one net class can be equally applied to the other classes, just differentiate between uncoloured Petri nets ($\mathcal{PN}$) and coloured Petri nets ($\mathcal{PN}^{\mathcal{C}}$) [3]. Furthermore, we give more details about Latin Hybercube Sampling strategies used by Snoopy's $\mathcal{FPN}$.

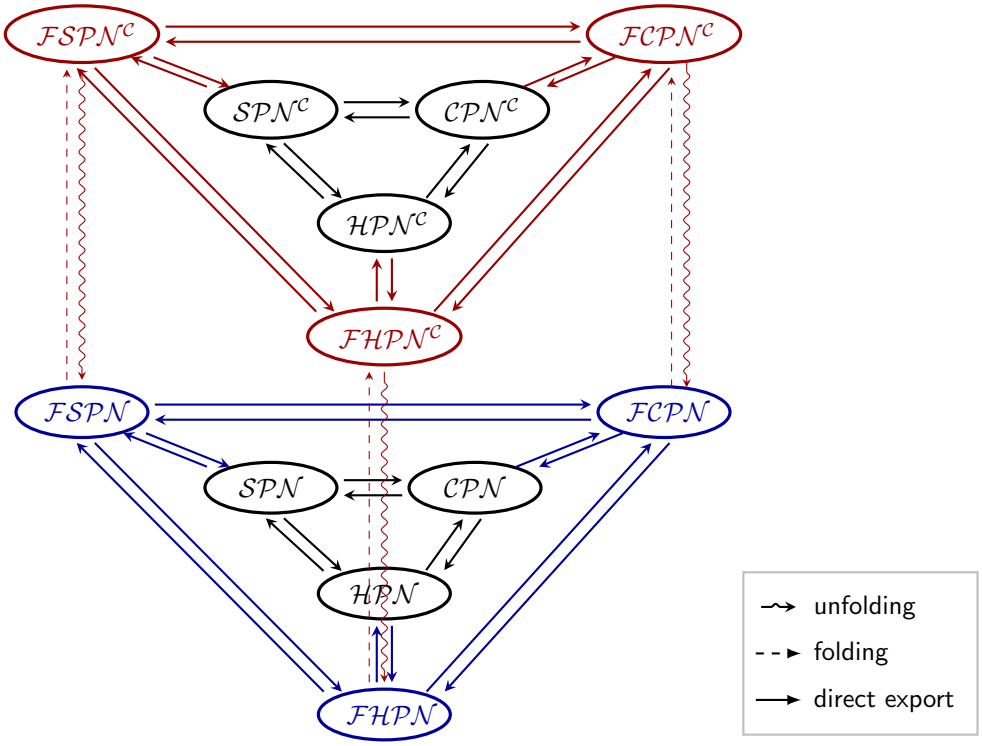



Figure 1: Export relation between some of *Snoopy*'s Petri net classes. Fuzzy nets differ from their crisp counterparts by additional pre-defined data types, supporting fuzzy numbers, which can be used as kinetic parameters. The extensions presented in [1] are coloured in blue, while the latest addition of net classes supported by Snoopy and their export relation are coloured in red. Please note that for clarity there are three folding/unfolding relations not shown in the figure ($\mathcal{SPN}$– $\mathcal{SPN}^{\mathcal{C}}$, $\mathcal{CPN}$– $\mathcal{CPN}^{\mathcal{C}}$, $\mathcal{HPN}$– $\mathcal{HPN}^{\mathcal{C}}$).

# 1 Fuzzy Petri nets (uncoloured )

There are three uncoloured $\mathcal{FPN}$ net classes, comprising $\mathcal{FSPN}$, $\mathcal{FCPN}$ and $\mathcal{FHPN}$. The modelling procedure of these classes can equally be applied. Please note that each net class has its own modelling elements; these elements are listed in the elements tree of that class.

## 1.1 Modelling

Creating an $\mathcal{FPN}$ model starts off with creating a new (empty) net file by choosing file menu and then clicking **new** command; the list of Snoopy net classes will appear as a result; compare Figure 3. The second way is to use the **Export** feature; compare Figure 2.

Figure 4 shows the fuzzy continuous Petri net model Decay dimerzation model which is adopted from [4] in which the transition r3 has k3 as a fuzzy kinetic parameter in its rate function; the Table 1 shows the rate functions associated with each transition.

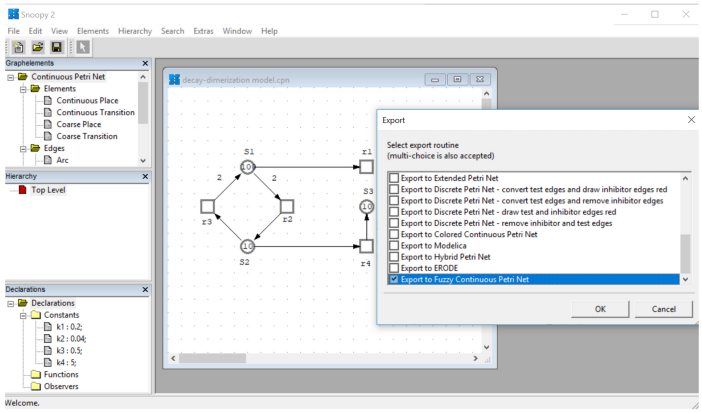



Figure 2: Creating an $\mathcal{FCPN}$ net by exporting a Continuous Petri Networks.

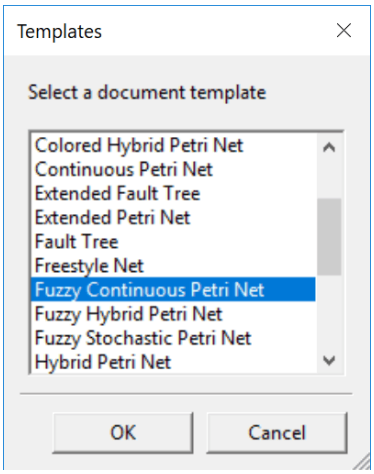



Figure 3: Creating a new (empty) $\mathcal{FCPN}$ model by selecting the appropriate template.

Table 1: Decay dimerisation $\mathcal{FCPN}$- rate functions of transitions, all following mass/action kinetics.

| Transition $r_i$ | Rate function | Kinetic constant $k_i$ |
|---|---|---|
| r1 | $k_1 \cdot S1$ | 0.2 |
| r2 | $k_2 \cdot S1 \cdot S1$ | 0.04 |
| r3 | $k_3 \cdot S2$ | (0.45,0.5,0.55) |
| r4 | $k_4 \cdot S2$ | (4.9,5.0,5.4) |

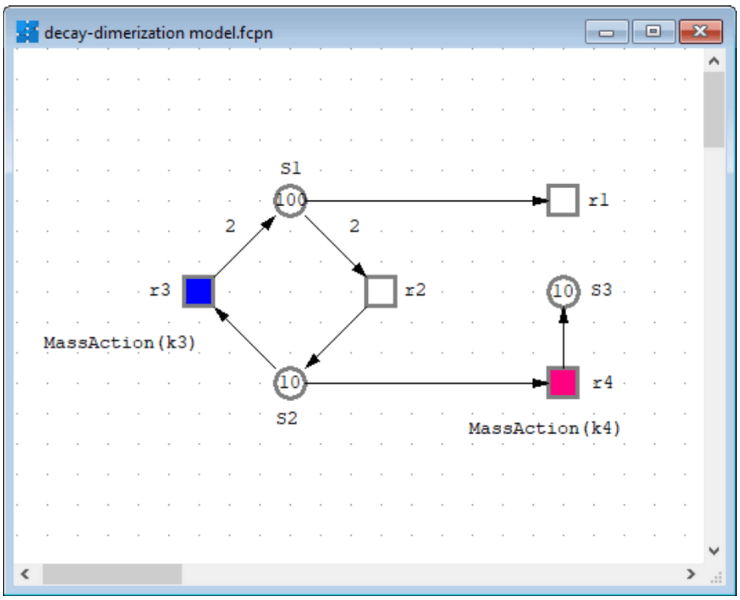



Figure 4: decay dimerisation model ($\mathcal{FCPN}$). The fuzzy kinetic parameters $k3$ and $k4$ are used in the highlighted transitions $r3$ and $r4$, respectively. It consists of a degradation reaction $r1$, two reversible dimerization reactions ($r2$ and $r3$) and a conversion reaction $r4$; this model is adopted from [5]

## 1.2 Constant definitions

Constant Definitions window allows users to create constants by specifying constants name, the group to witch the constant it belongs, the type and the corresponding value. $\mathcal{TFN}$ data type can be specified for defining a constant as a triangular fuzzy number; compare Figure 5.

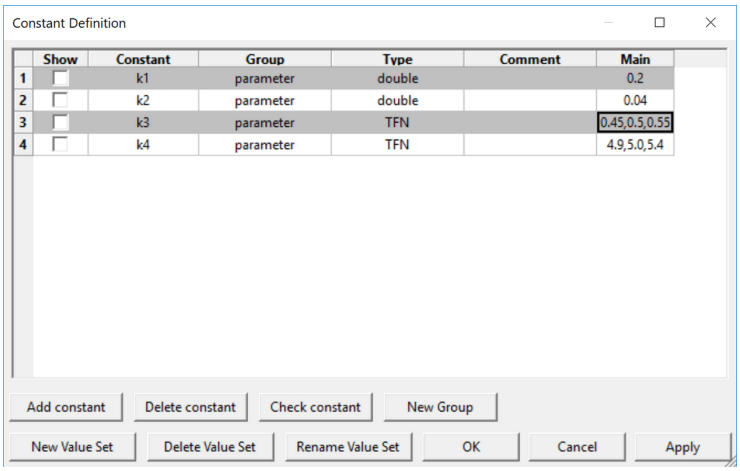



Figure 5: Constant definitions window.

## 1.3 Fuzzy Petri nets simulation

Simulation of $\mathcal{FPN}$ models starts off with switching to simulation mode (choose view Start simulation-mode from *Snoopy* 's view menu). Once simulation configuration window appears, a user can configure the experiment as usual. For $\mathcal{FPN}$ simulation settings, a user can specify number of alpha level, number of sampling points and the sampling strategy; compare FIgure 6. Then a user can start the simulation by clicking on start simulation button. Once simulation finishes, fuzzy band and membership functions of the selected variables can be viewed using viewer window; compare Figure 7.

## 1.4 Fuzzy Continuous PN Test Cases

Besides to decay dimerzation, we give another $\mathcal{FCPN}$ test case; It is called Heat Shock Response [10]. It is an ancient, evolutionary conserved regulatory mechanism that allows the cell to quickly react to elevated temperatures and other forms of physiological and environmental stress. The transition $r7$ has a fuzzy kinetic parameter. Figure 8 gives the $\mathcal{FCPN}$ model in Snoopy and the value of the fuzzy kinetic parameter. Figure 9 gives the fuzzy configuration and simulation results (fuzzy band and timed-membership function) of the variable *hsp* (heat shock protein)

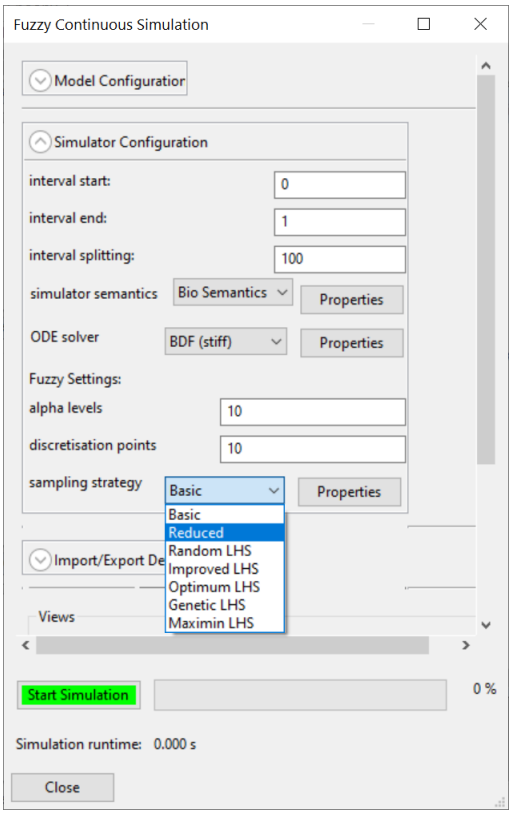
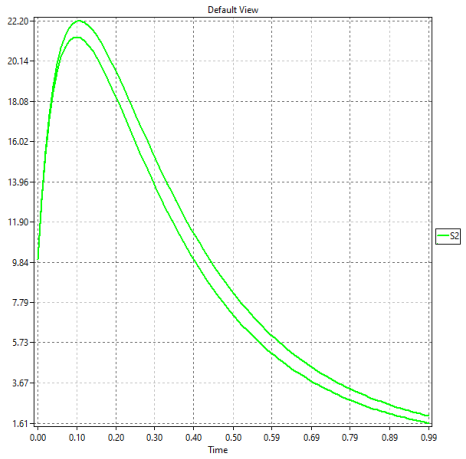
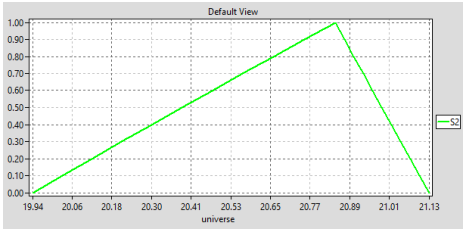



Figure 6: $\mathcal{FCPN}$ simulation configuration window. The simulation dialog consists of the same settings as the standard CPN simulation dialog.

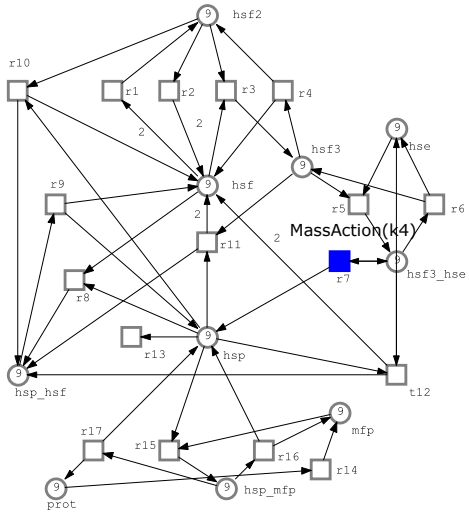



(a) Fuzzy band of the variable $S2$.

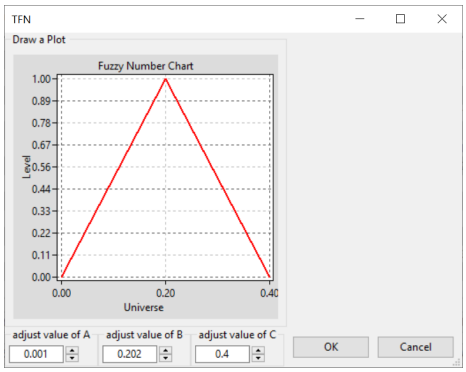



(b) Membership function of the variable $S2$ at $t = 16$.

Figure 7: Fuzzy band and membership function of the variable $S2$ at the time point$t = 16$. **(a)** Fuzzy band . **(b)** Membership function.

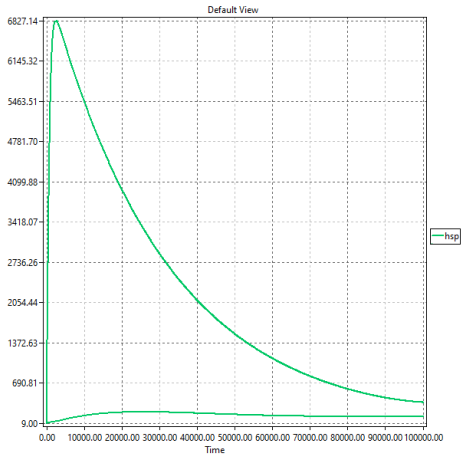



(a) The $\mathcal{FCPN}$ model for eukaryotic heat shock response model which is adopted from [10]

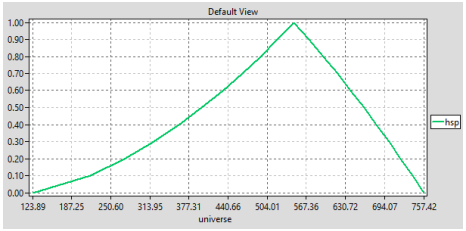



(b) TFN drawing sub-window of the constant definitions window, showing the fuzzy kinetic parameter $k4$.

Figure 8: The $\mathcal{FCPN}$ model for eukaryotic heat shock response model which is adopted from [10]. The fuzzy kinetic parameter is set in the transition $r7$ (marked with blue). (a) $\mathcal{FCPN}$ model. (b) $k4$ defined as TFN .




(a) Fuzzy band of the variable *hsp*




(b) Membership function of the variable *hsp* at the time point $t = 718$.

Figure 9: Fuzzy band and membership function of the variable *hsp* at $t = 718$. (a)Fuzzy band. (b) Membership function of the variable *hsp* at $t = 718$ .

## 1.5 Fuzzy Stochastic PN Test Cases

Here we present two fuzzy stochastic Petri net models. The first model is called Yeast Polarization describing the pheromone-induced G-protein cycle in Saccharomyces cerevisiae; see Figure 10. It is kinetic rate parameters are given in the Table 2. Figure 11 gives the simulation results.

Table 2: Yeast polarization $\mathcal{FSPN}$- rate functions of transitions, all following mass/action kinetics.

| Transition $r_i$ | Rate function | Kinetic constant $k_i$ |
|---|---|---|
| r1 | $k_1 \cdot \phi$ | 0.38 |
| r2 | $k_2 \cdot R$ | 0.04 |
| r3 | $k_3 \cdot (L + R)$ | 0.082 |
| r4 | $k_4 \cdot RL$ | 0.12 |
| r5 | $k_5 \cdot (RL + G)$ | 0.12 |
| r6 | $k_6 \cdot G_a$ | (0.08,0.1,0.12) |
| r7 | $k_7 \cdot (G_d + G_b g)$ | 0.005 |
| r8 | $k_8 \cdot G_a$ | (10,13.21,15) |

The second model is called Virus Infection [8]; which describes the infection of healthy cells by a virus. Cells grow or die. The virus may enter a healthy cell (UCell) and infect it (ICell). Then the virus starts the replication of itself and more viruses are released (note the arc weight of 10). Besides, infected cells may die and viruses may degrade. Figure 12 gives the $\mathcal{FSPN}$ model in Snoopy of the Virus Infection model; Table 3 gives the crisp/fuzzy kinetic parameters used by the experiment. Figure 13 gives the simulation results in terms of fuzzy bands and timed-membership functions.

Table 3: Virus Infection $\mathcal{FSPN}$- rate functions of transitions, all following mass/action kinetics.

| Transition $r$ | Rate function | Kinetic constant $k$ |
|---|---|---|
| $Cell\_growth$ | $k_1$ | 1.0 |
| $UCell\_death$ | $k_2 \cdot Uninfected\_cells$ | 0.04 |
| $Infection$ | $k_3 \cdot Uinfected\_cells \cdot Virus$ | (0.01,0.5,1.0) |
| $Virus\_release$ | $k_4 \cdot Infected\_cells$ | 1.0 |
| $ICell\_death$ | $k_5 \cdot Infected\_cells$ | 0.5 |
| $Degradation$ | $k_6 \cdot Virus$ | 0.1 |

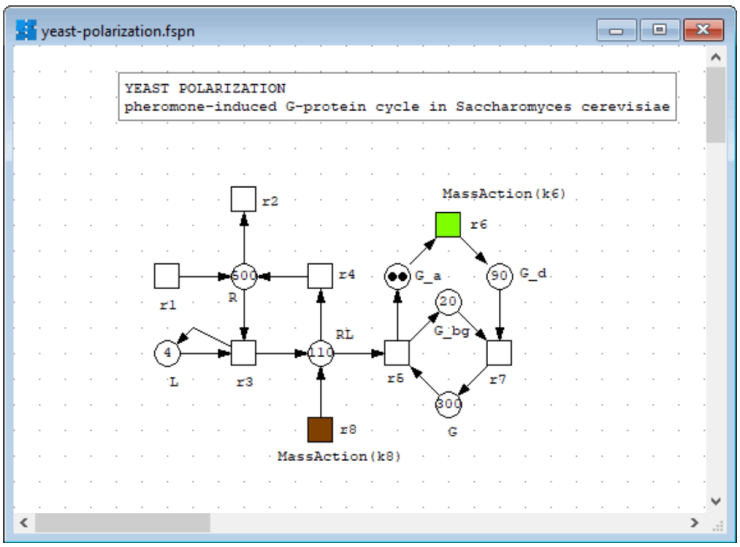
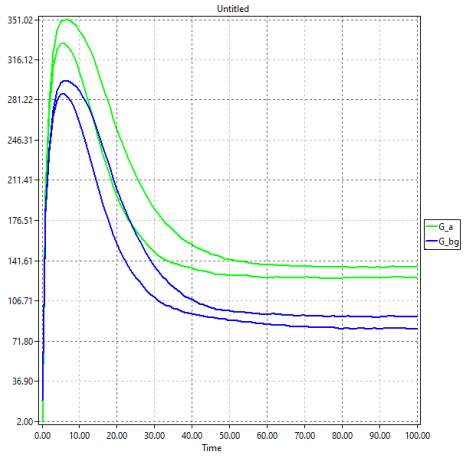
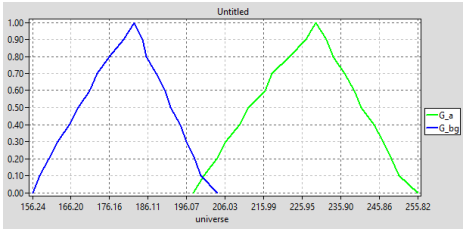



Figure 10: The $\mathcal{FSPN}$ model for yeast polarisation which is adopted from [8]. The fuzzy kinetic parameters $k6 = (0.08, 0.1, 0.12)$ and $k8 = (10, 13.21, 15)$ are used in the highlighted transitions $r6$ and $r8$, respectively.




(a) Fuzzy bands of the variables $G\_a$ and $G\_bg$.




(b) Membership function of the variables $G\_a$ and $G\_bg$ at the time point $t = 20$.

Figure 11: Fuzzy bands and membership functions of the variables $G\_a$ and $G\_bg$ at $t = 20$. Number of stochastic runs is 200. The sampling algorithm is Random LHS. (a)Fuzzy bands. (b) Membership functions.

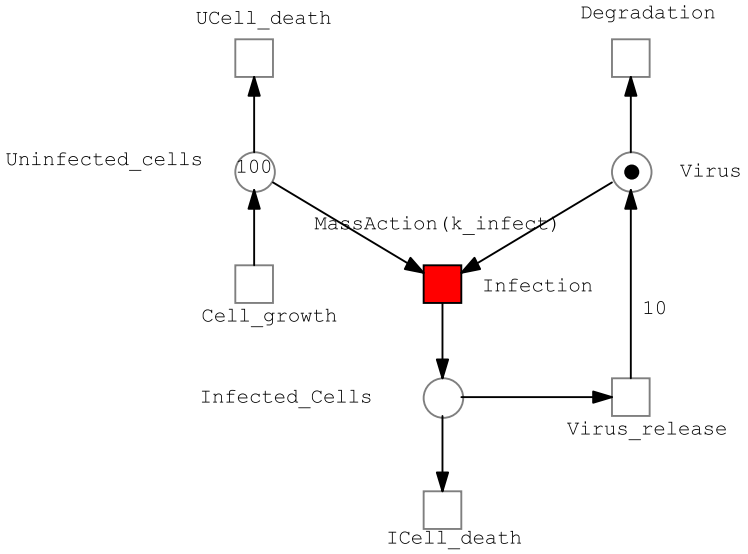
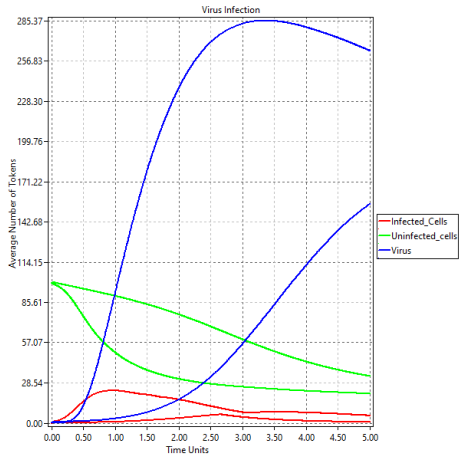
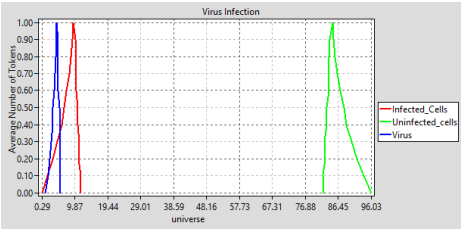



Figure 12: The $\mathcal{FSPN}$ model for virus infection which is adopted from [8]. The fuzzy kinetic parameter is set in the transition $Infection$ (marked with red)




(a) Fuzzy bands of the variables.




(b) Membership functions of the variables at $t = 82$.

Figure 13: Fuzzy bands and membership functions of the variables at the time point$t = 82$. **(a)** Fuzzy band . **(b)** Membership function.

## 1.6 Fuzzy Hybrid PN Test Case

We changed the Yeast Polarisation model to be in a hybrid fashion. First we exported the $\mathcal{FSPN}$ model into $\mathcal{FHPN}$; then we converted the transitions $r1$, $r2$ and $r6$ to the continuous type; we also converted the places $R$, $G\_a$ and $G\_d$ to the continuous type. All other transition/places remain as such. Please note that we also kept values of kinetic parameter as such. Figure 14 gives the Snoopys' $\mathcal{FHPN}$ model; while Figure 15 gives the simulation results.

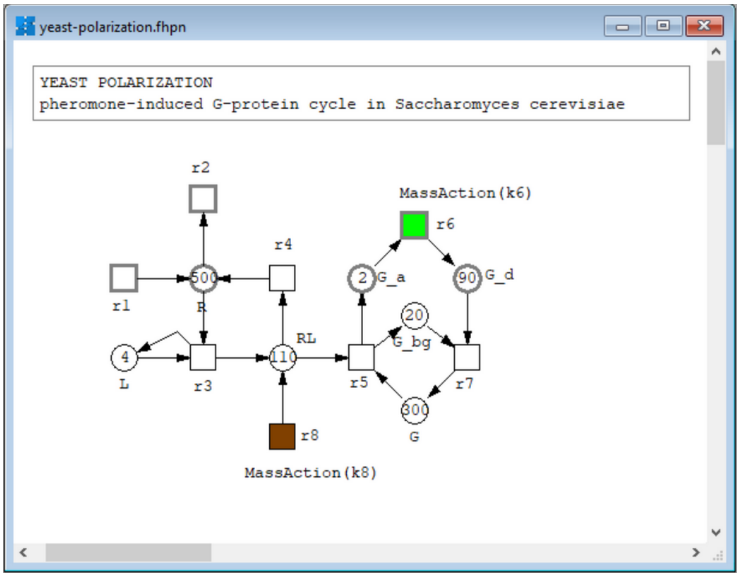
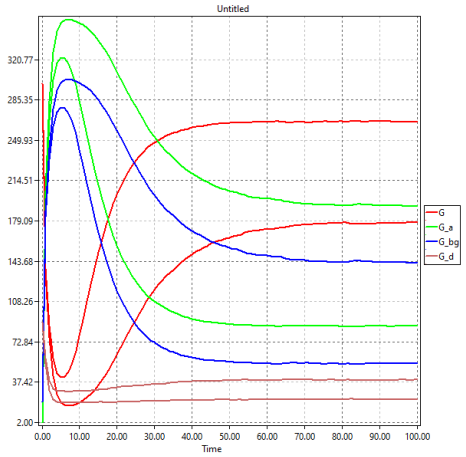
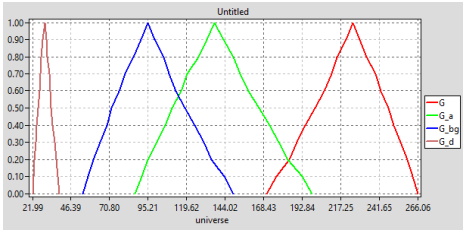



Figure 14: The $\mathcal{FHPN}$ model of yeast polarisation, obtained by exporting the $\mathcal{FSPN}$ model shown in Figure 10 to an $\mathcal{FHPN}$ model and converting a few nodes to continuous ones (drawn with thick grey line style).




(a) Fuzzy bands of the variables.




(b) Membership functions of the selected variables at $t = 57$.

Figure 15: Fuzzy bands and membership functions of the variables at the time point$t = 57$. Nimber of stochastic runs is 200 and the time synchronization algorithm is (Static exact). **(a)** Fuzzy bands. **(b)**Timed membership functions.

# 2 Coloured fuzzy Petri nets

Here we provide a collection of *Snoopy* screenshots outlining the workflow of modelling and simulating $\mathcal{FPN}^{\mathcal{C}}$, comprising the $\mathcal{FCPN}^{\mathcal{C}}$, $\mathcal{FSPN}^{\mathcal{C}}$ and $\mathcal{FHPN}^{\mathcal{C}}$. We demonstrate the modelling and simulating procedures using Repressilator test case as coloured fuzzy stochastic Petri net, the coloured Fuzzy continuous Petri net and coloured Fuzzy hybrid Petri net can be applied in an equivalent way.

## 2.1 Modelling

The modelling procedure starts off with creating a new coloured Fuzzy Petri net file, a net class can be chosen from the list of *Snoopy*'s Petri nets family; compare Figure 16.

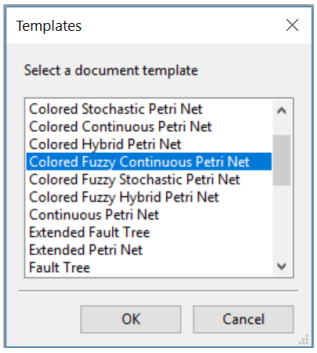



Figure 16: List of Petri net classes in *Snoopy*.

Coloured fuzzy Petri nets can also be created by using Export feature e.g., coloured stochastic Petri nets $\mathcal{SPN}^{\mathcal{C}}$ cen be exported to coloured fuzzy stochastic Petri net.; compare Figure 17.

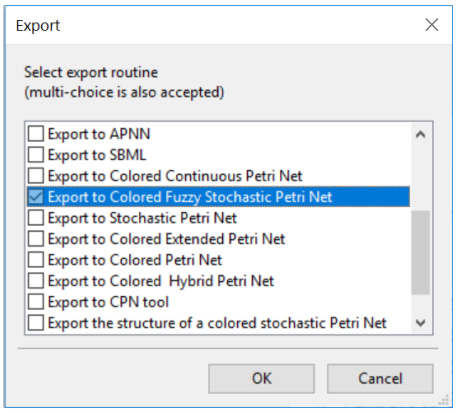



Figure 17: Export dialog of coloured stochastic Petri net.

## 2.2 Constant definitions

Once coloured fuzzy net has been created, a user can define constants using constant definitions window; by double clicking on the constants item on the declarations tree. A user can define constants to be used, e.g., in the net as whole or in the rate functions of transitions. For defining a constant as triangular fuzzy number, a user has to choose the $\mathcal{TFN}$ data type, then the value of the constants can be specified by writing the value directly in the value field; compare Figure 5, or by drawing it graphically using $\mathcal{TFN}$ drawing window which appears by double clicking on the value field ; compare Figure 18.

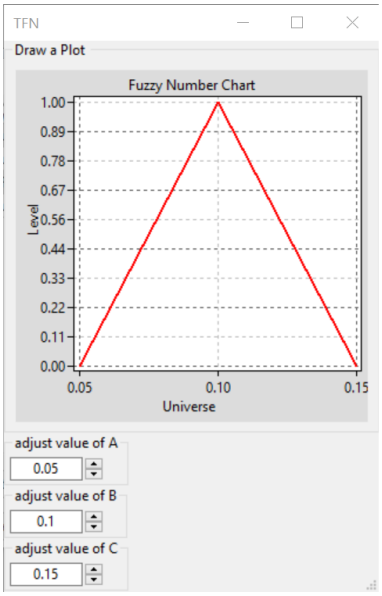



Figure 18: TFN drawing window.

## 2.3 Example - Repressilator

The coloured model of repressilator consists of three places and three transitions, each place is assigned to a colour set **Gene**, the transition **generate** has the following rate function

$$k\_gen * gene$$

, where k_gen is a fuzzy kinetic parameter $\mathcal{TFN}$ (0.05,0.1,0.15) and gene is a variable (pre-place); compare Figure 19. The Table 4 shows the transition rates associated to each transition.

The kinetic parameters can be colour-dependent,e.g., one colour can have a crisp kinetic parameter, and another colour a fuzzy kinetic parameter defined as

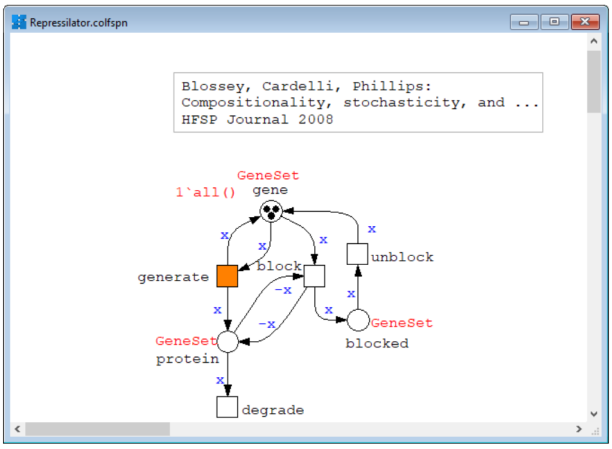



Figure 19: Coloured fuzzy stochastic Petri net model in *Snoopy* which is exported from $\mathcal{SPN}^{\mathcal{C}}$ [7] The declarations: colorset GeneSet = enum with a,b,c, and variable x: GeneSet.

$\mathcal{TFN}$. In the Repressilator model, the coloured transition (in orange) indicates that this transition has a fuzzy kinetic parameter in its rate function, whereas the other transitions have crisp kinetic parameters (white).

## 2.4 Model simulation

Simulation of coloured fuzzy Petri nets requires unfolding step, by switching to the simulation mode, the unfolding engine appears; compare Figure 20.

After unfolding the coloured fuzzy stochastic Petri net, we implicitly get the unfolded fuzzy stochastic Petri net at the background, which is equivalent to the coloured model. As a result, unfolding the Repressilator gives: 9 uncoloured places, 12 stochastic transitions and 30 standard arcs. Furthermore, each transition instance will get assigned a function rate after evaluating the coloured function rate of its corresponding coloured transition, e.g., the function rate of transition **generate** will be evaluated to k_gen*gene_a, k_gen*gene_b and k_gen*gene_c.

Once the simulation result dialog appears, a user can configure the experiment settings e.g., simulation time, the simulator and its properties and the fuzzy-related setting e.g., number of alpha levels, number of discretisation points and sampling strategy, and then a user can start simulation by clicking Start Simulation button; compare Figure 21 for more details.

Table 4: Repressilator $\mathcal{FSPN}^{\mathcal{C}}$- rate functions of transitions.

| Transition $r$ | Rate function | Kinetic constant $k$ |
|---|---|---|
| generate | $k\_gen \cdot gene$ | (0.005,0.1,0.15) |
| degrade | $k\_deg \cdot protein$ | 0.001 |
| blocked | $k\_block \cdot protein$ | 1 |
| unblocked | $k\_unblock \cdot blocked$ | 0.0001 |

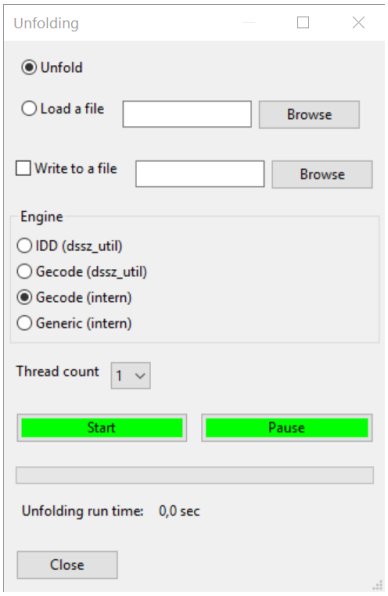
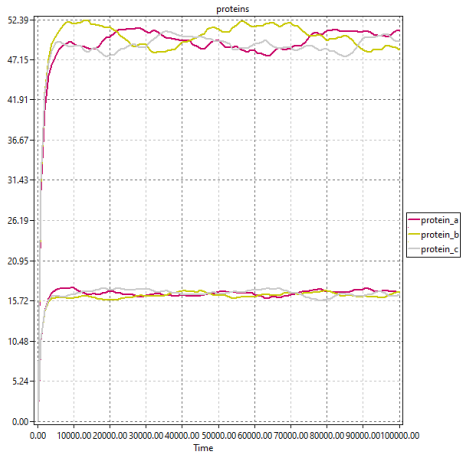
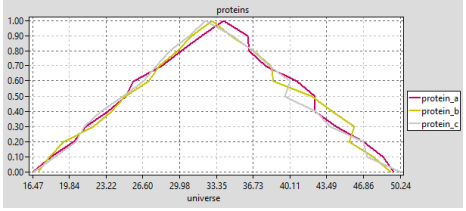



Figure 20: Unfolding engine dialog in *Snoopy*.




(a) Fuzzy bands of the selected variables.




(b) Membership functions of the selected variables at $t = 40$.

Figure 21: Fuzzy bands and membership functions of the variables Protein_a, Protein_b and Protein_c, the membership functions of the variables at the time point 40. Please note that the number of simulation runs of Gillespie's simulator is set up to 10 000 runs **(a)** Fuzzy band . **(b)** Membership function.

## 2.5 $\mathcal{FSPN}^{\mathcal{C}}$ of Coupled $Ca^2+$ Channels

The Coupled $Ca^2+$ Channels is a biological test case describing an ubiquitous second messenger used to regulate a wide range of cellular processes [6]. It basically consists of two-state channel : 'closed' and 'open'; which can be generalized to N channels. The transition diagram between channels can be described as continuous time Markov chain (CTMC), which can be converted into a Petri net model. Figure 22 gives the coloured stochastic fuzzy Petri net of six-state $Ca^2+$ Channels, For more details about the kinetic rates and other basics, please consult [6].

Figure 23 gives the fuzzy bands of some selected (unfolded) variables; while Figure 24 gives the corresponding timed-membership functions at the time point $t = 46$. Please note that membership functions are exactly describe how the fuzzy band is developing over time; we could easily notice that at the middle of the simulation time, when the fuzzy band of the place $NumOpen$ is tending to be narrow; this tendency can obviously be seen from the membership function of the place $NumOpen$ at the same time point as well; see Figure 24. This is equally applied on the other unfolded places; which have narrow membership functions over time.

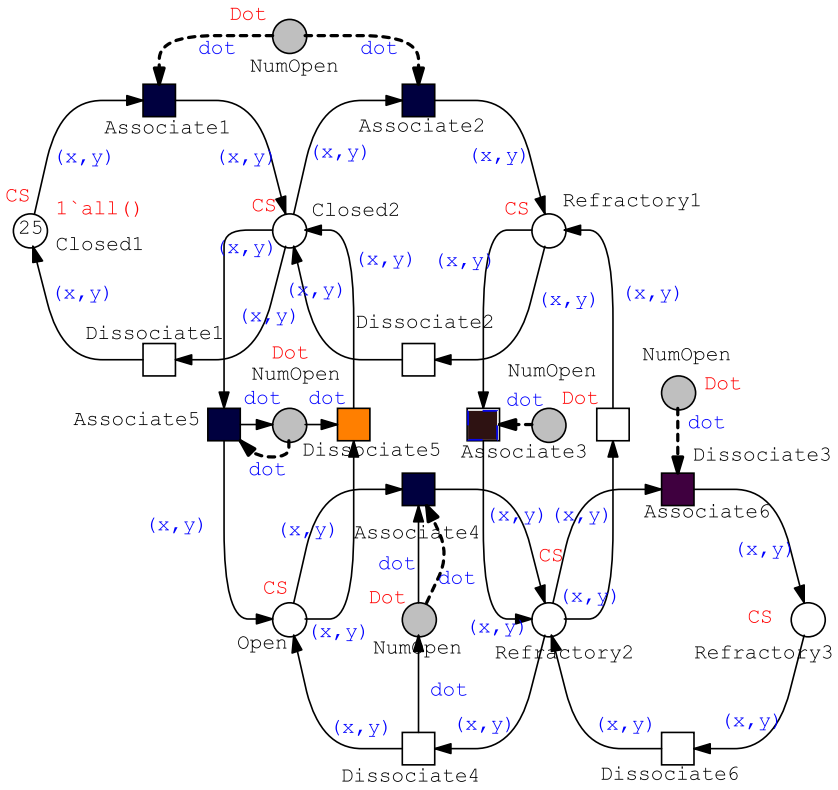



Figure 22: $\mathcal{FSPN}^{\mathcal{C}}$ Six-state $Ca^2+$ in *Snoopy*. Color definitions: Constants: $N = 5$; Color sets: Row = 1 -$N$; Comumn = 1 -$N$; CS = Row X Comumn. Variables: x: Row; y: Comumn. This model has two fuzzy kinetic parameters: $c\_inf$ = (0.01,0.05,0.1) which has been set in the transitions (marked with black); while the second fuzzy kinetic parameter is $k\_em$ = (0.01,6.0,12.01) which has been set in the transition (marked with orange).

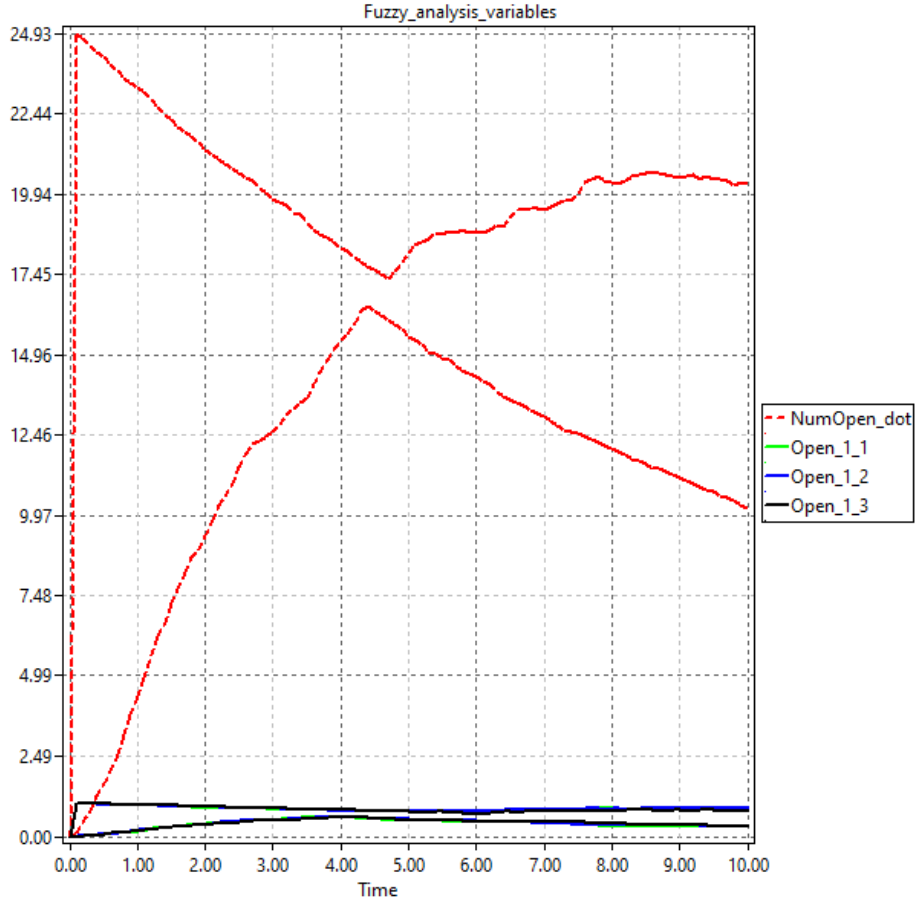



Figure 23: Fuzzy bands of some unfolded variables. Simulation traces have been averaged averaged over 100 runs.

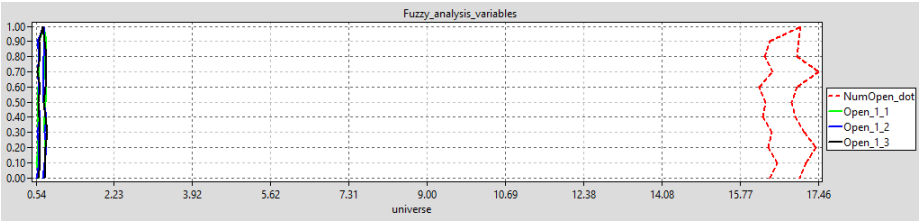



Figure 24: Membership functions of some unfolded variables (at the time point t=46.

# 3 Latin Hypercube Sampling (LHS)

Latin hypercube sampling is a statistical method for generating a near-random sample of parameter values from a multidimensional distribution [9].

The sampling method is often used to construct computer experiments or for Monte Carlo integration. When sampling a function of $N$ variables, the range of each variable is divided into $M$ equally probable intervals. $M$ sample points are then placed to satisfy the Latin hypercube requirements; this forces the number of divisions, $M$, to be equal for each variable. This sampling scheme does not require more samples for more dimensions (variables); this independence is one of the main advantages of this sampling scheme. Another advantage is that random samples can be taken one at a time, remembering which samples were taken so far.

Besides Basic and Reduced Sampling, Snoopy supports five LHS sampling algorithms. The LHS algorithms are implemented in lhslib Library [2]. By using LHS we get the minimal number of simulation runs; no matter how many fuzzy kinetic parameters (Numbers) exist in the model.

## 3.1 Random LHS

**randomLHS(4,3)** returns a 4x3 matrix with each column constructed as follows: A random permutation of (1,2,3,4) is generated, say (3,1,2,4) for each of K columns. Then a uniform random number is picked from each indicated quartile. In this example a random number between .5 and .75 is chosen, then one between 0 and .25, then one between .25 and .5, finally one between .75 and 1.

This algorithm takes three arguments, which are:

- **n** the number of rows or samples,
- **k** the number of columns or parameters/variables,
- **preserveDraw** should the draw be constructed so that it is the same for variable numbers of columns?.

## 3.2 Improved Latin Hypercube Sample

Draws a Latin Hypercube Sample from a set of uniform distributions for use in creating a Latin Hypercube Design. This algorithm attempts to optimize the sample with respect to an optimum euclidean distance between design points. The optimum distance $D$ is calculated using the following equation:

$$D = n/n^{\frac{1,0}{k}}$$

This algorithm takes the following arguments:

- **n** the number of rows or samples
- **k** the number of columns or parameters/variables

- **dup** A factor that determines the number of candidate points used in the search. A multiple of the number of remaining points than can be added.

## 3.3 Optimum Latin Hypercube Sample

Draws a Latin Hypercube Sample from a set of uniform distributions for use in creating a Latin Hypercube Design. This Algorithm uses the Columnwise Pairwise (CP) algorithm to generate an optimal design with respect to the S optimality criterion.

This Algorithm generates a Latin Hypercube Sample by creating random permutations of the first n integers in each of k columns and then transforming those integers into n sections of a standard uniform distribution. Random values are then sampled from within each of the n sections. Once the sample is generated, the uniform sample from a column can be transformed to any distribution by using the quantile functions, e.g. qnorm(). Different columns can have different distributions.

S-optimality seeks to maximize the mean distance from each design point to all the other points in the design, so the points are as spread out as possible.

This Algorithm uses the CP algorithm to generate an optimal design with respect to the S optimality criterion. The Optimum LHS algorithm takes the following arguments as an input:

- **n** the number of rows or samples
- **k** the number of columns or parameters/variables
- **maxSweeps** the maximum number of times the CP algorithm is applied to all the columns.
- **eps** the optimal stopping criterion. Algorithm stops when the change in optimality measure is less than eps*100 % of the previous value
- **verbose** Print informational messages

## 3.4 Latin Hypercube Sampling with a Genetic Algorithm

Draws a Latin Hypercube Sample from a set of uniform distributions for use in creating a Latin Hypercube Design. This algorithm attempts to optimize the sample with respect to the S optimality criterion through a genetic type algorithm.

This algorithm generates a Latin Hypercube Sample by creating random permutations of the first n integers in each of k columns and then transforming those integers into n sections of a standard uniform distribution. Random values are then sampled from within each of the n sections. Once the sample is generated, the uniform sample from a column can be transformed to any distribution by using the quantile functions, e.g. qnorm(). Different columns can have different distributions.

S-optimality seeks to maximize the mean distance from each design point to all the other points in the design, so the points are as spread out as possible. The Genetic Algorithm does the following:

- Generate pop random latin hypercube designs of size n by k
- Calculate the S optimality measure of each design
- Keep the best design in the first position and throw away half of the rest of the population
- Take a random column out of the best matrix and place it in a random column of each of the other matricies, and take a random column out of each of the other matricies and put it in copies of the best matrix thereby causing the progeny
- For each of the progeny, cause a genetic mutation pMut percent of the time. The mutation is accomplished by swtching two elements in a column.

This Algorithm takes the following arguments as an input:

- **n** The number of partitions (simulations or design points or rows)
- **k** The number of replications (variables or columns)
- **pop** The number of designs in the initial population
- **gen** The number of generations over which the algorithm is applied
- **pMut** The probability with which a mutation occurs in a column of the progeny
- **criterium** The optimality criterium of the algorithm. Default is S. Maximin is also supported
- **verbose** Print informational messages. Default is FALSE

## 3.5 Maximin Latin Hypercube Sample

Draws a Latin Hypercube Sample from a set of uniform distributions for use in creating a Latin Hypercube Design. This function attempts to optimize the sample by maximizing the minium distance between design points (maximin criteria).

This Algorithm takes the following arguments as an input:

- **n** The number of partitions (simulations or design points or rows)
- **k** The number of replications (variables or columns)
- **method** build or iterative is the method of LHS creation. build finds the next best point while constructing the LHS. iterative optimizes the resulting sample on $[0, 1]$ or sample grid on $[1, N]$

- **dup** A factor that determines the number of candidate points used in the search. A multiple of the number of remaining points than can be added. This is used when method="build"
- **eps**

  The minimum percent change in the minimum distance used in the iterative method
- **maxIter** The maximum number of iterations to use in the iterative method
- **optimize.on**

  **grid** or **result** gives the basis of the optimization. grid optimizes the LHS on the underlying integer grid. result optimizes the resulting sample on $[0, 1]$
- **debug** prints additional information about the process of the optimization.

# References

[1] G Assaf, M Heiner, and F Liu. Biochemical reaction networks with fuzzy kinetic parameters in Snoopy. In L Bortolussi and G Sanguinetti, editors, *Proc. CMSB 2019*, volume 11773 of *LNCS/LNBI*, pages 302–307. Springer, September 2019. First Online: 17 September 2019.

[2] Latin, Hypercube Sample library. https://github.com/bertcarnell/lhslib".

[3] F. Liu. *Colored Petri Nets for Systems Biology*. PhD thesis, BTU Cottbus, Computer Science Institute, January 2012.

[4] F Liu, S Chen, M Heiner, and H Song. Modeling Biological Systems with Uncertain Kinetic Data Using Fuzzy Continuous Petri Nets. *BMC Systems Biology*, 12(Suppl 4):42, 2018. accepted for publication: January 2018.

[5] F. Liu, S. Chen, M. Heiner, and H. Song. Modelling biological systems with uncertain kinetic data using continuous Petri nets. *BMC Systems Biology*, 12:64–74, 2018.

[6] F Liu and M Heiner. Multiscale modelling of coupled ca2+ channels using coloured stochastic petri nets. *IET Systems Biology*, 7(4):106 – 113, August 2013.

[7] F Liu and M Heiner. *Petri Nets for Modeling and Analyzing Biochemical Reaction Networks*, chapter 9, pages 245–272. Springer, 2014.

[8] F. Liu, M. Heiner, and M. Yang. Fuzzy stochastic Petri nets for modelling biological systems with uncertain kinetic parameters. *PLoS ONE*, pages 1–19, 2016.

[9] M. D. Mckay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21:239–245, 1979.

[10] I Petre, A Mizera, C Hyder, A Meinander, A Mikhailov, R Morimoto, L Sistonen, J Eriksson, and R Back. A simple mass-action model for the eukaryotic heat shock response and its mathematical validation.