Ers:
A Tool Set for Performance Evaluation of Discrete Event Systems

Alain Jean-Marie
Mistral Project
INRIA Sophia-Antipolis
2004 Route des Lucioles
06902 SOPHIA-ANTIPOLIS CEDEX
France

Draft for Ers version 0.2.1, March 11, 1999

Abstract

This document describes the set of tools developped in the Mistral and Sloop teams of INRIA.

Chapter 1

Introduction

Ers is a set of tools for the modeling and performance evaluation of Discrete Event Systems. It is being developped in the Mistral and Sloop projects of INRIA.

This document is intended to give a description of the system, primarily for users, but also for developpers.
In its state, it is extremely incomplete, and should be considered more as a working document than as a reference guide.
The part of the system actually described in this document corresponds to the version 0.1.5 of Ers.

Ers is conceived as a *modular* and *open* system. It is composed of a number of independent programs, each with a precise functionality. Programs communicate through files with specified formats. By chaining the action of several modules, it is possible to acheive relatively complex analyses. For instance, the following command creates a Petri Net representing a rectangular Job Shop of size 3×7, and computes its maximal cycle time:

```
gen_eg -t job_shop -s 3 7 -tl C 12.0 -pl C 2.1 | eg2eg -flatten \
   | eg2matrix -f bruno | scscyc
```

Tools of Ers sometimes make use of other tools and applications developped outside the project. The reader is referred to the specific documentation of these systems. In particular, the tools for distributed simulation of queueing networks (PARSEVAL, PASTEQ and PROSIT) are not described here. Information on them may be found in [16, 15, 17].

The tools described here are intended for the performance analysis of discrete event systems. Several formalisms are used, including Petri Nets, Queueing Networks and Task Graphs. They are (at least conceptually) included in a *modeling and performance evaluation environment* allowing the user to go from model specification to result analysis.
The tools can be grouped in two categories: user (graphical) interfaces, and analysis tools. They communicate with each other through files using several languages. A (partial) picture of the system is given in Figure 1.1. In this picture, program "objects" are represented by rectangles with rounded corners, and file objects are represented by normal rectangles. Arrows indicate possible transitions, with the name of the tool that allows it. The name of the tools developped by INRIA and described herein are written in a bold font. Tools currently under development are indicated with dashed lines.
The description of the system includes

- a description of the analysis tools, with a brief review of the theoretical principles they use (Chapter 2);

- a discussion of the main features of the user interfaces (Chapter 3, very incomplete at this stage);
- the description of the principal data interchange formats and the grammar for the description languages used for the communication between tools (Chapter 4);
- manual pages for the tools themselves (Chapter 5);
- (in a future version) a description of the data structures used in the code of the tools, intended for developpers.

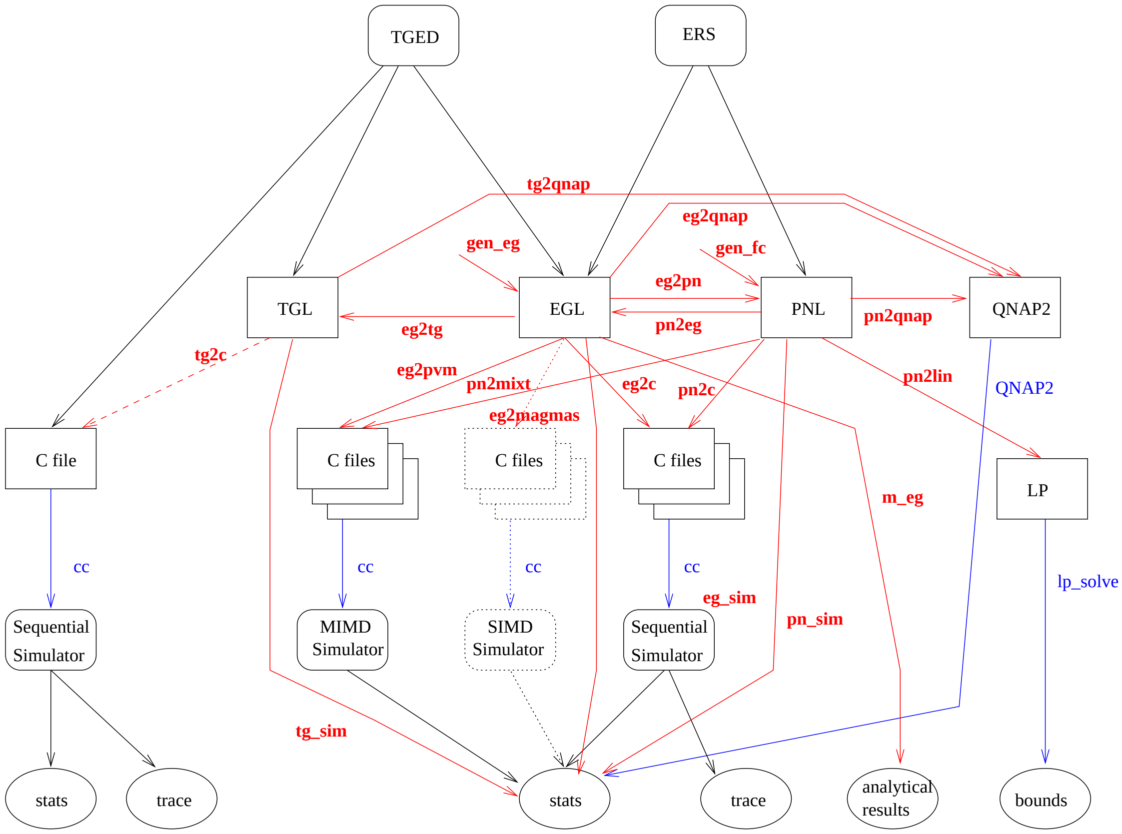


Figure 1.1: A global picture of the system

Chapter 2

Description of the tools

Tools within ERS are grouped into categories, according to the type of object they act on, or the principal technique they use. Current categories are: Basic $(\max, +)$, Petri Nets, Event Graphs, Task Graphs, Queueing Networks, Markov Chains, and generic statistical tools.

2.1 $(\max, +)$ tools

The tools described as "Max-Plus" tools are those concerned with the direct manipulation of matrices and recurrences in the semi-ring $(\mathbb{R}, \max, +)$, and with closely tied representations (Interaction Graphs, Heaps of Pieces). Other tools use $(\max, +)$ concepts for solving problems with a higher order representation (Petri nets, Task Graphs). Those are listed in the relevant sections.

2.1.1 Generalities

$(\max, +)$ systems

In this paragraph, we recapitulate the different concepts and notations used in the description of $(\max, +)$ systems and the related tools.
In the $(\mathbb{R}, \max, +)$ semi-ring, the addition operation (denoted by $\oplus$ is the maximization, and the multiplication operation (denoted by $\otimes$) is the standard addition. In other words:

$$a \ \oplus \ b \ \equiv \max\{a, b\}, \qquad a \ \otimes \ b \ \equiv a \ + \ b \ .$$

The (extended) number $-\infty$ is the neutral element for $\oplus$, and is also absorbing for $\otimes$. It is denoted by ε. These operations extend to operations between matrices with the usual convention for matrix multiplication: given two matrices A and B of compatible sizes,

$$(A \otimes B)_{i,j} \ = \ \bigoplus_k \{A_{i,k} \otimes B_{k,j}\} \ \equiv \ \max_k \{A_{i,k} + B_{k,j}\} \ .$$

Finally, a *linear $(max, +)$ recurrence* is a sequence of vectors $\{X(n), n \in \mathbb{N}\}$ given by

$$X(n+1) \ = \ A(n) \otimes X(n) \ ,$$

for some sequence of matrices $\{A(n), n \in \mathbb{N}\}$ and starting from some vector $X(0)$. Under general assumptions (see [3]) on the stochastic nature of $A(n)$, the following limit exists almost surely:

$$\gamma_i \ = \ \lim_{N \to \infty} \frac{X_i(N)}{N} \ .$$

The number γ_i is the asymptotic *average growth rate* of component i. It is usually referred to as a *Lyapunov Exponnent*.

Heaps of Pieces

Heaps of pieces (or Tetris games) constitute a random dynamic discrete event system that can be represented by a stochastic $(\max, +)$ recurrence. Consider a number N of columns. A piece is described by a lower profile $(\ell_1, \ldots, \ell_N)$ and an upper profile $(u_1, \ldots, u_N)$, where $\varepsilon \leq \ell_i \leq u_i$ for any i. The fact that $u_i = \varepsilon$ means that the piece does not span the column i. See the description of the interface `teted` in paragraph 3.4) for the familiar graphical representation. Given the current height $(X_1, \ldots, X_N)$ of the columns, the piling up of this piece results in a new vector of heights Y given by $Y = A \otimes X$, where the $(\max, +)$ matrix A is constructed as follows:

$$A_{i,j} \;=\; \begin{cases} u_i - \ell_j & \text{if } u_i \neq \varepsilon \text{ and } \ell_j \neq \varepsilon \\ 0 & \text{if } u_i = \varepsilon \text{ or } \ell_j = \varepsilon \text{ and } i = j \\ \varepsilon & \text{otherwise.} \end{cases}$$

To a given set of pieces $\{t_1, \ldots, t_m\}$ corresponds a set of matrices $\{A(t_1), \ldots, A(t_m)\}$. The height of the colums after a sequence of pieces $\{t(i), 0 \leq i < n\}$ has piled up is given by the product:

$$X(n) \;=\; A(t(n-1)) \otimes \ldots \otimes A(t(0)) \otimes X(0) \ ,$$

where $X(0)$ is the initial height (or profile) of columns.

Interaction Graphs

Interaction (or exclusion) graphs in the context of $(\max, +)$ systems are defined and exploited in [6]. An interaction graph is a (possibly directed) graph $(\mathcal{V}, \mathcal{E})$ with weights p_i on the vertices, which are interpreted as probabilities after normalization.
The evolution of the system is according to the following rules. To each node i is attached a "height" X_i. At each time instant n, a node is selected according to the probabilities p_i. If node i is selected, its height $X_i(n)$ becomes:

$$X_i(n+1) \;=\; 1 \;+\; \max_{j|(j,i)\in\mathcal{E}} \{X_j(n)\} \ .$$

If there is no edge (j, i) in the graph, the result is ε by convention, but usually, the arc (i, i) ibelongs to $\mathcal{E}$. All other nodes keep their height: $X_j(n+1) = X_j(n)$, $j \neq i$.

2.1.2 `scscyc`

`scscyc`, performs the spectral analysis of a given deterministic $(\max, +)$ matrix A. The results of the analysis include the maximal eigenvalue of A, the corresponding critical graph structure, and the cyclicity of the matrix. If requested, the program computes the transient and the periodic part of the sequence $A^n X_0$, for a given initial vector X_0.
The algorithm used is the standard Karp algorithm.
In order to circumvent difficulties arising when manipulating real numbers, `scscyc` uses the following convention. All numbers encountered in the description of the matrix are converted (actually, truncated) to integers after multiplication by a "precision parameter". The resolution algorithms use only integers and rationals. For instance, if the precision parameter is set to 1000, entries equal

to 3.141 and 3.1415 will both be handled internally as 3141. The precision is 1 by default, which amounts to considering the integer part $\lfloor x \rfloor$ of all entries x.

Example 1
Input file example1.mat:

```
3
eps     3       0
1       0       eps
eps     0       2

4       0       0
```

Command:

```
    scscyc -f example1.mat -transient
```

Result:

```
cycle mean: 4 / 2

Critical sub-graph:
   eps     0   eps
     0   eps   eps
   eps   eps     0
Agregated critical sub-graph:
   eps     0   eps
     0   eps   eps
   eps   eps     0
Number of critical strongly connected components: scs = 2
 component 0:
    2    1
component 1:
    3

Cyclicity: cyc = 2

T = 0:     4.0000    0.0000    0.0000
T = 1:     3.0000    5.0000    2.0000
T = 2:     8.0000    5.0000    5.0000
T = 3:     8.0000    9.0000    7.0000
T = 4:    12.0000    9.0000    9.0000

length of transient period:  3
```

2.1.3 `mat2c`

`mat2c` transforms the description of a random product of $(\max, +)$ matrices into a C program that performs its simulation. The program essentially a loop repeatedly computing the $(\max, +)$ recurrence equations.

The following example illustrates the main loop of the program for the example of [4]. In this example, there are two matrices:

$$A_0 = \begin{pmatrix} 3 & \varepsilon & \varepsilon & 0 \\ 3 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 0 & \varepsilon & 0 \\ \varepsilon & \varepsilon & 4 & \varepsilon \end{pmatrix} \qquad A_1 = \begin{pmatrix} 1 & 1 & \varepsilon & \varepsilon \\ \varepsilon & 2 & 4 & \varepsilon \\ 0 & \varepsilon & \varepsilon & 2 \\ 1 & 2 & \varepsilon & \varepsilon \end{pmatrix} \tag{2.1}$$

The respective probablities of occurence are 0.9 and 0.1.

Example 2
Input file example2.mat:

```
4
2
0.9 0.1

3 eps eps 0
3 eps eps eps
eps 0 eps 0
eps eps 4 eps

1 1 eps eps
eps 2 4 eps
0 eps eps 2
1 2 eps eps
```

Command line:

```
mat2c -f example2.mat
```

Result:

```
for ( Time = 0; Time < Sim_Length; Time ++ ) {

    Proba = U_0_1();
    for ( k=0; ( k < NB_MATRICES ) && ( Proba > Pr_Cum[k] ); k++ );
      switch( k ) {
      case 0:
        /* multiplication by matrix 0 */
        Y[0] = MAX( X[0] + 3.000000, X[3] );
        Y[1] = X[0] + 3.000000;
        Y[2] = MAX( X[1], X[3] );
        Y[3] = X[2] + 4.000000;
      break;

      case 1:
        /* multiplication by matrix 1 */
        Y[0] = MAX( X[0] + 1.000000, X[1] + 1.000000 );
        Y[1] = MAX( X[1] + 2.000000, X[2] + 4.000000 );
        Y[2] = MAX( X[0], X[3] + 2.000000 );
        Y[3] = MAX( X[0] + 1.000000, X[1] + 2.000000 );
      break;

      } /* end of switch */
```

```
    for ( i=0; i<NB_NODES; i++ )
      X[i] = Y[i];

} /* end of main loop */
```

2.1.4 mat_sim

`mat_sim` performs the simulation of $(\max, +)$ systems of the form

$$X(n+1) \;=\; A(n) \;\otimes X(n) \;,$$

where $A(n)$ is a $(\max, +)$ matrix. Two different types of sequences are handled: *periodic* or *random*. In both cases, the matrix is chosen from a finite set $A_1, \ldots, A_m$.

In the random case, $A(n)$ is equal to A_i with some probability p_i (a *Bernoulli* distribution). The random sequence $\{A(n), n \in \mathbb{N}\}$ is i.i.d.

In the periodic case, if the sequence is $(i_1, i_2, \ldots, i_p) \in [1..m]^p$, then $A(np + j)$ is A_{i_j} for all $n \geq 0$ and $0 \leq j < m$.

The set of matrices and the probability vector (always present, even if not used) is described using the format described in Section 4.5. In the periodic case, the sequence is specified in a separate file by a sequence of numbers.

In the random case, the simulation consists in computing $X(N)$ for some (large) N, starting from an initial vector $X(0) = (0, \ldots, 0)'$. The result of the simulation is $\hat{\gamma}_i(N) = X_i(N)/N$, which is an estimator of the *growth rate* of each coordinate. It is possible to obtain confidence intervals on this value by asking the simulator to make several independent replications of the experiment and compute variance estimates. It is also possible to produce *gradient estimates* for $\hat{\gamma}_i(N)$. See [11] for the theoretical principles.

Two strategies are available for computing $X(N)$. The first one internally uses full matrices and performs repeatedly matrix/product multiplications. The second one uses `mat2c` (see 5) to generate a C program that computes recursively the recursive $(\max, +)$ equations.

Example 3
Input file: example2.mat (see Example 2).
Command line:

```
mat_sim -f exemple2.mat -nevt 1000000 -compile
```

Result:

```
Executing: mat2c -f exemple2.mat -o /tmp/mat_sim.c
Executing: gcc /tmp/mat_sim.c -o /tmp/mat_sim -lm -O2
Executing: /tmp/mat_sim -nevt 1000000
gamma[0] = 2.812716
gamma[1] = 2.812716
gamma[2] = 2.812713
gamma[3] = 2.812714
```

In the periodic case, the "simulation" actually consists in computing the matrix

$$A(n) \;=\; A_{i_n} \;\otimes\; A_{i_n - 1} \;\otimes\; \ldots \;\otimes\; A_{i_1} .$$

and write it out in a specified format.

2.1.5 `mat_bound`

`mat_bound` computes bounds on the Lyapunov exponent of Bernoulli matrix products. Two bounding methods are implemented.

The first one is based on the decomposition of trajectories according to blocks of the form $A^n \otimes B$. It results in a *lower bound* on $EX_i(N)/N$. Note that this is not necessarily a bound on the Lyapunov exponent. This method works only for products with two matrices A and B. The method will be documented in a forthcomming report. The result is obtained as a formal polynomial of the probability p of choosing A, suited for plotting within `gnuplot` or `Maple` for instance.

The second method consists in enumerating all trajectories of a given length N. A lower bound on the Lyapunov exponent is obtained by *oversynchronizing* the system, and an upper bound is obtained by *undersynchronizing* it:

$$E(\min_i X_i(N)) \;\leq\; \gamma_i \;\leq\; E(\max_i X_i(N)) \;.$$

Both expectations are computed starting from the initial vector $X(0) = (0, \dots, 0)'$.

Example 4
Input file: example2.mat (see Example 2).
Command line:

```
mat_bound -f exemple2.mat -nmax 15 -exact -pstep 0.1
```

Result:

```
  # Bounds on gamma computed from Emin/max(X_i(15)),
# Range of p: [0.000000,1.000000]
#
0.000000 2.600000 2.666667
0.100000 2.552996 2.694644
0.200000 2.513205 2.680858
0.300000 2.473132 2.654424
0.400000 2.438570 2.627379
0.500000 2.417623 2.607581
0.600000 2.418505 2.604468
0.700000 2.450256 2.630853
0.800000 2.521911 2.701431
0.900000 2.639113 2.825981
1.000000 2.800000 3.000000
```

2.1.6 `eigenfig`

`eigenfig` is a tool for representing eigenvectors and attraction domains for $(\max, +)$ operators of dimension 3 or 2. It is based on the analysis of [14]. It takes as input a 3×3 matrix A (also 2×2 matrices), and produces a file in the FIG language. Therefore, the application `xfig` must be used to display the figure produced by `eigenfig`. This figure is a two-dimensional representation of the eigenvectors of A, and the set of points which are ultimately mapped to these vectors when iterating A on them.

The representation is a isometric projection of the space $\mathbb{R}^3$ onto the plane. The origin as well as x, y and z axes are represented. They point respectively to the SE, N and SW of the figure. In this representation, points (a, b, c) and $(a + \ell, b + \ell, c + \ell)$ have the same image, which corresponds to considering the $(max, +)$ *projective* space.

When the input matrix is 2×2, it is extended into a 3×3 matrix in the following way:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad \rightarrow \quad \begin{pmatrix} a & b & b \\ c & d & \varepsilon \\ c & \varepsilon & d \end{pmatrix} .$$

This correspond to replicating node 2 of the graph into a third node. The resulting figure is symmetrical. The eigenvectors and the attraction domains of the original matrix can be read on the axis of symmetry.

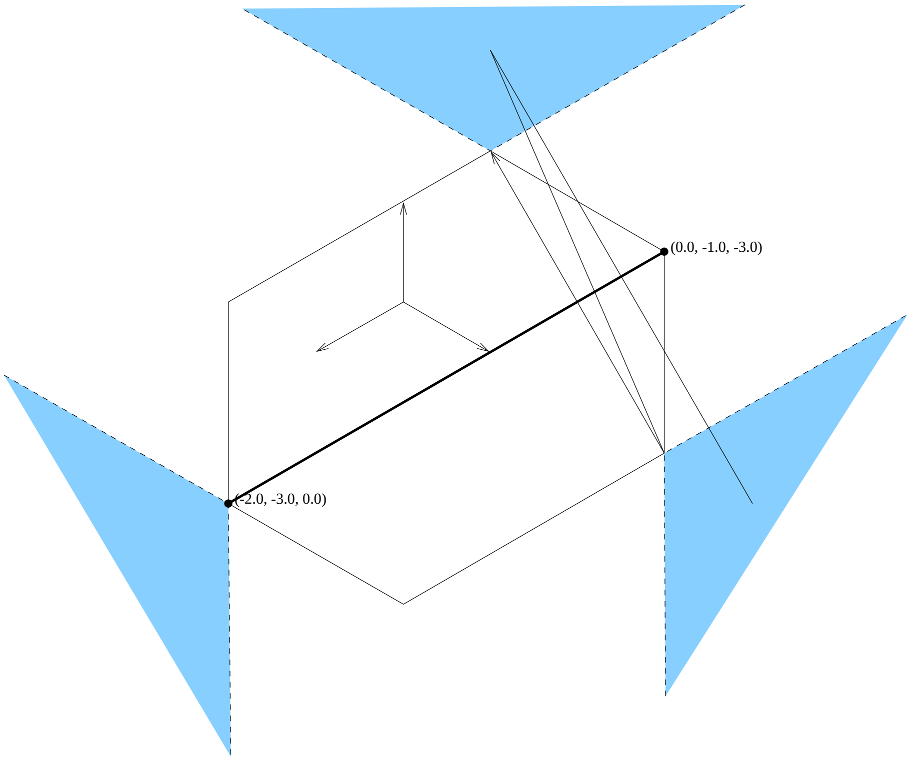
Interpretation of the figure The interpretation of the figure depends on the scs-cyc class of matrix A (see `scscyc` on page 4). The semantics of the main objects in the figure can be summarized as follows. For more details, see [14].

- the point $(0, 0, 0)$ and the three axes are represented.
- Bold points and bold lines represent eigenvectors (or fixed points, in the projective space) of A. The convex hull (in the (max,+) sense) of all the bold objects is the set of all the eigenvectors of A. The coordinates of the extreme points are given in the figure.
- Shaded areas are attraction bassins. The ultimate behavior of an arbitrary point in such an area is the same as the behavior of the angle of that zone.
- Thin lines only appear when A has a cyclicity c larger than 1. They represent the eigenvectors of A^c. The behavior of any point in their convex hull (in the (max,+) sense) is periodic with period c and symmetric w.r.t. the eigenvectors.
- empty strips between the shaded areas are sets of points such that any two points on a line parallel to the direction of the strip have the same behavior.
- Dashed lines indicate a change in the direction of the convergence.
- Broken arrows only appear when option `-trajectory` is used. They represent the trajectory of a point.

Example 5
Input file: example1.mat (see Example 1). Command line:

```
eigenfig -f exemple1.mat -trajectory -o result.fig
xfig result.fig
```

Result:



2.1.7 `gen_gr`

This program generates interaction graphs of a given size N with a given structure. Available structures are:

Cycle The edges are all pairs $(i, i+1)$, $1 \leq i < N$ and $(N, 1)$.

Line The edges are all pairs $(i, i+1)$, $1 \leq i < N$.

Star The edges are all pairs (i, N), $1 \leq i < N$.

Complete The edges are all pairs (i, j), $1 \leq i \leq j < N$.

Complete Bipartite The edges are all pairs (i, j), $1 \leq i \leq N_1 < j \leq N_1 + N_2$.

Random Each edge (i, j) with $1 \leq i < j \leq N$ exists with a given probability p.

In all graphs, including random graphs and the complete bipartite graphs, the "self loops" (i, i) are always present. This is because interaction graphs normally satisty the "consistency condition" which states that objects are in interaction with themselves. This default behavior can be modified. See the manual pages on page 70.
The graph is generated in a format described in Section 4.7.

2.1.8 `gr_bound`

`gr_bound` implements bounds on the Lyapunov exponent γ of an interaction graph. According to [6], Theorem 10 / Corollary 3.6.2, and Theorem 11 / Corollary 3.6.3, the following bounds hold:

$$
\begin{aligned}
\frac{1}{N^2}\sum_i d_i \;&\leq\; \gamma \;\leq\; f^{-1}\left(\frac{\rho}{N}\right)\\
\frac{1}{N}\sum_{(i,j)\in\mathcal{E}} p_i p_j \;&\leq\; \gamma \;\leq\; e\,\max_i\left(\sum_{(i,j)\in\mathcal{E}} p_j\right) .
\end{aligned}
$$

Here, ρ is the spectral radius of the graph, d_i the degree of vertex i, and f is the function:

$$
f(x) \;=\; exp\left(\frac{x\ln(x)+(1-x)\ln(1-x)}{x}\right) .
$$

The first bounds are valid when all p_i are equal. The second hold generally.

2.1.9 `gr2mat`

`gr2mat` transforms an interaction graph into a Bernoulli product of $(\max,+)$ matrices. To each node in the graph is associated a matrix according to the following construction: for the node k:

$$
A_{i,j} \;=\; \begin{cases} 1 & \text{if } i=k \text{ and } (k,j)\in\mathcal{E} \\ 0 & \text{if } i\neq k \text{ and } i=j \\ \varepsilon & \text{otherwise.} \end{cases} \qquad (2.2)
$$

As for `eg2matrix` (see Section 2.3.10), several matrix formats are available. The ones proper to ERS are described in Section 4.5. When the SCILAB format is chosen, the string `%0` representing $\varepsilon=-\infty$. It is possible to specify a prefix string (`A` by default) for the quantities defined.

Example 6
Input file star3.mat:

```
3
 1 1 1
1 0 1
0 1 1
1 1 1
```

Command:

```
gr2mat -f star3.mat -scilab -prefix Star3
```

Result:

```
Star3_Size = 3
Star3_Number = 3
Star3_Weights = [ 1 1 1]
Star3_0=#([
           1.0000   %0       %0     ;
           %0       0.0000   %0     ;
           1.0000   %0       0.0000])

Star3_1=#([
           0.0000   %0       %0     ;
```

```
          %0        1.0000    %0    ;
          %0        1.0000    0.0000])

Star3_2=#([
          0.0000    %0        1.0000;
          %0        0.0000    1.0000;
          %0        %0        1.0000])
```

2.1.10 gr2c

gr2c transforms an interaction graph into a C program performing the simulation of the corresponding stochastic recurrence. See mat2c for the principle.

2.1.11 gr_sim

gr_sim performs the simulation of the $(\max, +)$ stochastic recurrence equation corresponding to the evolution of an interaction graph. The principle of this simulation is the same as for the general mat_sim. Two strategies are available: one using matrix products, the other using a compiled program generated by gr2c . The matrices used are described in 2.1.9.

2.1.12 gr2tetris

gr2tetris transforms an interaction graph into a Heap of Pieces.
There are several sets of pieces corresponding to an exclusion graph. All of them have the same number of pieces, which is the number of nodes in the graph. The number of columns and the shape of the pieces depends on the algorithm used.
All algorithms look for *cliques* in the graph. The larger the maximal size of the cliques is, the longer the computation time, but (hopefully) the smaller the number of columns.
When only cliques of size 2 are used, the cliques are the edges of the graph, and there are as many columns as edges. This is the "standard" method.
The resulting heap of pieces is produced in a format suited for loading into teted (see 4.10).

2.1.13 tet_bound

tet_bound computes bounds on the growth rate of a random Heap of Pieces.
The lower bound is computed using the simple observation that:

$$\gamma \ \geq \ \max_{c \text{ column}} \mathbb{E}(h(c) - l(c)) \ = \ \max_{c \text{ column}} \sum_{p \text{ piece spanning } c} \mathbb{P}(p) \times (h^{(p)}(c) - l^{(p)}(c)) \ ,$$

where $h^{(p)}(c)$ and $l^{(p)}(c)$ are the high and low profiles of the piece p in column c. This bound is particularly fast to compute.
The other bounds are obtained by partially exploring the Markov chain of profiles in the projective space. Let C be the number of columns in the stack. If $X(n) = (X_1(n), \ldots, X_C(n))$ is the profile of the stack after n pieces have been dropped, then the process $\{(X_1(n), Z(n)), n \in \mathbb{N}\}$ with $Z(n) = (X_2(n) - X_1(n), \ldots, X_C(n) - X_1(n))$ is a Markov reward (or Markov additive) process. The Lyapunov exponent is the expected reward.
The construction of a bounding Markov chain is as follows. Starting from state $(0, \ldots, 0)$, the state space of the chain$\{Z(n)\}$ is explored until a specified number of states is reached. The chain is then truncated, the transitions leading "outside" begin sent back to the "flat" profile $(0, \ldots, 0)$ while the

reward is set to the maximum height of the profile. This amounts to *oversynchronizing* the heap of pieces, and gives a larger value for the Lyapunov exponent. Similarly, if the reward is set to the minimum height of the profile, the heap of pieces is *undersynchronized* and a loger bound for the Lyapunov exponent is obtained. Observe that it might be that the state $(0, \ldots, 0)$ is not even recurrent with the original chain.
Several issues have a possible influence on the result. First, of course, the number of states explored. Then also, the order in which unexplored states are explored. At the moment, the order is FIFO, which amounts to a breadth-first-search in the covering tree of the transition graph. Other choices may be considered. For instance, taking the probabilities of pieces into account, or using the stationary probabilities resulting from a previous exploration to detect the more important states.

2.2 General Petri Net tools

2.2.1 `gen_fc`

`gen_fc` generates Petri Nets with a Free Choice topology, in the PNL language. The graphs generated are actually dual of those obtained by `gen_eg` (see Section 2.3.6 p. 15).

2.2.2 `pn_read`

`pn_read` is an application that simply reads a Petri Net in a PNL file, performs neighborhood analysis and dumps the information. It is useful for debugging purposes.

2.2.3 `pn2eg`

`pn2eg` writes a description in the EGL language of a Petri Net given in the PNL language. If the Net is not an Event Graph, no output is produced.

2.2.4 `pn2graph`

`pn2graph` is an utility that converts a Petri Net in one of several graph representation format. The idea is to attempt to solve the problem of displaying a given Net, presumably generated without a graphical interface. There exist several public domain programs able to perform some layout of graphs. Some are even specialized in it.
Currently, the format for three graphical systems are available:

`dot/dotty` from the graphviz system of AT&T Bell Labs.

`daVinci` from the University of Brehme. Information at: `http://www.informatik.uni-bremen.de/ davinci/`

SCILAB from INRIA. Information at: `http://www-rocq.inria.fr/scilab/`

2.2.5 `pn_bound`

This is a structural analysis tool that computes bounds on the marking of places or subnets of a Petri Net. The method consists in deriving a linear program and solving it using `linsolve`.

2.2.6 `pn2lin`

This program computes bounds on the throughput of a Markovian Petri Net. The method consists in deriving a linear program and solving it using `linsolve` See the paper of Z. Liu for details.

2.2.7 live_fc

live_fc performs liveness tests on Petri Nets of the Free Choice family. The test is based on the analysis of [Gaujal et. al.].

2.2.8 fc2totmatrix

fc2totmatrix converts a Petri Net of the Free Choice family into matrices that serve as the basis for liveness tests (see live_fc, Section 2.2.7).

2.2.9 pn2qnap

pn2qnap transforms a Petri Net into an equivalent queueing network in the form of a QNAP program. Such a transformation is possible only when the Petri Net is a *routed* Net in the sense of [Gaujal et al.].

2.2.10 pn2c

pn2qnap transforms a Petri Net into an equivalent queueing network in the form of a QNAP program. Such a transformation is possible only when the Petri Net is a *routed* Net in the sense of [Gaujal et al.].

2.3 Event Graph Analysis Tools

2.3.1 Generalities

An Event Graph is a Petri Net with the constraint that places have exactly one input transition and one output transition.
Event Graphs have a special relationship with the $(\max,+)$ algebra through their *evolution equations*. Consider an Event Graph with the FIFO property: the n-th firing to start at a certain transition ends before the $n+1$-st one. Let $X_i(n)$ be the time at which the n-th firing starts at transition i. Then the vectors $X(n) = (X_1(n), \ldots, X_T(n))'$ satisfy the following relation in the $(\max,+)$ algebra:

$$X(n) \;=\; A_0(n) \otimes X(n) \;\oplus\; A_1(n) \otimes X(n-1) \oplus \ldots \oplus A_\mu(n) X(n-\mu) \;.$$

The $(\max,+)$ matrices $A_i(n)$ are derived from the topology of the Net and the timing assumptions. If the timings are stochastic, the matrices are also. See [3] for details.
When the network is *live* (see Section 2.3.2), this implicit equation can be "solved" using the *Kleene star*. Given a matrix A, it is defined as $A^\star = \oplus_{m=0}^{\infty} A^{\otimes m}$. One obtains the fundamental recurrence:

$$X(n) \;=\; A_0(n)^\star A_1(n) \otimes X(n-1) \oplus \ldots \oplus A_0(n)^\star A_\mu(n) X(n-\mu) \;. \tag{2.3}$$

Because of this relationship with the $(\max,+)$ algebra, Event Graphs have received a special attention in the early development of ERSFor this reason, they have a special description language, EGL. The syntax of this language is reviewed in Section 4.1. The general purpose Petri Net language, PNL, has been introduces later. It may replace EGL completely one day. The advantage of EGL over PNL is that the representation of Nets is more compact: places are not explicitly described.

2.3.2 `live_eg`

`live_eg` is a topological analysis tool which checks for the liveness of the Event Graph. An Event Graph is live if and only if all its cycles contain at least one token. The analysis consists in checking that the graph obtained by removing marked places has no cycle. This is performed by a topological sorting. When a cycle is detected, it may be produced as output on request.
This application is typically used by graphical interfaces in order to detect and display empty cycles in an Event Graph.

2.3.3 `eg2c`

`eg2c` converts an Event Graph into a C program which performs its simulation. The program implements the $(\max, +)$ evolution equations. See 2.1.3 p. 5 for a similar generation in the case of $(\max, +)$ matrices.

2.3.4 `eg2qnap`

`eg2qnap` converts an Event Graph described in the EGL language into a Queueing Network with the same behavior, described in the QNAP language. The synchronization constraints are implemented using the "FUSION" and "FISSION" constructs of QNAP.

2.3.5 `eg_sim`

`eg_sim` performs the simulation of the evolution of an Event Graph. Two simulation "engines" are available.

2.3.6 `gen_eg`

`gen_eg` is a utility to generate Event Graphs with certain regular structures of arbitrary sizes and stochastic properties. This utility is useful combined with solution tools, simulators etc. It is also useful combined with transcription tools (`eg2pn`, `eg2tg`) to create similar structures with other representations.
The Event Graph created has all its transitions recycled. Therefore, it has automatically the FIFO property, and it can be analyzed by $(\max, +)$ tools such as `eg_sim`.
The available topologies are:

Cycle Transitions and places are arranged in a circular manner. The number P of places (counting the recycling places) is equal to two times the number of transitions T: $P = 2T$. This topology is also called "closed series".

Lines Transitions and places are arranged in a line. The first transition is therefore a source, and the last one is a sink. Here, $P = 2T - 1$. This topology is also called "open series".

Double cycles Transitions and places are arranged in two disjoint cycles connected by one place. The sizes of the two cycles, say S_1 and S_2, are parameters. The transitions of the first cycle are numbered from 1 to S_1, and those of the second cycle from $S_1 + 1$ to S_2. The connecting place is between transitions 1 and $S_1 + 1$. There are $S_1 + S_2$ transitions and $2(S_1 + S_2) + 1$ places.

Job Shops Transitions and places are arranged in a rectangle, and are connected by two series of cycles: one horizontal, and one vertical. Such a Petri Net represents the repetitive processing of parts by machines in a factory. The tokens in one of the series of cycles represent parts to be processed, the ones in the other series represent the machines. In this Petri Net, $P = 3T$. A 3×4 Job Shop is obtained in the next example. A picture of the Event Graph is in Figure 2.1 (this picture was obtained using `ers`).

Example 7
Command:

```
gen_eg -t job_shop -s 3 4 -pl C 0.0
```

Result:

```
Size <- 12
1 "Trans_0_0" E   1.0000 , 1 C 0.0 1 0.0, 5 C   0.0000 1 0.0, 2 C   0.0000 1 0.0;
2 "Trans_0_1" E   1.0000 , 2 C 0.0 1 0.0, 6 C   0.0000 1 0.0, 3 C   0.0000 1 0.0;
3 "Trans_0_2" E   1.0000 , 3 C 0.0 1 0.0, 7 C   0.0000 1 0.0, 4 C   0.0000 1 0.0;
4 "Trans_0_3" E   1.0000 , 4 C 0.0 1 0.0, 8 C   0.0000 1 0.0, 1 C   0.0000 1 0.0;
5 "Trans_1_0" E   1.0000 , 5 C 0.0 1 0.0, 9 C   0.0000 1 0.0, 6 C   0.0000 1 0.0;
6 "Trans_1_1" E   1.0000 , 6 C 0.0 1 0.0, 10 C   0.0000 1 0.0, 7 C   0.0000 1 0.0;
7 "Trans_1_2" E   1.0000 , 7 C 0.0 1 0.0, 11 C   0.0000 1 0.0, 8 C   0.0000 1 0.0;
8 "Trans_1_3" E   1.0000 , 8 C 0.0 1 0.0, 12 C   0.0000 1 0.0, 5 C   0.0000 1 0.0;
9 "Trans_2_0" E   1.0000 , 9 C 0.0 1 0.0, 1 C   0.0000 1 0.0, 10 C   0.0000 1 0.0;
10 "Trans_2_1" E   1.0000 , 10 C 0.0 1 0.0, 2 C   0.0000 1 0.0, 11 C   0.0000 1 0.0;
11 "Trans_2_2" E   1.0000 , 11 C 0.0 1 0.0, 3 C   0.0000 1 0.0, 12 C   0.0000 1 0.0;
12 "Trans_2_3" E   1.0000 , 12 C 0.0 1 0.0, 4 C   0.0000 1 0.0, 9 C   0.0000 1 0.0;
```

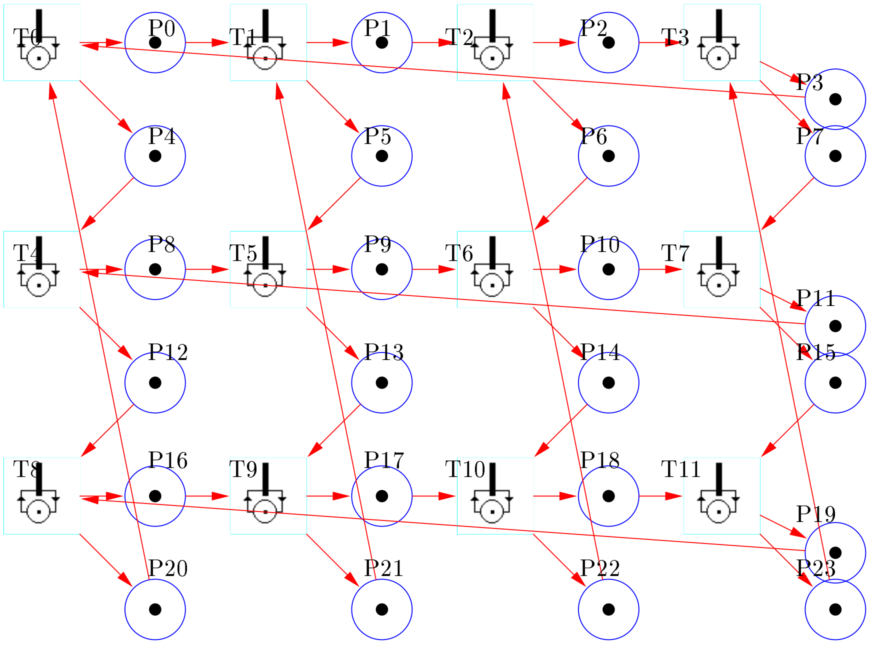


Figure 2.1: A 3×4 Job Shop

2.3.7 `eg2eg`

`eg2eg` transforms an Event Graph into another Event Graph, after performing various modifications. The possible operations are:

Recycling consists in adding a "recycling" place to each transition. The places added have the following characteristics: one mark, zero timing, one input and one output, both being the recycled transition.

Flattening consists in transforming the Net so that the result has a maximal marking of 1, but has the same dynamical behavior as the original graph. There are several methods for doing this. See the report [9] (or [8]) for details.

Determinizing consists in replacing all random distributions in the graph by their means. This may be useful for computing bounds on performance measures (throughputs, markings...).

These operations may be combined.

2.3.8 `eg2pn`

`eg2pn` converts an Event Graph described in the EGL language into the same Petri net, described in the PNL language. The translation is straightforward.

2.3.9 `eg2tg`

`eg2tg` converts an Event Graph described in the EGL language into the same Petri net, described into an "equivalent" repetitive task graph in the TGL language. The translation converts each transition into a task with a "max" synchronization type. Places are converted into dependence links. If the marking of the place is 0, the dependence is of type "DATA", that is, between tasks of the same program. If the marking is larger than one, the dependence is of type "PROGRAM", that is, between tasks of different programs, the offset in the number of the program being the value of the marking. The timing of places is ignored.

2.3.10 `eg2matrix`

`eg2matrix` translates an Event Graph into a $(\max, +)$ matrix. The Event Graph considered has to be "flat", that is, have a maximal marking of 1 (see `eg2eg` Section 2.3.7). It also has to be FIFO in order to admit a $(\max, +)$ representation. When the Event Graph is deterministic, the resulting matrix is $A_0^\star \otimes A_1$. When the Event Graph is stochastic, it is an *instance* of $A_1(n) \otimes A_0(n)^\star$.
The matrix may be written in various formats corresponding to different systems or analysis tools. The standard format is the one suited for analysis with `scscyc` (see Section 2.1.2). Another possibility is the SCILAB format for $(\max, +)$ matrices, with the string `%0` representing $\varepsilon = -\infty$. It is then possible to specify a name for the objects (matrix and initial vectors).

2.3.11 `eg2graph`

`eg2graph` transcripts an Event Graph into a FIG file representing the corresponding developped task graph.

2.3.12 `m_eg`

`m_eg` computes the distribution of the waiting time in a deterministic Event Graph driven by a Poisson source. It uses `mk1` (see 2.5.2 p. 19).
The method is applicable to Event Graphs with the following structure:

- There exists one and only one "source", that is, a recycled transition with no other input than its recycling place. This source must also have one single output.
- The rest of the network must be a *strongly connected* Event Graph with *deterministic* times.

The waiting time of a customer/token is formally defined as follows. Let $T(n)$ be the time at which the n-th token is produced by the source. Let $X_i(n)$ be the time at which the n-th firing begins at transition i. Then the n-th waiting time is defined as

$$W_n \;=\; X_i(n) \;-\; T(n)\;.$$

The stationary waiting time is the limit of W_n when n goes to infinity, when it exists.
Note that the quantity W_n actually corresponds to the time spent by a tagged customer between its arrival in the system and the beginning of its service at a certain transition (thus the name "waiting time" by analogy with queueing theory), only when there is a path with no tokens between the source and the transition. Otherwise, this quantity may actually be negative.

2.3.13 The MAGMAS system

MAGMASS was a set of simulators developped by M. Canales [7]. These simulators run on a Connection Machine 2.
At one point, three simulators were available. They all used the "Spatial" method, but with different ways of handling matrices, and different implementation languages. They are:

- C-Star code with full matrices.
- C-Paris code with sparse matrices.
- C-Star code with sparse matrices.

The algorithms are described in [7] and [2]. The source code is not included in ERS but is available on request.

2.4 Task Graph Analysis Tools

2.4.1 `tg_read`

`tg_read` is an application that simply reads a Task Graph in a TGL file, performs neighborhood analysis and dumps the information. It is useful for debugging purposes.

2.4.2 `tg_crit_path`

`tg_crit_path` computes the critical path in a task graph.

2.4.3 tg_sim

tg_sim performs a simulation of a task graph.
If the graph is not repetitive (*i.e.* contains no program links), several samples of the graph are drawn, and statistics on the makespan are gathered.
If the graph is repetitive, the graph is (virtually) unfolded a number N of times, and the makespan in a sample of the resulting graph is computed. The result is then an estimate of the *cycle time* as the ratio of the total makespan by N.

2.4.4 tg2qnap

tg2qnap transforms a (repetitive) task graph into a QNAP program. The synchronization constraints are implemented using the "FUSION" and "FISSION" constructs of QNAP.

2.4.5 tg_read

The program tg_read reads a task graph in the TGL language and produces some information on its topology.
This application is typically used to check the syntaxical correctness of files, and for debuggin purposes.

2.5 Queueing Theory Tools

2.5.1 md1

md1 computes the stationary waiting time distribution in the $M/D/1$ queue. The result is in the form of points suited for plotting.

2.5.2 mk1

mk1 computes the distribution of the waiting time in a single server queue with Poisson inputs and known service times (so called the "M/Known/1" queue). It implements the analysis of [12].
The distribution of the response time of customer n is given by

$$P(W_n \leq x) \ = \ H_n(b_0, b_1, \dots, b_n; x) \ ,$$

where $\{b_i\}$ is an increasing sequence and H_n a known function. The increments of the sequence $\{b_i\}$ correspond to the service times of consecutive customers.

2.5.3 mrouted1

mrouted computes the value of the average waiting time of customers in the following queueing system. The customers arrive according to a Poisson process. They are routed according to a *periodic* scheme to one of several queues, each with a single server with constant service, and an infinite capacity. Customers may also be rejected, which amounts to routing them to the "exterior".
The analysis uses the fact that the workload in each queue behaves like the one in a particular $M/Known/1$ queue (see Section 2.5.2).

2.5.4 `onoff_sim`

This application performs the simulation of a fluid queue with a single "on/off" source and a constant capacity. The parameters of the source (laws of on and off periods, peak rate) are specified using a syntax common with `gen_iid` (Section 2.6.1).
The result of the simulation is an estimation of the distribution of the workload in the queue. The technique used for constructing the estimator is described in the manual page p. 95. Confidence intervals may be requested. They are estimated using the regeneration points at the beginning of "on" periods with zero workload.

Markov Chains

A few Markov Chain tools are provided. None of them is efficient for large complex chains. They prove however handy for demonstration purposes, in connection with the `mced` editor.

2.5.5 mc_solve

`mc_solve` is a solver that attempts to find *a* stationary distribution of some Markov chain numerically. In the current version, the method simply consists in iterating the generator on the initial distribution given by the initial state in the input file. Convergence issues aside, this would lead to the stationary distribution of the final classes reachable from this initial state.
The result of the analysis is normally the probability vector The solver handles reward Markov chains, and produces the expected reward upon request.

2.5.6 mc_sim

`mc_sim` simulates the evolution of some Markov chain. It produces, according to the user's specification, a trajectory of the chain, and/or sojourn time statistics, as well as average rewards when applicable.

2.6 Statistical tools

2.6.1 gen_iid

`gen_iid` generates samples of *i.i.d.* sequences of random variables. The distribution of this random variable is specified on the command line according to a syntax described in the table of the manual page p. 68. This syntax is common to other tools that use a stochastic law as parameter: `gen_eg` (Section 2.3.6 p. 15), `gen_fc` (Section 2.2.1 p. 13) or `onoff_sim` (Section 2.5.4 p. 20).

2.6.2 moment

`moment` computes the empirical moment of a sample.

2.6.3 distrib

`distrib` computes the empirical distribution of a sample. One algorithm implemented computes the function

$$\hat{F}(x) \;=\; \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}\{X_n \leq x\} \;.$$

and lists it in the output. This is the most accurate result possible but it requires the sorting of the sample, and produces an output of the same order of size as the input sample.
The seconf algorithm implemented actually computes the values of the function $F(x)$ at certain points only. This is equivalent to constructing a histogram of the sample.

2.6.4 laplace

`laplace` computes the empirical Laplace transform of a sample, given by:

$$\hat{F}^*(x) \;=\; \frac{1}{N} \sum_{n=1}^{N} e^{-sX_i} \;.$$

Chapter 3

Graphical Interfaces

This chapter gives the description of the different user interfaces to the ERS tools.
Those of these interfaces that have been written with Tcl/Tk are part of the ERS package: `pned`, `tged`, `gred`, `teted`, `Mtool`. Other interfaces of an earlier generation were written using the proprietary GSS package: `ers`, `pnanim`, `TGED`, `tganim`, `Marktool`). These are not provided in the standard package, but they may added to it on request. Some of the functionalities of these interfaces have not been ported yet to their replacement.

3.1 `pned`: an editor for Petri Nets

`pned` is the graphical interface for drawing and analyzing Petri Nets.

Net creation The creation of Petri Nets is done either by placing places, transitions and links "by hand", or by using the "Create" menu.
When drawing with the interface, the user must select one *drawing mode* by clicking one of the icon/buttons at the left of the display: "place", "transition", "link", "Poisson source" (the fish in a blue box).

The "magnifier" icon gives access to the mode where objects attributes can be inspected and modified.
The attributes of places and transitions include the timing associated with them (firing times/holding times). The distribution of this time is currently limited to constants...

The "pirate" icon gives access to the delete mode. WARNING: there is no way to undo mistakes at this stage.
Finally, the last icon on the right gives access to the "select" mode.
When drawing graphs, the following keyboard accelerators are active. WARNING: some of these functionalities are not accessible otherwise:

`a` inspect attributes of the current object

`+` increment the marking of a place

`-` increment the marking of a place

`f` fire a transition

`u` un-fire a transition

`r` recycle a transition

`s` select an object (toggle)

`<del>` delete an object

The "create" menu allows one to create easily a Petri Net of an arbitrary size with a given regular structure. It is also possible, through the "Matrix" entry of this menu, to create an Event Graph with a topology given by an adjacency matrix, either in the format in Section 4.6, or in the SCILAB format.

Inputs and Outputs Petri Nets may be saved and read in various formats.
Through the "File" menu, the Petri Net is saved in a format particular to `pned` (to be described) which includes graphical information
Through the "Export" menu, the Petri Net can be written in one the description languages PNL/EGL, transformed in a Queueing Network described in QNAP, or into a $(\max, +)$ matrix.

Net analysis The "Solve" menu gives access to solution methods for Event Graphs. The methods are those of the deterministic $(\max, +)$ spectral analysis (see `scscyc` in Section 2.1.2), or of Poisson-Driven techniques (see `mk1` and `m_eg` in Sections 2.5.2 and 2.3.12).

3.2 `tged`: an editor for Task Graphs

`tged` is the graphical interface for drawing and analyzing Task Graphs.

3.3 `gred`: an editor for Interaction Graphs

`gred` is the graphical interface for drawing and analyzing Interaction Graphs (see Section 2.1.1).

Graph creation Graphs contain two sorts of objects, nodes and edges.
When drawing graphs with the interface, the user selects one *drawing mode* by clicking one of the icon/buttons at the left of the display: "node", or "edge".
Nodes are placed on the main canvas when clicking the left button while in the Node mode. WARNING: at the present time, it is not possible to move nodes once they are on the canvas.

The "magnifier" icon gives access to the mode where objects attributes can be inspected and modified.
Nodes have a single attribute: their weight. Arcs do not have attributes.

The "pirate" icon gives access to the delete mode. WARNING: there is no way to undo mistakes at this stage.
When drawing graphs, the following keyboard accelerators are active.

`a` inspect attributes of the current object

`s` select an object (toggle)

`<del>` delete an object

Finally, the last icon on the right gives access to the "select" mode. Selection is useful for creating *cliques* (complete subgraphs) through the "Edit/Clique selection" menu.
The "create" menu allows one to create easily a graph of an arbitrary size with a given regular structure. It is also possible, through the "Matrix" entry of this menu, to create a graph with a topology given by an adjacency matrix, given one of the formats of Section 4.6.

Inputs and Outputs Interaction graphs may be saved and read in various formats.
Through the "File" menu, the graph is saved in a format particular to `gred` (see Section 4.11) which includes graphical information
Through the "Export" menu, the graph can be written as a simple adjacency matrix, a format suited for analysis with tools of the `gr*` family. Another possibility is to generate the corresponding set of $(\max, +)$ matrices (see Section 2.1.1) in a format suited for analysis with tools of the `mat*` family, or in the SCILAB format. Finally, it is possible to transform the interaction graph in a "Tetris" game (see `gr2tetris`, Section 2.1.12).

Analysis The "Solve" menu gives access to solution methods for interaction graphs. Three methods are presently available:

- simulation using `gr_sim` with a compiled simulator (see Sections 2.1.11 and 2.1.10);
- bounds using `gr_bound` (see Section 2.1.8);
- bounds using the $(\max, +)$ matrix representation, and `mat_bound` with the `-exact` option (see Section 2.1.5). The `-nmax` parameter is computed so that the computation lasts only a few seconds.

Experience proves that the bounds obtained with the two methods are complementary: `gr_bound` gives a rather good lower bound, while `mat_bound` has gemarally a better upper bound.

3.4 `teted`: an editor for Heaps of Pieces

`teted` is a graphical interface for creating, editing and solving problems involving heaps of pieces (Tetris games).
The user interface manipulates four different objects:

the piece store containing the pieces defined so far;

the piece editor allowing to specify the shape of the pieces, their color and weight;

the drop zone where the heap of pieces appears;

the sequence editor that allows to specify periodic sequences of pieces.

The display normally shows the piece store and the drop zone only. The piece editor and the sequence editor may be brought forth using the "Edit" menu.

Creating Set of Pieces In order to create a set of pieces, the user specifies first the number of columns, throught the menu “Edit/New”. This initializes the piece store and the drop zone. New pieces can then be drawn by calling the piece editor (menu “Edit/Set of Pieces”).
Pieces are drawn by clicking on the squares of the drawing board with the left button. Pieces may be moved around, rotated and mirrored using the buttons on the left of the display, or the “Move”, “Rotate”, “Mirror” menus. Squares not fitting in the display after such a transformation are clipped out. The color of pieces may be changed by entering a valid color name in the “Color” window, or using the “Color” menu.
When a piece is finished, is may be stored in the piece store using the “Store” menu. Several options are availlable: replacing an old piece or storing a new one. Pieces already present in the store are loaded into the editor by clicking on them with the left button. WARNING: at this moment, there is no checking whether pieces have been saved or not; there is no “undo” facility either.

A faster way to create set of pieces is to use the “Create” menu. Three classes of sets with a given “topology” are provided: cycle, complete and star. These denominations correspond to the Interaction Graph associated with the heap of pieces. See Sections 2.1.1 and 3.3 for details.

Pieces may be deleted from the store by entering the “delete mode” (orange button at the left of the display), then clicking on the pieces with the left button.
Sets of pieces may be saved and loaded in the `teted` format (see Section 4.10) with the “File/Save” and “File/Load” menus. It is possible to merge set of pieces, provided that they have the same number of columns.

Specifying sequences of pieces The “editor” for sequences is called by the menu “Edit/Sequences of Pieces”. The editor simply contains a field where the sequence is written, and allows input/output and several manipulations of sequences.
A sequence is of the form

```
1 3 < 5 ( 6 7 )^ 3 > 1
```

The numbers refer to piece numbers in the piece store. The construction `a<b` means all numbers from `a` to `b` in increasing order, modulo the current number of pieces. The symbol `n` denotes the current number of pieces. For instance, if there are 5 pieces in store, `2<4` means `2 3 4`, `2<n` means `2 3 4 5`, whereas `4<2` means `4 5 1 2`. The contruction `a>b` is similar but in decreasing order. Subsequences may be grouped with parenthesis and raised to any integer power. For instance, `(1 2)^ 3` means `1 2 1 2 1 2`. The sequence above is expanded into `1 3 4 5 6 7 6 7 6 7 6 5 4 3 2 1`.
With the “Edit” menu, sequences may be “cleaned” (formatted in a standard form) and expanded into purely numerical sequences. Expanding a sequence may be necessary before saving it, because applications working on sequences (*e.g.* `mat_sim`) Section 2.1.4) may not be able to interpret non-numerical symbols. There is a “Sequence/Save Expanded” menu for this purpose.

Dropping pieces Pieces may be dropped into the drop zone in various ways. One is to enter the “drop” mode by clicking the yellow button at the left of the main display. In this mode, clicking on pieces in store with the left button will drop them onto the stack. WARNING: there is no way of “undropping” pieces at the time.
Sequences of pieces may be dropped (all at the same time) by using the “Drop/Drop Once” menu of the Sequence editor.

Finally, pieces or sequences may be dropped automatically with the Tetris Game. The control pannel is invoked using the menus "Solve/Tetris Game" in the main display, or "Drop/Tetris/Game" in the sequence editor. Two modes for the game are possible: random and deterministic (in sequence).
In the random game, pieces are chosen randomly, proportionally to their weight (Bernoulli choice).
In the deterministic game, pieces are dropped according to the sequence specified by the editor (if applicable).

Analysis and connection to $(\max, +)$ tools The following operations are possible, in connection with the $(\max, +)$ analysis tools.

- Export the game as a Bernoulli product of $(\max, +)$ matrices, either in one of the formats described in 4.5, or as a SCILAB file (main menu "Export/as (max,+) matrices").
- Export the game as an Interaction Graph in the format described in 4.7. This is possible only when all pieces have height one exactly (main menu "Export/as interaction graph").
- Simulate the game (random or periodic) using `mat_sim` in order to obtain estimates of the growth rates of columns (main menu "Export/Simulate").
- Compute the cycle time for periodic sequences, using the `scscyc` application (sequence editor, menu "Analysis").

3.5 `Mtool`: an application navigator

`Mtool` is an interface facilitating the use of the numerous independent application of ERS by allowing

- to easily access to the available options of the programs, with a reminder of their signification
- to specify input files
- to browse and store output results.

Upon initialization, `Mtool` presents iteslf as the juxtaposition of a file browser and several series of buttons, arranged in columns. Each button is labelled with the name of an application. Clicling on a button causes a window, containing all known options for this application, to pop up. Option may be activated using the mouse; a reminder for their signification is displayed automatically. The manual page for the application is also readily available.
When the application is run, the output is displayed in a text window. It may then be edited, and saved to a file.

3.6 ERS: an editor for Petri Nets

Features The main feature of the editor is its ability to manipulate a set of regular objects in a *virtual* way.
These regular objects are: series, cycles, two-dimentional rectangular jobshops.
This feature deserves a precise description, to be provided... In the meantime, information can be found in [10].

Code generation and analysis There is the possibility of generating code from the interface in many different representation languages. This is done through the "Code Generation" menu, and its four sub-menus.

- Generate for Event Graph will generate code in EGL language (see section 4.1). This will work only if the Petri Netis an Event Graph.
- Generate for general Petri nets will generate code in the PNL language (see section 4.2).
- Generate for QNAP will generate a QNAP2 file. The generation is done in two phases: first, an EGL file is produced, then the utility `eg2qnap` is used.

  This will work only if the Petri Netdoes not have competition routing.

There are presently three distinct ways to generate a simulation from the editor:

1. *via* an EGL language representation. This can be performed through the "Simulation" menu, item: "Simulate Event Graph". This has the effect of generating code in the EGL language and simulate it using the general purpose Event Graphsimulator `eg_sim` (p. 59). The results appear in the window from where the editor was called.
2. generate code in the EGL language, convert it in C-language files, compile them and execute the result. The EGL code is generated as above, and one uses `eg2c` (see p. 49) to create the C files and the executable. This possibility is not accessible yet through the "Simulation" menu of ERS, and the work has to be done by hand.
3. *via* a QNAP language representation. This can be performed through the "Simulation" menu, item: "Simulate QNAP". This has the effect of generating code in the QNAP language and simulate it using QNAP2. A special version 9 of QNAP2 (including FUSION/FISSION mechanisms) is necessary, with an initialisation bug corrected. The results appear in the window from where the editor was called.

All work only with Event Graphs, since there are no simulator in the system (yet) able to handle more general networks, even using QNAP code.
If the operation does not work properly from the interface, the user has always the possiblility of generating the relevant code (using the "Generate Code" menu) and perform the simulation "by hand".
Note that these operations are likely to change in the next versions of the interface.

Graphics Generation The graphical appearance of the model may be translated into the FIG language, and from there in a variety of other graphical languages (Postscritpt, LaTeX, etc.) using the `fig2dev` utility.

Bugs Some known bugs are:

- The EGL code generated will be wrong if two transitions are connected by a place located into a submodel: the output link of the place will not be followed properly, and the place will not appear in the EGL file.

3.7 `pnanim`: a trace-based animator for Petri Nets

This tool is based on the Animator package, provided with GSS.
The procedure to use it is as follows:

1. Create a model, using the `ers` editor (see the description of the editor in section 3.6). Save it in a file, say: `model.mod`.
2. Generate EGL code, in another file, say: `model.egl`.
3. Run `eg2c` on the file `model.egl`, specifying a trace beginning and/or a trace end (see the `eg2c` man page p. 49). For instance:
   `eg2c -f model.egl -te 50`
4. Compile the produced C files together with the "skeleton" files (see `eg2c` again).
5. Run the resulting simulator. A trace file named `trace.results` is produced.
6. Call the animator on the model file, with: `pnanim model.mod`
7. Use the "OPEN TRACE FILE" item of the background menu, and fill in the name of the trace file: `trace.results`.
8. Use the "Control Window" item of the menu to call a window allowing to control somewhat the execution of the animation.

There you go!
It is possible to re-run a simulation and display the new run without exiting and entering again the animator (provided the network has not changed). For this: generate the new simulation from the `ers` editor, and load the new trace file using the "Open trace file" menu of `pnanim`.
Caution: NEVER exit `pnanim` with the "Done" button. Use the "Abort" button, or kill the window.

3.8 TGED: an editor for Task Graphs

It is also based on GSS.
It is conceived to edit task graphs with the possibility of representing repeating (periodic) graphs.
The essential things to know are:

- there is only one type of tasks...
- there are three types of links:
  - standard task dependency links;
  - program dependency links (bold), with which one indicated that tasks of some program may depend on tasks of previous programs. The depth of the dependency is an attribute of the link
  - resource constraint links (dotted), not used yet.
- the type of synchronization at the "entrance of a task" (OR or AND) can be specified in the attributes of the task.

The main functionalities of the interface can be grouped in five themes: graphical features, code generation, simulation, graphics generation and miscellaneous analyses. Most of these functionalities depend on programs end/or files external to the interface itself, called "resources" hereafter. The presence of these resources is checked when the application starts, and menu options depending on unacesible resources are disabled.

Graphical Features In addition to the usual GSS features, there is the possiblility of creating vertical "strings" of tasks. This can be done using the "Create String" item of the main menu. This creates a small window where the number of tasks in the string can be specified (the window usually has to be enlarged first). The string is created when the "Create " button is pressed. If it is pressed several times, several strings will be created, at a horizontal distance of 50 pixels. The starting point of the first string is the point where the menu was selected.
Warning: the possibility of changing the default parameters of tasks and links using the attributes of the model (attibutes "Default Task Duration" etc.) is not implemented yet. Tasks and links are all created with a constant and zero duration. Program links have a default depth of 1.

Code Generation Code can be generated in five different languages, corresponding to the five sub-menu items. Note that code generation in some languages are not always possible: for instance, a Task Graphadmits a Petri Netrepresentation (actually, an Event Graphrepresentation) only when all task synchronizations are of "and" type.

- Task Graph will generate code in TGL language (see section 4.3). . The code is put in the file `sim_code.tgl`.
- Petri Net will generate code in EGL language (see section 4.1). This will work only if the Task Graphis an Event Graph. The code is put in the file `sim_code.egl`.
- C Language will generate code in the C language. This code is that of a simulator, and is intended to be compiled and executed to obtain performance measures (see the Simulation feature). The code is put in the file `sim_code.c`.
- Generate for QNAP will generate a QNAP2 file. The generation is done in two phases: first, a TGL file is produced, then the utility `tg2qnap` is used. The code is put in the file `sim_code.qnp`.

  This will work only if the Task Graphhas only "and" synchronizations, but future versions will allow generation for other synchronizations.

Simulation Five simulators are accessible, coresponding to the five different codes generated. Selecting an item in the sub-menu of "Simulate " will first generate code in the corresponding formalism. Then, according to the choice:

- Task Graph will call the `tg_sim` simulator.
- Petri Net will call the `eg_sim` simulator. Note that if some tasks are not repeated (they are not in a cycle containing at least one program link), the result of the simulation is erroneous for some (yet) unknown reason.
- C Language will compile the C language and execute the result. This requires the accessiblity of a C compiler, of the include file `stochastics.h` and of the object code `stochastics.o`.

- Generate for QNAP will use the QNAP2 simulator. Note that a special version of QNAP2, Version 9, may be necessary due to a bug in the standard release.

Graphics Generation The graph can be saved in the FIG graphic language , in color or black and white. The FIG code is used to generate code in the Postscript, LaTeXor PsTeX languages using the `fig2dev` program .

Miscellaneous Features The additional feature "Show Critical Paths" is available. It uses the program `tg_crit_path`. When the durations of tasks and/or communications are random, a sample (a realization) of the graph is drawn, and critical path analysis is performed on it. Note that (for the moment), program links are ignored, so that only the "building block" is analyzed.
In the future, more analysis tools should be connected to `tged`.

Resources Resources are external programs called by the interface (through system calls) to perform some of its functionalities. All of them are configurable through environment variables. If the variables are not set, a default value is sought. If they are set to a name not beginning with a / character, the resource is sought in all directories contained in the `$PATH` variable.
The current list of resources, nature and their default values is:

`TG2QNAP`	executable	`/u/yragael/0/bin4/tg2qnap`
`EGSIMULATOR`	executable	`/u/yragael/0/bin4/eg_sim`
`TGSIMULATOR`	executable	`/u/yragael/0/bin4/tg_sim`
`SIM_DIR`	directory	`/u/yragael/0/meval/ajm/rech/sim_eq/task_graphs/tasktool`
`SIM_OBJ`	readable file	`/u/yragael/0/meval/ajm/rech/sim_eq/task_graphs/tasktool/stochastics`
`QNAP_BIN`	executable	`/u/yragael/0/etc/qnapv93`
`C_COMPILER`	executable	`/usr/ucb/cc`
`TG_CRIT_PATH`	executable	`/u/yragael/0/bin4/tg_crit_path`
`TMP_EGL_FILE`	writeable file	`/tmp/task_graph.egl`
`TMP_TGL_FILE`	writeable file	`/tmp/task_graph.tgl`
`TMP_C_FILE`	writeable file	`/tmp/task_graph.c`
`TMP_EXEC_FILE`	writeable file	`/tmp/task_graph.simulator`
`TMP_PATH_FILE`	writeable file	`/tmp/tasktool.crp`
`TMP_QNAP_FILE`	writeable file	`/tmp/task_graph.qnp`
`FIG2DEV`	executable	`/u/yragael/0/bin4/fig2dev`

3.9 `tganim`: a trace-based animator for Task Graphs

As `pnanim`, it is based on the Animator package.
The procedure to use it is as follows:

1. Create a model, using the `tged` editor (see the description of the editor in section 3.8).
2. Run the "C Language" item of the "Simulate" menu.
3. Call the animator on the model file, with:
   `tganim model.mod`
4. Use the "OPEN TRACE FILE" item of the background menu, and fill in the name of the trace file: `trace.results`.

5. Use the "Control Window" item of the menu to call a window allowing to control somewhat the execution of the animation.

The functionalities of the animator are the same as that of `pnanim`, that is, very poor.

3.10 marktool: a Maple interface for solving Markov chains

Marktool is a simple graphical interface for representing Markov chain, and solving them using Maple. It is based on GSS4.

Drawing Chains The interface proposes four types of nodes:

- Discrete Time Markov Chain (DTMC) states, represented as plain circles,
- Continuous Time Markov Chain (CTMC) states, represented as clocks,
- Probability probes, represented as small histograms,
- Throughput (or transition rate) probes, represented as small taps.

There are four types of links:

- DTMC state transition links, connecting DTMC states; they have as attribute the transition probability (1 by default).
- CTMC state transition links, connecting CTMC states; they have as attribute the transition rate (1 by default).
- Probability probe links, connecting a probability probe node with states (either DTMC or CTMC); they have as attribute a multiplicative factor (1 by default). They can be connected with several states. When solving the Markov chain, the linear combination of stationary probabilities corresponding to the connected states and the multiplicative coefficients will be computed.
- Rate probe links, connecting a rate probe node with states (either DTMC or CTMC) or state transition links; they have as attribute a multiplicative factor (1 by default). They can be connected with several states or links. When solving the Markov chain, the linear combination of the stationary trantition rates corresponding to the connected states (meaning, the total transition rate into/out of this state) and links (transition rate on this particular link) with the multiplicative coefficients will be computed.

Notice that one may *a priori* use both CTMC and DTMC nodes on the same

Solving Chains The interface offers the ability of:

- generating Maple code for the model;
- direclty solving for the stationary probabilities.

These functionalities are accessible through the **Generate Code** and **Solve** items of the main menu.

Maple code can be generated in two versions. The first one consists in the code creating the matrix of the chain, called `P`. This matrix is the (transposed) infinitesimal generator Q in the case of a CTMC, and the matrix $P - I$ in the case of a DTMC. The second one consists in the above matrix, plus the code solving for the stationary probabilities by solving the system $P\pi = 0$ and normalizing the solution.
When solving the chain, the above code is generated (in a temporary file) and Maple is called on it. The result appears in the window from where the editor was called. It consists in the list of stationary probabilities, and the list of agregated probabilities/rates corresponding to the probes that were connected to the chain. This solution has been tested for Maple, version 5.2 only. It may fail for other versions of Maple, due to a change in the symbolic name of the "unknowns" which appear when solving linear systems. Adapting the code seems relatively easy however.

Ideas for future developments Among the numerous ideas for new functionalities, one may single out:

- symbolic (parametric) Markov chain, and symbolic solution of the chain. This would be even more spectacular if the system is connected with Modline and the Experimenter/Reporter...
- Automatic uniformization of a CTMC into a DTMC
- Hierarchical description for "large" chains. Would allow the construction of big chains, but then again, Maple would not be the best way of solving the chain.

Chapter 4

Languages and Data Formats

This chapter describes the different languages and data formats used by the different tools of ERS. For historical reasons, there are two languages for representing Petri Nets: the "Event Graph Language" (EGL) and the more general "Petri Net Language".
Task graphs have also a representation language: the "Task Graph Language", described in section 4.3.

4.1 The EGL language

The (pseudo) BNF description of the language is as follows.

```
<network> ::= <declarations> <transition>*
<declarations> ::= <size_dcl>
<size_dcl> ::= ["Size" | "Taille"] "<-" <integer>
<transition> ::= <transition-id> [<name>] [<name>] <desc-law> ["," <desc-place>]* ";"
<transition-id> ::= <integer>
<desc-law> ::= <law-id> <parameter>+
<law-id> ::= C | E | B | H | R | U | D
<parameter> ::= <integer> | <real>
<desc_place> ::= <transition-id> [<pl_data>] <desc-law> <marking_desc>
<pl_data> ::= <name> | <name> <name> | <name> <name> <name>
<marking_desc> ::= <integer> <real>*
```

The nonterminals `<integer>`, `<real>` and `<name>` are respectively standard integers, real and character strings between double quotes (").
The semantics of the language is as follows. The declarations state the size of the network, that is, the number of transitions. Each transition declaration is the enumeration of:

- the transition identificator (in the current version of the language: a number);
- two (optional) strings, for instance for names or other data;
- the description of the timing asociated to the transition;
- the list of output places of the transitions; a place is coded by the identification of the output transition of this place, then a description of the holding time in this place, then the initial population with (if needed), initial set-up times.

In addition, there are optional data fields, see below.

In order that the network be correct, the following rules must be followed:

- If the `Size` of the network is declared to be N, there should be exaclty N transitions declared, numbered from 1 to N. The order of their declaration is however irrelevant.
- The parameters of the laws of timings have types (real or integer) that depend on the law itself. The proper types and number of parameters should be used.
- When an initial population is given, there should be as much set-up times as tokens declared. If not, the missing set-up times will be taken as 0.0, and the extra ones ignored.

The "`<name>`" fields may be use to transmit to analysis programs information useful for data analysis. For instance, the Event Graphanimator `pnanim` (section 3.7) requires the GSS name of graphical objects (nodes and links), which have to be written by the simulator into the trace files. See also `eg2c` (p. 49).

Specification of timing In EGL and PNL, timing of transitions and places is specified with the same syntax. This syntax is also used in applications generating EGL and PNL: `gen_eg` (Section 2.3.6) and `gen_fc` (Section 2.2.1).

The general syntax is (most of the time) a single-letter identificator followed by a list of parameters:

```
<desc-law> ::= <law-id> <parameter>+
<law-id> ::= B | C | D | g | G | H | O | P | R | U
<parameter> ::= <integer> | <real>
```

but the exact number and type of the parameters to supply depend on the law. They are reviewed in the following table.

Symbol	Name	Number of Param.	Domain	Interpretation
C	Constant	1	real ≥ 0	value
E	Exponential	1	real ≥ 0	mean
R	Erlang	2	integer, real ≥ 0	number of steps, mean
U	Uniform	2	real, real	[a,b]
H	Hypergeometric	2n	real	see below
O	On/Off	3	all real ≥ 0	mean, mean on, mean off
P	IPP		see below	
B	Bernoulli	1	$[0, 1]$	proba of 0
g	Geometric on $\{0, 1, \dots\}$	1	real	proba of > 0
G	Geometric on $\{1, 2, \dots\}$	1	real	proba of > 1
D	Discrete	2n	real	see below

Table 4.1: Description of timing

Some distributions have a variable number of parameters:

- The Hypergeometric distribution is a mixture of n exponential distributions. The sequence of parameters is $p_1, m_1, p_2, m_2, \dots, p_n, m_n$, where the p_i are probabilities, and the m_i the mean value of the corresponding exponential distribution. The distribution function is

$$F(x) \; = \; \sum_{i=1}^{n} p_i \; (1 - e^{-x/m_i}) \; .$$

- The general Discrete distribution is described by a sequence of values, followed by their respective probabilities: $v_1, \ldots, v_n; p_1, \ldots, p_n$. The syntax for this is:

  ```
  D [ v1 ...  vn ] [ p1 ...  pn ]
  ```

  The distribution function is then:

  $$F(x) \; = \; \sum_{i=1}^{n} \; p_i \; \mathbf{1}_{\{x \geq v_i\}} \; .$$

- The IPP (Interrupted Poisson Process) is specified by the average interarrival time of a Poisson process, followed by the description of two distributions: the distribution of "on" periods during which the random variable is exponential and "off" periofs, during which arrivals are interrupted. For instance, if the Poisson process has intensity 10 (average interarrival 0.1), if "on" period have a fixed duration of 2 units of time, and "off" periods an exponential distribution with average 3 units of time, the specification will be:

  ```
  P 0.1 C 2.0 E 3.0
  ```

4.2 The PNL Language

The (pseudo) BNF description of the language is as follows.

```
<network> ::= <declarations> <object>*
<declarations> ::= <trans_dcl> <place_dcl>
<trans_dcl> ::= "Transitions" "<-" <integer>
<place_dcl> ::= "Places" "<-" <integer>
<object> ::= <transition> | <place>
<transition> ::= <desc-transition> ["," <place_id>]* ";"
<desc-transition> ::= <transition-id> [<name>] <desc-law>
<place> ::= <desc_place> ["," <trans_id>]* ";"
<desc-place> ::= <place-id> [<name>] <desc-law> <marking_desc> [<routing_desc>]
<place-id> ::= <integer>
<transition-id> ::= <integer>
<desc-law> ::= <law-id> <parameter>+
<law-id> ::= C | E | B | H | R | U | D
<parameter> ::= <integer> | <real>
<marking_desc> ::= <integer> <real>*
<routing_desc> ::= <random> | <cyclic> | <competition>
<random> ::= "Random" "[" <real>+ "]"
<cyclic> ::= "Cyclic" "[" <integer>+ "]"
<competition> ::= "Competition" | "Competition" "Priority" "[" <integer>+ "]"
```

The semantics is as follows. The number of transitions and places are declared first. Follows the declaration of places and transitions themselves:

- A transition declaration consists in its identificator, the description of its firing law, and the list of output places;
- A place declaration consists in its identificator, the description of its holding time law, its initial population with initial set-up times, an optional routing discipline, and the list of output transitions.

A routing discipline may be:

- Cyclic (keyword: `Cyclic`): the tokens are routed by repeatedly reading a specified list of output transitions;
- Random (keyword: `Random`): the tokens are routed randomly between all output transitions according to a specified list of probabilities;
- Competition (keyword: `Competition`): the tokens are routed according to the "first finished firing" of output transitions. The additional keyword `Priority` allows to specify a priority order in case the competition rule is not sufficient to break ties (for instance when transitions are instantaneous).

In some cases (for example Event Graphs), routing information is unnecessary and can be left empty. As for EGL programs, the number of transitions and places actually declared should be exactly the number announced in the declaration part, and the numbering of places and transitions has to be consecutive, starting from 1. The order of declaration is irrelevant.

4.3 The TGL language

A TGL description of a Task Graph is a list of task and link descriptions.
Tasks are described as in:

```
TASK <name> [<given_name>]
  DURATION <law_desc>
END
```

The first name is the principal one. It is used to specifiy the connections of links (see below). The second is optional. They will typically be both used when the code is generated from a user interface: the first one will be the name of the object for the graphical interface (the GSS name), and the second the name chosen by the user.

Link descriptions are as follows:

```
LINK <name>
  TYPE <PROGRAM | DATA>
  IS <NECESSARY | OPTIONAL>
  FROM <name>
  TO <name>
  DURATION <law> | DEPTH <number>
END
```

Here, the `TYPE` attribute is one of: `DATA` or `PROGRAM`. The first refers to a link indicating some data dependency (the usual one in task graphs), whereas the second indicates a dependence of one task from one *iteration* of the graph on a task of a previous iteration. In that case, the graph is assumed to be the (infinite) periodic repetition of the graph consisting of normal (`DATA`) links, the different instances being connected with `PROGRAM` links.
`DATA` links have an attribute `DURATION` describing the duration of the task. `PROGRAM` links have a `DEPTH` attribute telling how many instances back the task actually connected if to be found.
Both links have an attribute `IS` describing how the program treats the dependence. If this attribute is `NECESSARY`, the task to which this link is connected cannot start before the data (or signal) went through this link. On the other hand, if the attribute is `OPTIONAL`, the data (or signal) may be ignored. A task which input links are all `NECESSARY` starts only when *all* data have arrived ("and" synchronization), whereas if all links are `OPTIONAL`, the task may start as soon as one data has arrived ("or" synchronization) [5].
Finally, the `FROM` and `TO` attributes indicate the ends of the link. The names used are that of the tasks.

4.4 $(\max, +)$ matrix

A $(\max, +)$ matrix is described as follows:

```
N
A-11 A-12 ...  A-1N
A-22 A-21 ...  A-2N
...
A-N1 A-N2 ...  A-NN
```

The signigication of the entries is:

- `N` is the size of the matrix
- The remaining `N`×`N` numbers are the entries of the matrix. Entries are real numbers, or the character string `eps` to denote $\varepsilon = -\infty$.

4.5 Random products of $(\max, +)$ matrices

```
N
P
p1 ...  pP
matrix1
matrix2
...
matrix P
```

Each matrix `matrixp` is specified either in *Full format:*:

```
A1-11 A1-12 ...  A1-1N
A1-22 A1-21 ...  A1-2N
...
A1-N1 A1-N2 ...  A1-NN
```

of in the *Sparse format:*

```
sp
i1 j1 v1
i2 j2 v2
...
iq jq vq
end
```

Both descriptions may be mixed. Blank lines are allowed to separate matrices.
The signification of the entries is the following:

- `N` is the size of the matrices
- `P` is the number of matrices
- The numbers `p1, ..., pP` are the *weights* of the matrices, which will be normalized into probabilities. They have to be positive.
- The entries `A-ij` of the full format are the `N x N` entries of the matrices A. Entries are real numbers, or the character string `eps` to denote $\varepsilon = -\infty$.
- The lines `(i,j,v)` in the sparse format represent the non-ε of the matrix: $A_{i,j} = v$. The values `v` must be real numbers. The numbering of the indices `i,j` starts with 0.

4.6 Adjacency matrices for graphs

```
N
a11 a12 ...  a1N
a22 a21 ...  a2N
...
aN1 aN2 ...  aNN
```

The signification of the entries is:

- The integer number `N` is the number of nodes.
- The following numbers describe the adjacency matrix: `aij` is 1 if there is an edge from node `i` to node `j`, and 0 otherwise.

4.7 Interaction graphs

Two formats for the declaration of interaction graphs are supported: one with full matrices, one with sparse matrices.
The format for full matrices is a slight modification of the format for adjacency matrices:

```
N
p1 ... pN
a11 a12 ... a1N
a22 a21 ... a2N
...
aN1 aN2 ... aNN
```

The signification of the entries is:

- The integer number `N` is the number of nodes.
- The positive numbers `p1, ..., pN` are the weights of the nodes, converted to probabilities by normalization.
- The following numbers describe the adjacency matrix as in Section 4.6.

The format for sparse matrices is the following:

```
sp
N
p1 ... pN
i1 j1
i2 j2
...
ip jp
```

The signification of the entries is:

- The keyword `sp` indicates that the following is in the "sparse" format.
- The integer number `N` is the number of nodes.
- The couples `ik jk` are the arcs of the matrix. The range of the entries is between 0 and `N`-1.

Note that the parsing of the matrix continues as long as pair of numbers can be read. Reading a number out of range will cause an error. The end of the matrix can be signalled by anything else (for instance the keyword `end`.)

4.8 The MCL language

The "Markov Chain Language" is currently very poor. It is foreseen that a more powerful way of describing chains will be available in the future.
For the time being, the description of Markov chains basically consists in listing the possible transitions.

The first line in a MCL file should contain the time type of the chain (`continuous` or `discrete`), the optional specification `reward` if this is a Markov Reward Process, and the type of the matrix representation (`full` or `sparse`). The combination `full` and `reward` is not supported.
The second line should contain the size N (number of rows and columns) of the matrix.
The rest of the entries describe the transitions. According to the matrix type and the time type, the entries are arranged as:

- $N \times N$ entries for a `discrete` and `full` matrix;
- $N \times (N-1)$ entries for a `continuous` and `full` matrix, the *diagonal* elements of the infinitesimal generator being left out;
- an arbitrary number of triples `i j val`, terminated by the keyword `stop` in the case of a `sparse` matrix. The state numbers i and j *must be* numbered from 0 on. The element represented by `val` is then $P_{i,j}$ of $Q_{i,j}$ according to the time type. When the chain is a Markov Reward Process, this "triple" becomes a 5-tuple: `i j val wr reward`. The elements `reward` is the reward associated with the transition $i \to j$. Omitting the keyword `wr` may lead to unexpected behaviors.

The last entry in the file is the number of an initial state.
For instance, the following file

```
discrete reward sparse
3
0 0 0.1 wr 0
0 1 0.9 wr 1
1 2 1.0 wr 4
2 0 1.0 wr -1
stop
0
```

represents a Discrete Time Reward Markov chain with probability transition matrix P and reward matrix R:

$$P = \begin{pmatrix} 0.1 & 0.9 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \qquad R = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 4 \\ -1 & 0 & 0 \end{pmatrix}.$$

4.9 Petri Nets

The format used by `pned` for storing Petri Nets with their graphical information better discussed over an example. The file `line4.pned` is:

```
Tr 1 57.0 45.0 63.0 75.0 "T_WHITE_1" 0
Tr 2 177.0 45.0 183.0 75.0 "" 0
Tr 3 297.0 45.0 303.0 75.0 "" 0
Tr 4 417.0 45.0 423.0 75.0 "" 0
Pl 1 105.0 45.0 135.0 75.0 "" 0 1
Pl 2 225.0 45.0 255.0 75.0 "" 0 1
Pl 3 345.0 45.0 375.0 75.0 "" 0 1
```

```
TrPl 1 1 1 60.0 60.0 105.0 60.0
TrPl 2 2 1 180.0 60.0 225.0 60.0
TrPl 3 3 1 300.0 60.0 345.0 60.0
PlTr 1 2 1 135.0 60.0 180.0 60.0
PlTr 2 3 1 255.0 60.0 300.0 60.0
PlTr 3 4 1 375.0 60.0 420.0 60.0
```

Keywords `Tr`, `Pl`, `TrPl`, `PlTr` denote respectively: transitions, places, links from transitions to places, and places to transitions.
The number following `Tr` and `Pl` is the number of the object in its category. Numbers stats with 1 and *must* be consecutive in the file. The four numbers following are the coordinates of the bounding box of the object in the graphical representation. The standard dimensions of a transition are 6×30, and the diameter of a place is 30. The following string is the name of the object. Strings *must not* contain white spaces. These are replaced by the sequence of characters: `_WHITE_`. The remaining entries represent the timing, and, for places, the marking.
For links, the two first numbers are the identities of the objects connected, the third number is the weight of the link, and the remaining numbers (an even number) is the list of points through which the link goes.

4.10 Heaps of Pieces

The format for heaps of pieces used by `teted` is:

```
Columns N

Piece i
Weight pi
Color <color-descr>
LoProf 1 l1 2 l2 ...  N lN
HiProf 1 u2 2 u2 ...  N uN
...
```

The signification of the entries is:

- `N` is the number of columns
- For each piece `i`, where `i` must range sequentially from `1` to `N`, the description of piece `i` comprises:
  - its weight p_i. Weights are normalized into probabilities.
  - its color. Here, `<color-descr>` may be the name of a color as recognized by the system (see *e.g.* the output of `showrgb` under Unix/X11), or a RGB hexadecimal code of the form `#rrggbb`.
  - its lower profile, as a sequence of pairs (i, ℓ_i), where ℓ_i may be the character string `eps`.
  - its upper profile, as a sequence of pairs (i, u_i), where u_i may be the character string `eps`.

For instance, the code for the “star with three branches” is (file `st3.teted`):

```
Columns 3

Piece 1
Weight 1
Color #ff0000
LoProf 0 0 1 0 2 0
HiProf 0 1 1 1 2 1

Piece 2
Weight 1
Color #ff8000
LoProf 0 0
HiProf 0 1

Piece 3
Weight 1
Color #00ff00
LoProf 1 0
HiProf 1 1

Piece 4
Weight 1
Color #0080ff
LoProf 2 0
HiProf 2 1
```

4.11 Interaction Graphs

The format for graphs is illustrated in the following example (file `K32.gred`):

```
Node 1 154.0 103.0 170.0 119.0 1
Node 2 118.0 151.0 134.0 167.0 1
Node 3 61.0 132.0 77.0 148.0 1
Node 4 61.0 72.0 77.0 88.0 1
Node 5 118.0 54.0 134.0 70.0 1
Arc 1 4 155.0 109.0 77.0 83.0
Arc 1 5 158.0 105.0 131.0 69.0
Arc 2 4 122.0 153.0 74.0 87.0
Arc 2 5 126.0 151.0 126.0 70.0
Arc 3 4 69.0 132.0 69.0 88.0
Arc 3 5 74.0 134.0 122.0 69.0
```

Lines beginning by the keyword `Node` specify the location and the attributes of a node. The first number is... its number. The four following are the coordinated of the corners of the bounding box of the node. The last number is the weight of the node. Node numbers should appear consecutively, and range from one on.

Lines beginning by the keyword `Arc` specify arcs. The two first numbers are the node numbers of the extremities. The remaining pairs of numbers are the coordinates of the segments of the arc.

Chapter 5

Manual Pages

NAME

aman – Display manual pages

SYNOPSIS

`aman name`

DESCRIPTION

Displays the man page of the program `name`.
The man page is compiled from the file `name.tex` into a DVI file and displayed with `xdvi`.
The list of manual pages known to the system can be obtained with `awhat`.

NAME

awhat – Find available “aman” manual pages

SYNOPSIS

`awhat [<string>] [-all] [-h] [-?]`

DESCRIPTION

Displays the “`aman`” manual page of the program `name`. This is the equivalent of `man -k` for UNIX man pages.

If called without arguments, `awhat` will recompile the index from where the information is sought.

If called with a string pattern, it will find all manual page headers containing the string.

If called with the `-all` argument, all known pages are displayed.

See also `aman`.

OPTIONS

- `-all` looks for all manual page headers;
- `-h, -?` displays usage information.

NAME

distrib – Empirical distribution construction

SYNOPSIS

```
distrib   [-f <file>] [-o <file>] [-xfrom <real>] [-xto <real>] [-xval <real>]
          [-xstep <real>] [-queue] [-log] [-log10] [-steps] [-debug] [-h] [-?]
```

DESCRIPTION

Reads a list of numbers and builds the empirical distribution function associated to it. The result is a list of points suited for plotting.

By default, numbers are taken from the standard input and written on the standard output. This behavior may be modified using the `-f` and `-o` options.

It is possible to control (with the `-xfrom` and `-xto` options) the range on which the function is constructed. By default, this range is $[0, 10]$.

The default behavior is to construct the complete empirical distribution: if there are N samples read, say, $X_1, \ldots, X_N$, then $F(x)$ is the number of values less or equal to x, divided by N:

$$F(x) \;=\; \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}\{X_n \leq x\} \;.$$

Points are sorted using a binary search tree. Therefore, the computation time depends on the distribution analyzed, and on the "disorder" of the values read.

A second possibility is to compute the empirical distribution function at certain points only. This is equivalent to constructing a histogram of the sample. The width of the boxes is defined by the option `-xstep`. Together with the values of `-xfrom` and `-xto`, this defines a range of points x_i, and the function $F(x_i)$ is computed only at these points. This analysis is faster than the complete one.

In both cases, the result is the sequence of values of F at sample points. If the option `-steps` is specified, the result is a step function.

The `debug` options produces extra information on the values read. In the case when the complete distribution function is asked, it performs recursive explorations of the tree and is not efficient for large sample sizes.

OPTIONS

`-?` produces usage information.

`-debug` produces information on the values read.

`-f <file>` name of the input file; if none is specified, the graph is read from the standard input.

`-h` produces usage information.

`-o <file>` name of the resulting file; the standard output is the default.

`-steps` produces the coordinates of a step function; the default is to produce simple points.

`-xfrom <real>` sets the lower end of the range to analyze. The default is 0.

`-xto <real>` sets the lower end of the range to analyze. The default is 10.

`-xval <real>` computes the function at a single point.

BUGS
Unknown yet.

DESIRABLE DEVELOPMENTS
The sorting may certainly be improved in speed.

NAME
eg2c – C code generation for Event Graphs

SYNOPSIS

```
eg2c      [-bs <buf_size>] [-f <file>] [-debug] [-tb <time>] [-te <time>] [-from
          <int>] [-to <int>] [-tmp <dir>] [-nevt <int>] [-h] [?]
```

DESCRIPTION
Creates a set of C-language files, which purpose is to perform an equational simulation of an Event Graph. The equations are the standard "$(\max, +)$" equations (see [1]). They are ordered using a level dependency approach [7].
The result should be compiled with the standard, network independent piece of code: "skeletton.c", "draw.c" and "results.c". A copy of these files can currently be found in the directory:

$ERSHOME/lib/eg_sim

For instance:

```
cc skeletton.c -o simulator -lm
```

This in turn produces an executable, simulator, which execution provides the performance results. (If you prefer not to compile everything yourself, use eg_sim instead with the option -compile.)
The resulting executable accepts the option -nevt <int> which allows to change the number of events simulated.
The standard behavior of the simulator is to produce the list of the last firing times computed together with the estimated cycle time. If however the two options -from and -to are used, the "waiting times" of tokens between the two specified transitions will be computed. The result will therefore be a list of numbers (suited for analysis with the distrib or moment utilities). The waiting time is defined as the time elapsed between the n-th production of a token at a transition, and the n-th start of consumption at another.
There is provision for generating a trace, in connection with the editor ers. Trace is selected when either of the -tb or -te options have been specified. When trace has been selected, the simulation program produces it (in the Animator language) in a file trace.results.
The Net is read form an EGL-file and the following files are produced:

- egc_graph_parameters.h a configuration file containing the parameters of the Event Graph and of the simulation.
- egc_declarations.h other declarations.
- egc_initial_evolution.c the code for the beginning of the evolution, for which firing times (may) depend on initial lag times of tokens.
- egc_evolution.c the code for the main loop.
- egc_end_firing_trace.c the code for generating an Animator-compatible trace at the end of the simulation (when tracing is asked).

These files may be written in a directory different from the current one using the -tmp option.
The simulation program uses a cyclic buffer to (potentially) increase locality and execution speed. When the buffer is full, the statistics are collected, and new random variables are drawn. The buffer size can be controlled with the -bs option, but it has to be larger than the initial marking.

OPTIONS

`-debug` produces information on the network read.

`-f <file>` name of the input EGL file; if none is specified, the graph is read from the standard input.

`-bs <buf_size>` the size of the data buffer; 1000 by default.

`-nevt <int>` overrides the simulation length of the EGL file.

`-tb <time>` the time at which the trace begins; 0 by default.

`-te <time>` the time at which the trace ends; 1000 by default.

`-from <int>` the transition from which reseponse times are measured.

`-from <int>` the transition to which reseponse times are measured.

`-tmp <dir>` the directory where the C files are to be written; the default is the current directory.

`-?` produces usage information.

`-h` produces usage information.

BUGS

The trace has been caught at sending the clock back in time. Tss tss. Otherwise, the animation looks nice.
The trace won't work anyway if the buffer is too small.

DESIRABLE DEVELOPMENTS

There should be less files produced.

The trace mechanism should be more tested, especially when the beginnng of the trace is not at 0.
There could be some optimization in the size of the code produced and the execution time.

NOTES

The equations that are simulated correspond to the "real" behavior of the system under certain conditions. It it the responsibility of the user to check that the network does satisfy these conditions. Known sufficient conditions are: all transitions are recylced, or belong to a cycle with one token; constant delays in places.
The trace will work properly if the input file has been generated from the graphical interface: the names of all the objects have to be known to the simulation program in order ot generate a valid trace code.

NAME

eg2eg – Perform EGL to EGL transformations

SYNOPSIS

```
eg2eg   [-f <file>] [-downstream] [-upstream] [-split] [-flatten] [-recycle]
        [-debug] [-optimize] [-linprog] [-determinize] [-o <file>]
```

DESCRIPTION

Translates an Event Graph (coded in EGL) into another Event Graph (also coded in EGL), performing the transformations specified by the options. By default, the input graph is simply rewritten, which provides some kind of pretty printing.

One possible transformation is to enforce the recycling of transitions (option `-recycle`). This is done by adding, if needed, a place with one token and a null timing. The place is given the name "Dummy_Place".

The option `-determinize` changes the distributions of firing durations and hoding times, and replaces them by a constant distribution with the same average.

An other possible class of transformations is to "flatten" the graph, which means transform it in a dynamically equivalent graph with at most one token per place. This is useful for the parallel simulation of the graph. Three algorithms can be used to flatten a graph: tree-like transformations, downstream (this is the default) or upstream, and the linar splitting. Details on these algorithms may be found in [8].

Finally, a possibility is to optimize the marking with respect to the flattening of the graph. What is minimized is the number of new transitions created. This option is compatible with the recycling option and with the flattening option. The optimization procedure uses a linear program which is solved using `lp_solve`. It is possible to see this linear program with the option `-linprog`.

OPTIONS

- `-?` produces usage information.
- `-h` produces usage information.
- `-debug` produces information on the network read.
- `-determinize` replaces all timing by constants.
- `-f <file>` name of the input EGL file; if none is specified, the graph is read from the standard input.
- `-downstream` performs a "downstream tree-like transformation" of places. This is the default.
- `-upstream` performs an "upstream tree-like transformation" of places.
- `-split` performs a "linear splitting" of places.
- `-flatten` transforms the graph in another, equivalent graph, with at most one token per place.
- `-recycle` forces the creation of recycling places for all transitions.
- `-optimize` optimizes the marking of the network, with respect to the flattening.
- `-linprog` produces the linear program used by the `-optimize` option.

`-o <file>` name of the resulting EGL file; if none is specified, the result is written to the standard output.

BUGS

Unknown

NAME

eg2graph – Periodic task graph representation of an Event Graph

SYNOPSIS

`eg2graph [-dx <number] [-dy <number>] [-f <file>] [-bw] [-color] [-debug] [-o <file>] [-bn <number>] [-r <number>]`

DESCRIPTION

Creates a drawing of the (infinite) periodic task graph representation of an event graph. The graph is in FIG code.

This task graph is the data dependency graph obtained from the set of equations describing the dynamic behavior of the graph (see [1]). The columns of the graph represent the transitions, and the lines their successive firings. Arcs are drawn between two nodes if the possibility of a firing depends on the completion of an other. There is always a vertical arc between two nodes of the same column and of consecutive lines. This amounts to considering that transitions of the Event Graph are all recycled, even if it is not explicitly so (see the Bugs and the Notes sections).

The nodes are sorted in a way that arcs go always to the lower part of the graph, and such that "horizontal" arcs go from left to right. This is always possible if the Event Graph is *live*.

OPTIONS

`-?` produces usage information.

`-bw` a black and white drawing is generated (better for printing).

`-color` a color drawing is generated: this is the default.

`-debug` produces information on the network read.

`-dx <number>` the number of points (FIG units) between two columns: 50 by default.

`-dy <number>` the number of points (FIG units) between two rows: 50 by default.

`-f <file>` name of the input EGL file; if none is specified, the graph is read from the standard input.

`-h` produces usage information.

`-n <number>` the number of rows in the graph; 10 by default.

`-o <file>` name of the resulting FIG file; the standard output is the default.

`-r <number>` the radius (in FIG units) of the nodes; 5 by default.

BUGS

The arcs corresponding to recycled transitions will superpose to downwards arcs, and even nodes if these places contain more than one token.

DESIRABLE DEVELOPMENTS

The drawing of the arcs at the bottom of the graph should be more pretty.

There should be an option to disable/enable the "automatic recycling of transitions".

NAME

eg2matrix – Computes (max, +) matrices associated with Event Graphs

SYNOPSIS

```
eg2matrix [-f <file>] [-debug] [-o <file>] [-slice <int>] [-sven] [-scilab]
          [-matrix] [-vector] [-prefix <name>] [-int] [-flag_trans <int>]
```

DESCRIPTION

Generates (max, +) matrices useful for computing by recurrence the evolution of the firing times in an Event Graph (coded in EGL), as well as a vector representing initial conditions.
The Event Graph has to be "flat", that is, the maximum marking of its places has to be 1. Non-flat Event Graphs may be transformed into flat ones with `eg2eg`.
The matrix represents in some sense the longest path in a part of the developed graph associated with the Event Graph. Technically, let $Y(n)$ is the vector of the instants at which transitions *finish* firing for the n-th time. When the `-slice` option is set to h, then what is computed is the matrix A such that:

$$Y(h) \; = \; A \otimes Y(0) \; .$$

If the timings in the Event Graph are random, a sample of the evolution is generated and used for the computations. This is an alternate way for simulating the network. See `sim_eg`. The *same* sample is used at each call.
The option `-slice` specifies the height of the developed graph to analyze. It is 1 by default. If it is negative, an older, *unsafe*, algorithm is used.
The default format for the result is: the size (number of rows) of the matrix A, followed by the entries, row by row. The string `eps` means $\varepsilon = -\infty$. Then follows the initial vector as a row vector. The initial vector is suppressed with the `-matrix` option. The matrix is suppressed with the `-vector` option.
If the format is set to `-scilab`, the output is in the Scilab format, with the string `%0` to represent $-\infty$. With Scilab, names must be given to the matrix and the vector. The option `-prefix` may be used to substitute another name to the default "`A`". This character string is also inserted in the name of the initial vector. The user should take care of the correctness of the resulting name with respect to Scilab's syntax.
If the option `-int` is given, the matrix and the initial vector are output in integer format. If they are not integer, they are rounded off, and a warning message is issued.
If the format is set to `-sven`, the result is a Maple file containing the definitions of two matrices, `A0(n)` and `A1(n)` in a symbolic manner. These matrices represent the timing of transitions preceding empty places and places with one token, respectively. The symbols `X_i_n` used in this file correspond to the firing times usually denoted σ_n^i or $\sigma_i(n)$.

OPTIONS

- `-?` produces usage information.
- `-h` produces usage information.
- `-sven` produces a result in a Maple format.
- `-debug` produces information on the network read.
- `-f <file>` name of the input EGL file; if none is specified, the graph is read from the standard input.

`-int` produces integer matrices and vectors.

`-matrix` produces only the matrix.

`-o <file>` name of a file in which the result is written.

`-slice <int>` size of the "slice" of the graph to be computed; 1 by default.

`-flag_trans <int>` produces an initial vector giving a "Dirac" mass at a specified transition;

`-vector` produces only the initial vector.

BUGS
Unknown

NAME

eg2pn – EGL to PNL translation

SYNOPSIS

```
eg2pn     [-f <file>] [-debug] [-o <file>]
```

DESCRIPTION

Translates an Event Graph (coded in EGL) into a Petri Net (coded in PNL).
There is no routing information produced. It stays therefore at the default.

OPTIONS

`-?` produces usage information.

`-h` produces usage information.

`-debug` produces information on the network read.

`-f <file>` name of the input EGL file; if none is specified, the graph is read from the standard input.

`-o <file>` name of the resulting PNL file; if none is specified, the result is written to the standard output.

BUGS

Unknown

NAME
eg2qnap – EGL to QNAP translation

SYNOPSIS
`eg2qnap [-?] [-debug] [-f <file>] [-h] [-o <file>] [-O <number>]`

DESCRIPTION
Translates an Event Graph (coded in EGL) into an queueing network, (coded in the QNAP2 language).
A queue object is generated for each transition and each place, unless optimization is asked (see below). The synchronization constraints are implemented in QNAP using the FISSION/FUSION constructs. To that effect, it is necessary to use auxiliary customer classes. The number of classes is equal to the maximum input degree of transitions.
Optimization consists in trying to remove from the QNAP code unnecessary objects such as queues with 0 service, and reduce the explicit synchronization. For the time being, only places with no delay are skipped, and the optimization level is ignored.

OPTIONS

- `-?` produces usage information.
- `-h` produces usage information.
- `-debug` produces information on the network read.
- `-f <file>` name of the input EGL file; if none is specified, the graph is read from the standard input.
- `-o <file>` name of the resulting QNAP file; if none is specified, the result is written to the standard output.
- `-O <number>` swithes to the optimization level “number”.

BUGS
The initial set-up times of initial “tokens” is ignored.

DESIRABLE DEVELOPMENTS
Test the program and the resulting QNAP models.

Improve optimization.

NAME

eg2tg – EGL to TGL translation

SYNOPSIS

```
eg2tg     [-f <file>] [-debug] [-o <file>]
```

DESCRIPTION

Translates an Event Graph (coded in EGL) into a Task Graph (coded in TGL).
The translation is straightforward. Places without marks are translated into "dependency" arcs, whereas marked places are translated into "program" arcs, with the lag parameter set to the marking. Information relative to the simulation length, place timings and link names are lost.

OPTIONS

`-?` produces usage information.

`-h` produces usage information.

`-debug` produces information on the network read.

`-f <file>` name of the input EGL file; if none is specified, the graph is read from the standard input.

`-o <file>` name of the resulting TGL file; if none is specified, the result is written to the standard output.

BUGS

Unknown

DESIRABLE DEVELOPMENTS

It would be nice to generate partially developped graphs.

eg_sim – General purpose Event Graph simulator

SYNOPSIS

`eg_sim [-f <file>] [-nevt <int>] [-bs <int>] [-compile] [-keep] [-c] [-from <int>] [-to <int>] [-debug]`

DESCRIPTION

Simulates an Event Graph (coded in EGL) by iteratively computing its evolution equations.
The length of the simulation (measured in number of firings) may be controled with the option `-nevt`. It defaults to 1000.
The simulation uses a data buffer, the size of which may be controled with the option `-bs`. The total memory used is roughly the product of this size and the number of transitions in the Event Graph. It is probably desirable to keep it relatively small.
There are two possible ways to make the simulation. The first one is interpreted. It is the default. The second one consists in compiling a network-specific simulator using `eg2c`. It is chosen with the option `-compile`. With this option, the switch `-keep` allows to keep the temporary files, and the simulator, instead of removing them at the end. This method uses resources, which may be listed and checked with the option `-c`.
There is the possibility to measure response times instead of cycle times (the standard behavior). This is done with the `-from` and `-to` options. See `eg2c` for their more precise significance. This possibility exists only with the compiled simulator.

OPTIONS

`-bs <int>` sets the size of the data buffer; 1024 by default.

`-c` checks for resources.

`-compile` creates a compiled simulator and executes it.

`-from <int>` the transition from which reseponse times are measured.

`-from <int>` the transition to which reseponse times are measured.

`-debug` produces information on the network read.

`-h` produces usage information.

`-keep` keeps the source files and the simulator; the default is to clean up everything.

`-nevt <int>` the number of firings to simulate; the default is 1000.

`-?` produces usage information.

DESIRABLE DEVELOPMENTS

Should be tested more throughly, and the performance of interpreted *vs.* compiled simulators compared. In particular, the compiling of large networks seems to be painful.
There could be even more optimization: for instance not add zero times.
Does not provide trace capability.
It should be possible to perform waiting time analysis with the interpreted simulator.

NAME
eigenfig – Eigenvector representation of 3×3 $(\max, +)$ operators

SYNOPSIS

```
eigenfig  [-f <file>] [-a <file>] [-o <file>] [-fig_version <int>] [-trajectory]
          [-debug] [-h]
```

DESCRIPTION
Creates a drawing of the eigenspace and the attraction domains of a 3×3 $(\max, +)$ operator.
The operator is described in the input file, which must contain the 3×3 matrix with the following convention: the string "eps" represents $-\infty$ and numbers smaller than -10^8 are treated as $-\infty$.
Matrices of size 2×2 are also handled by being "expanded" into 3×3 matrices. See the documentation for details.
When the option -trajectory is used, the input file must also contain an initial point in dimension 3 (or 2) following the same convention. In this case, the successive iterations of the matrix over this vector will also be produced.
The result is a two dimensional figure in FIG language, which may be put into a file, or appended to an already existing one.
Two formats for the FIG file are supported: version 3.1 (the default) and version 2.1, with the option -fig_version 2.

OPTIONS

- -a <file> appends the result to file <file>; the standard output is the default;
- -f <file> reads the matrix in <file>; if not specified, the matrix is read from the standard input.
- -o <file> writes the result in file <file>.
- -fig_version specifies the version of the FIG language (2 or 3). The default is 3.
- -trajectory this options produces the list of the successive iterations of the matrix on an initial vector which must be given in the input file.
- -debug produces information about the computation.
- -h produces a help message.

BUGS
None reported.

DESIRABLE DEVELOPMENTS
There should be an output for XFIG 3.2.

NAME
ers – A graphical interface for Petri Nets

SYNOPSIS
`ers [<file>] [-c] [-h]`

DESCRIPTION
It allows to edit Petri Nets, check their topological properties, generate graphical representation code, and simulation codes [10]. For more information on the functionalities of this interface, see section 3.6 of the documentation.
`ers` is written with the Graphmod package. The latest version of Graphmod allow copy/paste, zoom and other standard editing features.

OPTIONS

`-c` only checks for the presence of resources and reports its findings.

`<file>` the (optional) name of a GSS file containing the model.

`[-h`] produces usage information.

BUGS
Lots.

DESIRABLE DEVELOPMENTS
Lots too.

NOTES
A certain number of environment variables should be properly configurated. If something goes wrong, ask your local GSS/Graphmod specialist about it.

fifo_eg – Test that an Event Graph is FIFO

SYNOPSIS

`fifo_eg   [-d] [-lazy] [-f <file>] [-o <file>] [-h] [-?]`

DESCRIPTION

Tests if an Event Graph (coded in EGL) is fifo. A Petri net is said to be FIFO if tokens exit places in the order in which they entered it, and if firings complete in the same order than they began.
The fact that an Event Graph is FIFO is necessary for the existence of the $(\max, +)$ evolution equations, used in `eg2c`, `eg_sim`, `eg2matrix` for instance.
A necessary condition for the network to be FIFO is that

- all transitions have constant firing times *or* are recycled (with a place with one token);
- all places have constant holding times.

The program checks if this condition is satisfied. If so, it exits and outputs nothing. If not, it calls `fifo_pn`, after translating the network into PNL. The result of the analysis is then that of this program.
When the option `-d` is selected, the result is a list of some non-fifo transitions and some non-fifo places. The list is complete, unless the `-lazy` option has been selected. Both lists are in DL (Data Language) syntax. They consist in sequences of integers, preceded by an empty line, and separated by the: `_E_O_E` keyword. The first of these numbers indicates the length of the corresponding list, and the remaining the identificators of the transitions (or places) which are not FIFO.
The output can be redirected into a file using the `-o` option.

OPTIONS

`-?`	produces usage information;
`-d`	dumps lists of non-fifo objects.
`-lazy`	performs an incomplete analysis.
`-f <file>`	name of the input EGL file; if none is specified, the graph is read from the standard input;
`-h`	produces usage information;
`-o <file>`	name of the file for the result of the analysis; the default is the standard output.

BUGS

Unknown.

DESIRABLE DEVELOPMENTS

The output should be consistent, whether the fast analysis succeeds or not.

In the output, places should be described as pairs of transitions, and not by their number, which is not in the EGL.

fifo_pn – Test that an Petri Net is FIFO

SYNOPSIS

`fifo_pn [-d] [-lazy] [-f <file>] [-o <file>] [-h] [-?]`

DESCRIPTION

Tests if an Petri Net (coded in PNL) is fifo. A Petri net is said to be FIFO if tokens exit places in the order in which they entered it, and if firings complete in the same order than they began.
For the time being, this notion applies only to Event Graphs.
The fact that an Petri Net is FIFO is necessary for the existence of the $(\max, +)$ evolution equations, used in `eg2c`, `eg2magmas`, `eg_sim`, `eg2matrice` for instance.
A necessary and (almost) sufficient condition for the network to be FIFO is that

- all transitions have constant firing times *or* have an input place which marking is bounded by 1;
- all places have constant holding times *or* have a marking bounded by 1.

A transition if FIFO if it is recycled. The program checks if a simple sufficient condition is satisfied. If not it checks the necessary condition, calling `pn_bound` for obtaining the bounds on the marking. When the option `-d` is selected, the result is a list of some non-fifo transitions and some non-fifo places. The list is complete, unless the `-lazy` option has been selected. Both lists are in DL (Data Language) syntax. They consist in sequences of integers, preceded by an empty line, and separated by the: `_E_O_E` keyword. The first of these numbers indicates the length of the corresponding list (possibly 0), and the remaining the identificators of the transitions (or places) which are not FIFO. The output can be redirected into a file using the `-o` option.

OPTIONS

`-?`	produces usage information;
`-d`	dumps lists of non-fifo objects.
`-lazy`	performs an incomplete analysis.
`-f <file>`	name of the input PNL file; if none is specified, the graph is read from the standard input;
`-h`	produces usage information;
`-o <file>`	name of the file for the result of the analysis; the default is the standard output.

BUGS

Unknown, yet.

gen_eg – Generator of Event Graphs

SYNOPSIS

```
gen_eg    [-t <type>] [-s <size(s)>] [-i <init>] [-l <sim length>] [-tl <law>]
          [-pl <law>] [-o <file>] [-h]
```

DESCRIPTION

Generates Event Graphs (in the EGL language) according to specified shapes, sizes and other parameters.
The type of the network is controlled by the `-t` option. Possible values for `<type>` are:

- `ser_closed` for a closed series network (a cycle): this is the default,
- `ser_open` for an open series network (a line),
- `double` for a double cycle connected by a bridge,
- `job_shop` for a job-shop structure.

The size of the network, is specified with `-s`. For series networks, the size is the number of transitions. For double loops and job-shops, two sizes must be specified. For a double loop, they represent the size of the two cycles. For a job-shop, they are the number of lines and columns, respectively.
The timings for places and transitions are controlled with `-pl` and `-tl` respectively. The syntax it the same as in the EGL and PNL languages. For instance, specifying a constant duration of 2.0 units is done by: `C 0.0`. The default for both is Exponential(1.0), that is: `E 1.0`. See also the information obtained with the `-h` option.
The initial marking of the places is controlled by the `-m` option, to specify the number of initial tokens, and `-i` to specify the initial location. Possible modes are:

- `full` (the default): tokens in every places,
- `empty` no tokens,
- `first` tokens in the first place of series, and in the upper left corner of job_shops,
- `even` tokens in even numbered places,
- `diagonal` tokens in the diagonal of job-shops,
- `checker` tokens as in a checker board for job-shops.
- `column` tokens in the first column, for job-shops.

OPTIONS

`-t`	specifies the type of network to generate; ser_closed by default.
`-s`	specifies the size(s) of the network; 2 by default for each dimension.
`-i`	specifies the initial marking mode; full by default.
`-tl`	firing law for transitions; exponential with mean 1 by default.
`-pl`	holding times law for places; exponential with mean 1 by default.

`-h` produces usage information.

`-?` produces usage information.

`-debug` produces information on the network read.

BUGS

There are probably quite a few ones.

DESIRABLE DEVELOPMENTS

No way to specify initial marking, as in `gen_fc`.

Other types of networks can be imagined.

gen_fc – Generation of Free Choice Nets

SYNOPSIS

```
gen_fc    [-t <type>] [-s <size(s)>] [-i <init>] [-l <sim length>] [-tl <law>]
          [-pl <law>] [-o <file>] [-h]
```

DESCRIPTION

Generates Petri Nets (in the PNL language) according to specified shapes, sizes and other parameters.
The type of the network is controlled by the `-t` option. Possible values for `<type>` are:

- `ser_closed` for a closed series network (a cycle): this is the default,
- `ser_open` for an open series network (a line),
- `double` for a double cycle connected by a bridge,
- `job_shop` for a job-shop structure.

The size of the network, is specified with `-s`. For series networks, the size is the number of transitions. For double loops and job-shops, two sizes must be specified. For a double loop, they represent the size of the two cycles. For a job-shop, they are the number of lines and columns, respectively.
The timings for places and transitions are controlled with `-pl` and `-tl` respectively. The syntax it the same as in the EGL and PNL languages. For instance, specifying a constant duration of 2.0 units is done by: `C 0.0`. The default for both is Exponential(1.0), that is: `E 1.0`. See also the information obtained with the `-h` option.
The initial marking of the places is controlled by the `-m` option, to specify the number of initial tokens, and `-i` to specify the initial location. Possible modes are:

- `full` (the default): tokens in every places,
- `empty` no tokens,
- `first` tokens in the first place of series, and in the upper left corner of job_shops,
- `even` tokens in even numbered places,
- `diagonal` tokens in the diagonal of job-shops,
- `checker` tokens as in a checker board for job-shops.
- `column` tokens in the first column, for job-shops.

OPTIONS

`-t`	specifies the type of network to generate; ser_closed by default.
`-s`	specifies the size(s) of the network; 2 by default for each dimension.
`-i`	specifies the initial marking mode; full by default.
`-m`	initial marking; 1 by default.
`-tl`	firing law for transitions; exponential with mean 1 by default.

`-pl`	holding times law for places; exponential with mean 1 by default.
`-h`	produces usage information.
`-?`	produces usage information.
`-debug`	produces information on the network read.

BUGS

There are probably quite a few ones.

DESIRABLE DEVELOPMENTS

There should be a way to specify the routing, which is `Random` by default.

Other types of networks can be imagined.

gen_iid – Generator of samples of i.i.d. sequences

SYNOPSIS

`gen_iid [-o <file>] [-law <law>] [-nbsamples <int>] [-debug] [-h] [-?]`

DESCRIPTION

Generates a list of numbers which are drawn from a given distribution. The law is specified with an identifying character followed by a list of parameters. For instance, `E 1.0` means an exponential distribution with mean 1.0. This is the default distribution.

The following table lists the currently known distribution, the syntax of the specification and the interpretation of the parameters. When needed, $\overline{F}(x)$ denotes the complementary distribution function $\mathbf{P}(X > x)$. The value of the mean value is also recalled. See also the notes below the table.

Id.	Name	Syntax	Interpretation Parameters	Mean
`C`	Constant	`C v`	v is the value	v
`U`	Uniform	`U a b`	$[a, b]$ is the interval	$\frac{b-a}{2}$
`E`	Exponential	`E m`	m is the mean	m
`H`	Hyperexponential	see note 1		
`R`	Erlang	`R n m`	n: nb. of phases, m: mean	m
`B`	Bernoulli	`B p`	p is the proba. of 1	p
`g`	Geometric[2] on $\{0, 1, \dots\}$	`G p`	p is the proba. of 0	$\frac{p}{1-p}$
`B`	Geometric[2] on $\{1, 2, \dots\}$	`g p`	p is the proba. of 1	$\frac{1}{1-p}$
`W`	Weibull	`W a` ν	$\overline{F}(x) = exp(-ax^{\nu})$	$\nu^{-1}\Gamma(\nu^{-1})a^{-1/\nu}$
`N`	Normal	`N m v`	m: mean, v: std. dev.	m
`L`	Log-Normal	`L m v`	m: mean, v: std. dev.	m
`p`	Pareto	`p a n`	$\overline{F}(x) = \left(\frac{a}{a+x}\right)^n$	$\frac{a}{n-1}\ (n > 1)$
`D`	Discrete	see note 2	v_i values, p_i probas	$\sum_i p_i v_i$
`O`	On/Off	see note 3		
`P`	IPP.	see note 4		

Special notes:

1. The hyperexponential distribution is a random mixture of exponential distributions. The syntax is:

   ```
   H p1 m1 p2 m2 ...
   ```

   where the $p_i s$ are probabilities (summing up to one), and the m_i the means of the exponential distributions. The cdf is then $\overline{F}(x) = \sum_i p_i e^{-x/m_i}$, and the mean: $\sum_i p_i m_i$.

2. There are two commonly used geometric distributions. They differ from their starting point: 0 or 1. The parameter being p, one has: $\mathbf{P}(\mathtt{G} = k) = p^k(1-p), p \geq 0$ while $\mathbf{P}(\mathtt{g} = k) = p^{k-1}(1-p), p \geq 1$.

3. The general discrete distribution has the followin syntax:

```
D [ v1 v2 ...  vn ] [ p1 p2 ...  pn ]
```

   where the $v_i s$ are the values, and pi_s are the probabilities. Both groups must have the same number of elements. The probabilities should add up to one. The whitespaces around the square brackets "`[`" and "`]`" are customary.

4. On/Off is an On/Off source. This is not implemented in `gen_iid` yet.

5. IPP is an Interrupted Poisson Process. This is not implemented in `gen_iid` yet.

OPTIONS

`-law <law>` specifies the according to which the samples are drawn. The default is Exponential with mean 1.0.

`-nbsamples <int>` specifies the number of samples to draw. 100000 by default.

`-debug` adds a line beginning with "`#`" recalling the law used and the number of samples.

`-o <file>` sends the result in the named file.

`-h` produces usage information.

`-?` produces usage information.

BUGS

Unknown.

gen_gr – Generator of interaction graphs

SYNOPSIS

`gen_gr [-t <type>] [-s <size>] [-reverse] [-o <file>] [-h]`

DESCRIPTION

Generates an *exclusion graph* according to the convention of Brilman 96. It is a directed graph $(\mathcal{V}, \mathcal{E})$ with weights p_i on the vertices, which are interpreted as probabilities after normalization.
The file produced may be in two matrix formats. The *sparse matrix* format (the default) is the following:

```
sp
N
p1 ... pN
i1 j1
i2 j2
...
ip jp
```

The `sp` keyword indicates that this is a sparse matrix. The integer number `N` is the number of nodes. The numbers `p1, ..., pN` are the weights of the nodes. The following pairs of numbers are the edges (i, j) of the graph. The matrix ends with the end of file, or anyting not a number.
The full matrix format (option `-full`) is as follows:

```
N
p1 ... pN
a11 a12 ... a1N
a22 a21 ... a2N
...
aN1 aN2 ... aNN
```

The number `N` and the numbers `p1, ..., pN` are as above. The following numbers describe the adjacency matrix: `aij` is 1 if there is an edge from node `i` to node `j`, and 0 otherwise.
The graph generated will typically be "simulated" using `gr_sim` or analyzed with `gr_bound`.
The type of graph generated is controlled by the `-t` option. Possible values for `<type>` are:

- `cycle` for a cycle: this is the default,
- `line` for a line,
- `star` for a star,
- `complete` for a complete graph,
- `komplete` for a (almost) complete bipartite graph,
- `random` and `Random` for a random graph.

The size of the network (number of vertices), is specified with `-s`. For complete bipartite graphs, two sizes are needed.
If the option `-reverse` is specified, the graph generated is the complemented one.
The difference between `random` and `Random` is that the second form changes the seed of the pseudo-random generator and results in a (possibly) different graph each time. The edge probability in both forms is given by the `-p` option. It is 0.5 by default.
In all graphs, *including* random and complete bipartite, the arcs (i, i) are always present, unless the `-reverse` option is used. The generation of these loops can be prevented using the option `-noloop`. This option does not affect complete graphs.

OPTIONS

`-t`	specifies the type of network to generate; `cycle` by default.
`-s`	specifies the size(s) of the network; 2 by default.
`-reverse`	causes `gen_gr` to create the complement graph.
`-p`	probability of an edge for random graphs.
`-full`	use a full matrix format for the output.
`-noloop`	prevents the creation of arcs (i, i).
`-o <file>`	sends the result in the named file.
`-h`	produces usage information.
`-?`	produces usage information.

BUGS
Unknown.

DESIRABLE DEVELOPMENTS
Other types of networks can be imagined. For instance, cycles with chords such as produced by `gred`.

gr_bound – Computing bounds for interaction graphs

SYNOPSIS

`gr_bound  [-f <file>] [-o <file>] [-epsilon <real>] [-debug] [-h] [-?]`

DESCRIPTION

Computes bound on the Lyapunov exponent (cycle time) of a $(\max, +)$ system corresponding to an *exclusion graph* according to the convention of Brilman 96. See `gr_sim` and `gen_gr` for the description of interaction graphs and the format of the input file.
The bounds computed are:

- upper bound: for uniform graphs (equal weights), the bound of Theorem 10, using the "f" function and the spectral radius of the graph; for general graphs, the bound of Corollary 3.6.2.
- lower bound: for uniform graphs, the bound of Theorem 11 based on the average degree; for general graphs, the bound of Corollary 3.6.3.

The result consists in the pair or reals (lower bound, upper bound), on the same line. It is written to the standard output, or to a file is the option `-o` is used.
The computation of the spectral radius and of the root of the f function is based on numerical algorithms. The precision of these computations may be controlled by the parameter `-epsilon`, which defaults to 1e-6.

OPTIONS

`-f <file>`	name of the file containing the graph.
`-o <file>`	sends the result in the named file.
`-epsilon`	precision of numerical computations. 1e-6 by default.
`-h`	produces usage information.
`-help`	produces usage information.
`-?`	produces usage information.

BUGS

Unknown.

DESIRABLE DEVELOPMENTS

The lower bound based on the chromatic number (or a lower bound of it) should be available.

Bounds for special classes of graphs (*e.g.* stars) seem possible.

gr_sim – Simulator of (max, +) interaction graphs

SYNOPSIS

```
gr_sim    [-f inputfile] [-o outputfile] [-nevt int] [-compile] [-keep] [-h]
          [-help] [-?]
```

DESCRIPTION

Simulates the evolution of a (max, +) system corresponding to an *exclusion graph* according to the convention of Brilman 96. An *exclusion graph* is a directed graph $(\mathcal{V}, \mathcal{E})$ with weights p_i on the vertices, which are interpreted as probabilities after normalization. The graph is read in the file specified by the option `-f`, or else in the standard input. See `gen_gr` for the format of this file.

The evolution of the system is according to the following rules. At each time instant n, a node is selected according to the probabilities p_i. If node i is selected, its "height" $X_i(n)$ becomes:

$$X_i(n) \ = \ 1 \ + \max_{j|(j,i)\in\mathcal{E}} \{X_j(n-1)\} \ .$$

If there is no edge (j, i) in the graph, the result is ϵ. The program computes $X_i(N)$ for a given N and all $i \in \mathcal{V}$. It returns the estimator for the cycle time of each nodes: $\hat{\gamma}_i = X_i(N)/N$. The value of N may be specified with the option `-nevt`. It defaults to 100000.

The result is written to the standard output, or a file is the option `-o` is used.

By default, the simulator performs (max, +) matrix/vector multiplications in a loop. A faster alternative is to generate a compiled program and run it. This is done by using the option `-compile`. This makes use of the `gr2c` translator and temporary files `/tmp/gr_sim.c` and `/tmp/gr_sim`. The option `-keep` prevents the destruction of these files, so that they can be inspected, modified or reused.

OPTIONS

- `-f <file>` name of the file containing the graph.
- `-o <file>` sends the result in the named file.
- `-compile` uses a compiled program to perform the simulation.
- `-nevt` number of time steps to simulate. The default is 10000.
- `-h` produces usage information.
- `-help` produces usage information.
- `-?` produces usage information.

BUGS

Unknown.

DESIRABLE DEVELOPMENTS

Other performance measures could be computed.

It should be possible to control where the temporary files are located.

NAME

gr2c – Conversion of a (max, +) interaction graph into a C program

SYNOPSIS

`gr2c [-f <file>] [-o <file>] [-nevt int] [-h] [-?]`

DESCRIPTION

Translates the description of a (max, +) *exclusion graph* into a C program to simulate it. See `gen_gr` and `gr_sim` for details on exclusion graphs and their simulation.
The graph is read in the file specified by the option `-f`, or else in the standard input. See `gen_gr` for the format of this file.
The result is written to the standard output, or a file if the option `-o` is used.
After compilation, the resulting program will recognize the command line option `-nevt` which allows to change the length of the simulation. The default value is the one specified to `gr2c` with its own `-nevt` option. This is 100000 by default.

OPTIONS

- `-f <file>` name of the file containing the graph.
- `-o <file>` sends the result in the named file.
- `-nevt` number of time steps to simulate. The default is 10000.
- `-h` produces usage information.
- `-help` produces usage information.
- `-?` produces usage information.

BUGS

Unknown.

NAME

gr2mat – Conversion of a (max, +) interaction graph into a Bernoulli product of $(max, +)$ matrices

SYNOPSIS

`gr2mat` `gr2mat [-f <file>] [-o <file>] [-full] [-scilab] [-h] [-help] [-?]`

DESCRIPTION

Translates the description of a (max, +) *exclusion graph* into a set of $(max, +)$ matrices.
The graph is read in the file specified by the option `-f`, or else in the standard input. See `gen_gr` for the format of this file.
The result is written to the standard output, or a file if the option `-o` is used. See `mat_sim` for the format of this file. By default, matrices are written using the *sparse* format. This behavior may be modified with the `-full` option, or with the `-scilab` option.

OPTIONS

- `-f <file>` name of the file containing the graph.
- `-o <file>` sends the result in the named file.
- `-full` produces matrices in full format.
- `-scilab` produces matrices in the scilab format.
- `-h` produces usage information.
- `-?` produces usage information.

BUGS

Unknown.

NAME

gr2tetris – Conversion of a $(\max, +)$ interaction graph into a heap of pieces

SYNOPSIS

```
gr2tetris gr2tetris [-f <file>] [-o <file>] [-maxsize <int>] [-optimize]
          [-greedy] [-h] [-?]
```

DESCRIPTION

Translates the description of a $(\max, +)$ *exclusion graph* into a set of "Tetris pieces".
The graph is read in the file specified by the option `-f`, or else in the standard input. See `gen_gr` for the format of this file.
The result is written to the standard output, or a file if the option `-o` is used.
There are several sets of pieces corresponding to an exclusion graph. All of them have the same number of pieces, which is the number of nodes in the graph. The number of columns and the shape of the pieces depends on the algorithm used.
All algorithms look for *cliques* in the graph. The maximal size of the cliques is specified by the option `-maxsize`. The larger this parameter, the longer the computation time, but (hopefully) the smaller the number of columns.
If `-maxsize` is 2 (the default), the cliques are the edges of the graph, and there are as many columns as edges.
If `-maxsize` is larger, then all cliques of this size are computed, and the unused edges are added to them. If the option `-greedy` is used, then edges are assigned to one clique only.
If the option `-optimize` is used (not implemented yet), the optimal number of columns is looked for. This may take some time for large graphs...

OPTIONS

- `-f <file>` name of the file containing the graph.
- `-o <file>` sends the result in the named file.
- `-maxsize` size of the largest cliques searched. The default is 2.
- `-greedy` causes the algorithm never to reuse an edge.
- `-optimize` looks for the smallest possible number of columns.
- `-h` produces usage information.
- `-?` produces usage information.

BUGS

Unknown.

NAME
gred – A graphical interface for Interaction Graphs

SYNOPSIS
`gred [-matrix] [<file>] [-ns <int>] [-numbers] [-nolog] [-debug] [-?]`

DESCRIPTION
It allows to edit task graphs and do various operations, including the generation of graphical representation code and simulation codes. For more information on the functionalities of this interface, see section 3.3.

OPTIONS

- `<file>` the (optional) name of a file containing the model.
- `-matrix` constructs an initial graph from the adjacency matrix read from the given file.
- `-ns <int>` sets the node size. The default is 8.
- `-numbers` causes internal node numbers to appear in the display of the graph.
- `-nolog` sets the logging style to none, thus preventing the launch of a Logger.
- `-debug` turns on a verbose debugging mode.
- `-?` produces usage information.

BUGS
Lots

DESIRABLE DEVELOPMENTS
Lots too

NAME

laplace – Empirical Laplace transform construction

SYNOPSIS

```
laplace   [-f <file>] [-o <file>] [-s <val>] [-sfrom <real>] [-sto <real>]
          [-sstep <real>] [-debug] [-h] [-?]
```

DESCRIPTION

Reads a list of numbers and builds the empirical Laplace transform associated to it. The result is a list of points suited for plotting.
By default, numbers are taken from the standard input and written on the standard output.
It is possible to control (with the `-sfrom`, `-sto` and `-sstep` options) the range on which the function is constructed, and the step of the grid. By default, this range is $[0, 10]$ with step 0.1.
The function constructed is as follows: if there are N values read, $F^*(s)$ is the sum of the $\exp(-sX_i)$, divided by N.
The result is the sequence of values of F at grid points points.
The `debug` options produces extra information on the values read.

OPTIONS

- `-?` produces usage information.
- `-debug` produces information on the values read.
- `-f <file>` name of the input EGL file; if none is specified, the graph is read from the standard input.
- `-h` produces usage information.
- `-o <file>` name of the resulting file; the standard output is the default.
- `-sfrom <real>` sets the lower end of the range to analyze. 0 by default .
- `-sstep <real>` sets the grid step. 0.1 by default.
- `-sto <real>` sets the lower end of the range to analyze. 10 by default.
- `-sval <real>` computes the function at a single point.

BUGS

Unknown yet.

live_eg – Test the liveness of an Event Graph

SYNOPSIS

`live_eg   [-d] [-f <file>] [-o <file>] [-h] [-?]`

DESCRIPTION

Tests if an Event Graph (coded in EGL) is live, by checking that all circuits are nonempty. Il the network is not live, exhibits (on demand) an empty circuit, in a format suited for the `ers` editor (see p. 49).
The format of the resulting cycle follows the DL (Data Language) syntax. It is a sequence of integers, preceded by an empty line, and separated by the: `_E_O_E` keyword. The first of these numbers indicates the length of the cycle, and the remaining the identificator of the transitions through which the circuit runs. These identificators are that of the EGL file.

OPTIONS

- `-?` produces usage information;
- `-d` dumps an empty circuit if the network is not live.
- `-f <file>` name of the input EGL file; if none is specified, the graph is read from the standard input;
- `-h` produces usage information;
- `-o <file>` name of the file containing the resulting cycle (when applicable); the default is the standard output.

BUGS

Unknown.

m_eg – Waiting time distribution in Poisson driven deterministic Event Graphs

SYNOPSIS

`m_eg` `[-f <file>] [-o <file>] -lambda <real> -trans_nb <int> -input_nb <int> -event_nb <int> [-xfrom <real>] [-xto <real>] [-xval <real>] [-xstep <real>] [-queue] [-log] [-moment <int>] [-stationary] [-debug] [-h] [-?]`

DESCRIPTION

Computes the distribution function of the "waiting time" of events in an Event Graph with deterministic times, fed by a single Poisson source of tokens. The waiting time of the n-th event at transition k is defined as the difference between the start of the n-th firing of this transition and the time at which the n-th token was released by the source. In some nets, this may be negative.
The Event Graph is described in an EGL file, *except* for the source. The source is described using the command line options: `-lambda` for its intensity, and `-input_nb` for the number of the transition (according to the numbering of the input file) to which it is connected.
The transition to be measured is given by the `-trans_nb` parameter. The option `-event_nb` specifies the number of the customer of which the response time is to be computed. If `-1` is specified, or if the switch `-stationary` is used, the stationary distribution is computed.
Unless moments are required (with the `-moment` option), lhe result is a list of points, representing $P(W_N \leq x)$, the response time distribution of the N-th customer. It is possible to control (with the `-xfrom`, `-xto` and `-xstep` options) the range on which the function is constructed, and the step of the grid. By default, this range is $[0, 10]$ with step 0.1. If the option `-xval` is used, the distribution is computed at a single point.
By default, numbers are written on the standard output. The output may be redirected to a file with the option `-o`.
It is also possible to compute the function $P(W > x)$ (option `-queue`) or the function $\log P(W > x)$ (with in addition the option `-log`).
The `debug` options produces extra information on the values read.
This application uses as resources the programs `eg2eg` (for flattening if necessary), `eg2matrix` and `scscycInt` for $(\max, +)$ analysis, and `mk1` for the response time analysis. All these four applications have to be installed and coherent. This may be checked with the `-c` option.

OPTIONS

- `-?` produces usage information.
- `-debug` produces information on the values read.
- `-h` produces usage information.
- `-c` checks for the presence of necessary resources.
- `-o <file>` name of the resulting file; the standard output is the default.
- `-xfrom <real>` sets the lower end of the range to analyze. 0 by default .
- `-xstep <real>` sets the grid step. 0.1 by default.
- `-xto <real>` sets the upper end of the range to analyze. 10.0 by default.

-xval <real> computes the function at a single point.

-lambda <real> the arrival rate of customers. Must be present and > 0.

-trans_nb number of the transition for which to compute the distribution.

-input_nb number of the transition to which the source is connected.

-event_nb number of the event of which to compute the response time.

-stationary computes the stationary distribution.

-moment <int> computes the moment of order <int>.

-queue computes the queue of the distribution. The default is to compute the cumulative distribution.

-log computes the log of the queue.

BUGS

The results have not been checked yet. In particular, they are known to be wrong if there are no paths without tokens between the input (source) and the transition measured.
The correctness of the Event Graph is not checked. If it is not correct... who knows what may happen.

DESIRABLE DEVELOPMENTS

The program suffers from the same limitations than mk1 concerning the possibility of performing stationary analysis when the cyclicity is not 1, and computing moments of order 2 and above.

mat_bound – Computes bounds for random products of (max, +) matrices

SYNOPSIS

```
mat_bound [-f inputfile] [-o outputfile] [-nmax int] [-exact] [-numeric]
          [-pval <real>] [-pfrom <real>] [-pto <real>] [-pstep <real>] [-maple]
          [-debug] [-h] [-help] [-?]
```

DESCRIPTION

Computes bounds on the expectation of the transients, or on the Lyapunov exponent of a system given by the random product of matrices in the (max, +) algebra. The elements of the product are chosen among a finite number P of matrices $N \times N$ according to some probabilities p_i, $1 \leq i \leq P$. The name of the file containing the matrices is given by the option `-f`. By default, the data is read on the standard input.
The format for the input data, and the rules of evolution of the system is described in the manual page for `mat_sim`.

Two types of bounds are possible.

- The first one is a *lower bound* on the value of $EX_i(N)/N$, for all i and a given N. The time complexity of this algorithm is $\mathcal{O}(N^3)$. It is applicable only to random products with two matrices. Note that none of these bounds, nor their minimum, is necessarily a lower bound on the Lyapunov exponent.
- The second one is a pair of lower and upper bounds for the Lyapunov exponent of the system. The bounds are respectively $E(\min_i X_i(N))$ and $E(\max_i X_i(N))$. They are computed by brute force enumeration. The complexity of this method is $\mathcal{O}(P^N)$.

For both methods, the value of N may be specified with the option `-nmax`. It defaults to 10.
For the first method, the result may be given either numerically (option `-numeric`) or symbolically, as a polynomial of the variable `p`. In the first case, a range of values for the probability p of the matrix A_0 may be specified (options `-pfrom`, `-pto`, `-pstep` or `-pval`). The result is a data file in the `gnuplot` format. In the second case, the output format may be specified for `gnuplot` (the default) or `Maple (TM)`. This is controlled by the option `-style`. The formal result is the default one.
For the second method, the probability vector used is the one given in the input file, and the result is a couple of points (min, max).
The result is written to the standard output, or a file is the option `-o` is used.

OPTIONS

`-f <file>` name of the file containing the matrices.

`-o <file>` sends the result in the named file.

`-nmax` number of time steps to use in the algorithms. The default is 10.

`-exact` uses the exact computation of $EX(N)$ to derive bounds on the Lyapunov exponent.

`-numeric` computes numerical values and not a formal polynomial.

`-pval` starting value for the range of probabilities. Implies `-numeric`. Default value is 0.0.

`-pto` stopping value for the range of probabilities. Implies `-numeric`. Default value is 1.0.

`-pstep` step for the increments in the range of probabilities. Implies `-numeric`. Default value is 0.05.

`-pval` specifies a unique probability value. Implies `-numeric`. Default is to compute a whole range.

`-maple` uses a `Maple` style to write the polynomials. The default is a `gnuplot` style.

`-h` produces usage information.

`-help` produces usage information.

`-?` produces usage information.

BUGS

When plotting the formal polynomials with `gnuplot`, numerical errors appear when N is large.

This is due to the presence of large binomial coefficients. The numerical algorithm uses a safe numerical method.

DESIRABLE DEVELOPMENTS

A better handling of memory would be useful.

Using sparse matrices instead of full ones would also be nice.

mat_sim – Simulator of random products of (max, +) matrices

SYNOPSIS

```
mat_sim   [-f <file>] [-o <file>] [-seq <file>] [-nevt int] [-nrep <int>]
          [-stats] [-gradient] [-dir <string>] [-compile] [-keep] [-h] [-help]
          [-?]
```

DESCRIPTION

Simulates the evolution of a system given by a product of matrices in the (max, +) algebra. The elements of the product are chosen among a finite number P of matrices $N \times N$ according to:

- either some probabilities p_i, $1 \le i \le P$,
- or a given sequence $i_1, i_2, \ldots, i_n$.

In the first case, the evolution of the system is according to the following rules. At each time instant n, a matrix is selected according to the probabilities p_i. If matrix A_i is selected, then the (column) vector $X(n)$ takes the value:

$$X(n) \;=\; A_i \;\otimes\; X(n-1) \; .$$

The starting value is $X(0) = (0, \ldots, 0)'$. The program computes $X_i(N)$ for a given N and all i. It returns the estimator for the cycle time of each coordinate: $\hat{\gamma}_i = X_i(N)/N$. The value of N may be specified with the option `-nevt`. It defaults to 100000.
In the second case, a sequence of matrix numbers is given in a file, through the option `-seq`. The file should contain a list of numbers $i_1, i_2, \ldots, i_n$ contained between 1 and P. The result is the matrix:

$$A(n) \;=\; A_{i_n} \;\otimes\; A_{i_n-1} \;\otimes\; \ldots \;\otimes\; A_{i_1} .$$

The format of the output is suited for compatibility with `scscyc`.
In any case, the result is written to the standard output, or a file is the option `-o` is used.
Input Format The problem is described in a file, according to the following format:

```
N
P
p1 ...  pP
matrix1
matrix2
...
matrix P
```

The numbers `p1, ..., pP` are the respective weights of the P matrices, which will be normalized into probabilities. The matrices are specified either in *Full format:*:

```
A1-11 A1-12 ...  A1-1N
A1-22 A1-21 ...  A1-2N
...
A1-N1 A1-N2 ...  A1-NN
```

of in the *Sparse format:*

```
sp
i1 j1 v1
i2 j2 v2
...
iq jq vq
end
```

In the first case, exactly N^2 entries must be specified. Entries may be a real number or the string `eps`, representing $-\infty$. In the second case, the matrix is a list of triples `(i,j,v)`, meaning that $A_{i,j} = v$. The values `v` must be real numbers. The numbering of the indices `i,j` starts with 0. Both descriptions may be mixed. Blank lines are allowed to separate matrices.
The name of the file containing the matrices is given by the option `-f`. By default, the data is read on the standard input.
Statistics Statistics may be collected on the estimator for the cycle time by using the `-stats` option. Confidence intervals (95 %) are computed using the replication method. The number of replicates is specified with the `-nrep` option. The default number is 1 (no statistics).
Sensitivity Analysis It is possible to perform sensitivity analysis with respect to the probability vector, in the following sense. Let q be a vector with P entries such that $\sum_i q_i = 0$. Let $\hat{\gamma}_i(p)$ denote $EX_i(N)/N$ when matrices are randomly drawn according to the probability vector p. It is possible to obtain an estimate for:

$$\lim_{\theta \to 0} \frac{1}{\theta} \left(\hat{\gamma}_i(p + \theta q) - \hat{\gamma}_i(p) \right) .$$

Sensitivity analysis is asked with the `-gradient` option. The direction of the perturbation, q, is specified by the option `-dir` followed by a string describing q in the canonical basis `p1`, `p2`, ..., `pP`. For instance `2p1-p2-p3` denotes the vector $q = (2, -1, -1)$ and `-p1-p2+2.0p4` represents $q = (-1, -1, 0, 2)$. By default, the vector q is $(1, -1, 0, \dots)$. The simulator implements the algorithm of [11].
Compilation By default, the simulator performs $(\max, +)$ matrix/vector multiplications in a loop. A faster alternative is to generate a compiled program and run it. This is done by using the option `-compile`. This makes use of the `mat2c` translator and temporary files `/tmp/mat_sim.c` and `/tmp/mat_sim`. The option `-keep` prevents the destruction of these files, so that they can be inspected, modified or reused.

OPTIONS

- `-f <file>` name of the file containing the matrices.
- `-o <file>` sends the result in the named file.
- `-seq <file>` reads a sequence from the named file.
- `-compile` uses a compiled program to perform the simulation.
- `-nevt` number of time steps to simulate. The default is 10000.
- `-nrep` number of replications to simulate. The default is 1.

`-stats` computes statistics (confidence intervals).

`-gradient` computes gradient estimates.

`-dir` specifies the direction in which to compute the gradient. The default is (1,-1).

`-keep` keeps the temporary C files used when compiling. The default is to remove them.

`-h` produces usage information.

`-help` produces usage information.

`-?` produces usage information.

BUGS
Compilation is not compatible with gradient analysis at the moment.

DESIRABLE DEVELOPMENTS
Other performance measures could be computed.

It should be possible to control where the temporary files are located.
It should be possible to control the confidence level of confidence intervals. It should be possible to specify a precision instead of a fixed number of replications.

NAME

mat2c – Make a C program for the simulation of random products in $(\max,+)$

SYNOPSIS

`mat2c [-f <file>] [-o <file>] [-nevt <int>] [-nrep <int>] [-gradient] [-h] [-?]`

DESCRIPTION

Creates a C program to simulate the random products of matrices in the $(\max,+)$ algebra. See `mat_sim` for details on the description format and the principle of the simulation.
The problem is read in the file specified by the option `-f`, or else in the standard input.
The result is written to the standard output, or a file is the option `-o` is used.
After compilation, the resulting program will recognize the following command line option:

`-nevt` which allows to change the length of the simulation. The default value is the one specified to `mat2c` with its own `-nevt` option. This is 100000 by default.

`-nrep` which allows to change the number of replications of the simulation. The default value is the one specified to `mat2c` with its own `-nrep` option. This is 1 by default.

`-stats` which instructs the program to actually compute statistics. By default, the program does not write out confidence intervals.

OPTIONS

`-f <file>`	name of the file containing the graph.
`-o <file>`	sends the result in the named file.
`-nevt`	number of time steps to simulate. The default is 10000.
`-nrep`	number of replications to simulate. The default is 1.
`-gradient`	generate code able to compute gradient estimates.
`-h`	produces usage information.
`-help`	produces usage information.
`-?`	produces usage information.

BUGS

The generation of code with gradient is not complete at this time.

NAME

md1 – Waiting time distribution in the M/D/1 queue

SYNOPSIS

md1 [-o <file>] [-xval <real>] [-xfrom <real>] [-xto <real>] [-xstep <real>] [-lambda <real>] [-mu <real>] [-queue] [-log] [-log10] [-debug] [-h] [-?]

DESCRIPTION

Computes the value of the distribution function of the waiting time in a M/D/1 queue with arrival rate λ (given by the -lambda option) and service $1/\mu$ (given by the option -mu). This computation may be required on an interval. The result is a single point (if the option -xval is used) or a list of points suited for plotting.
By default, numbers are written on the standard output.
It is possible to control (with the -xfrom, -xto and -xstep options) the range on which the function is constructed, and the step of the grid. By default, this range is $[0, 10]$ with step 0.1.
It is also possible to compute the function $P(W > x)$ (option -queue) or the function $\log P(W > x)$ (with in addition the option -log for natural logarithm, and -log10 for logarithm to base 10).
The debug options produces extra information on the values read.

OPTIONS

- -? produces usage information.
- -debug produces information on the values read.
- -h produces usage information.
- -o <file> name of the resulting file; the standard output is the default.
- -xfrom <real> sets the lower end of the range to analyze. 0 by default .
- -xstep <real> sets the grid step. 0.1 by default.
- -xto <real> sets the upper end of the range to analyze. 10 by default.
- -xval <real> computes the function at a single point.
- -lambda <real> the arrival rate of customers. 0 by default.
- -mu <real> the “service rate” (inverse of the service length).
- -queue computes the queue of the distribution. The default is to compute the cumulative distribution.
- -log computes the (natural) log of the queue.
- -log10 computes the logarighm to base 10 of the queue.

BUGS

Unknown yet.

NAME

mk1 – Waiting time distribution in the M/Known/1 queue

SYNOPSIS

```
mk1      [-f <file>] [-o <file>] [-lambda <real>] [-customer <int>]
         [-stationary] [-xfrom <real>] [-xto <real>] [-xval <real>] [-xstep
         <real>] [-queue] [-log] [-moment <int>] [-debug] [-h] [-?]
```

DESCRIPTION

Computes the value of the distribution function of the waiting time in a single server, FIFO, infinite buffer queue with Poisson interarrivals and known (deterministic) services.
The services of customers are described in an increasing sequence $(b_0, b_1, \ldots, b_n, \ldots)$, with $b_0 = 0$, which has the following interpretation: b_1 is the service time of the *last* customer which entered the queue; b_2 is the sum of the service times of the *two last* cursomers to enter, and so on. It is assumed that the sequence $\{b_n\}$ ultimately adopts a periodic behavior. More precisely, there exist integers T (as "transient") and P (as "periodic") and a real number $\gamma > 0$ such that: $\forall n \geq T, b_{n+P} = b_n + P\gamma$.
The format for describing the input is the following:

```
λ
T
P
b0 b1 ...  bT
d1 d2 ...  dP
N
```

The real number λ is the input rate of customers in the queue. The integer numbers T and P are the length of the transient part and the periodic part, respectively. If $T = 0$, the sequence starts in periodic mode. The numbers `b0 b1 ... bT` are the first $T + 1$ values of the sequence. They may be equal to the string `eps`, that is, $-\infty$. The sequence must be increasing. The numbers `d1 d2 ... dP` are the P *increments* of the sequence once the periodic behavior is reached. In particular, the sum `d1+...+dP` is equal to $P\gamma$. Increments may be zero. Finally, the integer N is the number of the customer, the response time of which is to be computed. If `-1` is specified the stationary distribution is computed.
The data of the input file may be partially overriden with options. If `-lambda` is chosen, it replaces the value of the data file. Similarly, if the option `-customer` or `-stationary` is chosen.
Unless moments are required (with the `-moment` option), the result is a list of points, representing $P(W_N \leq x)$, the response time distribution of the N-th customer. It is possible to control (with the `-xfrom`, `-xto` and `-xstep` options) the range on which the function is constructed, and the step of the grid. By default, this range is $[0, 10]$ with step 0.1. If the option `-xval` is used, the distribution is computed at a single point.
By default, numbers are written on the standard output. The output may be redirected to a file with the option `-o`.
It is also possible to compute the function $P(W > x)$ (option `-queue`) or the function $\log P(W > x)$ (with in addition the option `-log`).
The `debug` options produces extra information on the values read, and on the computations performed.

OPTIONS

`-f <file>` name of the input file; the standard input is read by default.

`-o <file>` name of the resulting file; the standard output is the default.

`-xfrom <real>` sets the lower end of the range to analyze. 0 by default.

`-xstep <real>` sets the grid step. 0.1 by default.

`-xto <real>` sets the upper end of the range to analyze. 10 by default.

`-xval <real>` computes the function at a single point.

`-lambda <real>` the arrival rate of customers.

`-customer <int>` the number of the customer of which the response time is computed.

`-moment <int>` computes the moment of order `<int>`.

`-queue` computes the queue of the distribution. The default is to compute the cumulative distribution.

`-log` computes the log of the queue.

`-?` produces usage information.

`-debug` produces information on the values read and the computations performed.

`-h` produces usage information.

BUGS

The computation of moments of large order for customers with a large number is unstable.

DESIRABLE DEVELOPMENTS

Moments of order larger than 3 are not implemented for stationary distributions.

NAME

mrouted1 – Waiting time distribution in $M/D/1$ queues in parallel with periodic routing.

SYNOPSIS

```
mrouted1  [-f <file>] [-o <file>] [-lambda <real>] [-customer <int>] [-force]
          [-trace] [-debug] [-h] [-?]
```

DESCRIPTION

Computes the value of the average waiting time of customers in the following queueing system. The customers arrive according to a Poisson process. They are routed according to a *periodic* scheme to one of several queues, each with a single server with constant service, and an infinite capacity. Customers may also be rejected, which amounts to routing them to the "exterior".
The data of the problem is described in the following format.

```
λ
N
s1 s2 ...  sN
D
q1 q2 ...  qD
```

The real number λ is the input rate of customers in the system. The integer N is the number of queues. The positive real numbers `s1 s2 ... sN` are the service times in these queues. The integer D is the period of the routing scheme. The integers `q1 q2 ... qD` are the successive numbers of the queues to which customers are routed within a period. Each of them must be between 0 and N. Queues are numbered from 1 to N. The value 0 means that the customer is not routed to any of the queues and is rejected. However, if the option `-force` is given, any number out of range will be considered to be the exterior.
The data of the input file may be partially overriden: if option `-lambda` is chosen, it replaces the value of the data file.
The default behavior is to compute the expected stationary waiting time of *accepted* customers. With the option `-customer`, the expected waiting time of a certain customer can be computed (not checked yet).
The `-trace` option forces the program to produce additional information, such that the load of the queues and the waiting times in each queue. The `-debug` options produces extra information on the internal computations.

OPTIONS

`-f <file>` name of the input file; the standard input is read by default.

`-o <file>` name of the resulting file; the standard output is the default.

`-lambda <real>` the arrival rate of customers.

`-customer <int>` the number of the customer of which the response time is computed.

`-force` causes the program to interpret out of range queue numbers as 0 (rejection).

`-?` produces usage information.

`-trace` produces queue by queue results.

`-debug` produces information on the internal computations.

`-h` produces usage information.

BUGS

The computation for a given customer has not been checked.

NAME

moment – Empirical moment construction

SYNOPSIS

```
moment    [-f <file>] [-n <int>] [-adjustment <int>] [-nonewline] [-debug] [-h]
          [-?]
```

DESCRIPTION

Reads a list of numbers and builds the empirical moment associated to it. The default moment computed is the average. Otherwise, the order of the moment is specified by the option `-n`.
By default, numbers are taken from the standard input and written on the standard output.
The option `-adjustment` may be used in order to (attempt to) control potential overflow and precision problems. See the notes below for the description of the algorithm. This option is useful only in the case of unusually large and variable samples.

OPTIONS

`-?` produces usage information.

`-debug` produces information on the values read.

`-f <file>` name of the input EGL file; if none is specified, the graph is read from the standard input.

`-h` produces usage information.

`-n <int>` computes the n-th moment. The default is 1.

`-adjustment <int>` set the period for adjustments of the estimator. The default is no adjustment.

`-nonewline` suppresses the newline when printing the result. Incompatible with `-debug`

BUGS

Unknown yet.

DESIRABLE DEVELOPMENTS

There should be centered moments (at least variance) available.

NOTES

The periodic adjustment of the estimator works in the following way. The program keeps a current value of the estimated moment $\hat{m}$, and the current value of the difference:

$$s_n = \sum_{i=1}^{n} (X_i - \hat{m})$$

using the recurrence

$$s_{n+1} = s_n + (X_{n+1} - \hat{m}) .$$

Once every p sample, the value following adjustment is performed:

$$\begin{aligned} \hat{m} &\leftarrow \hat{m} + \frac{s_n}{n} \\ s_n &\leftarrow 0. \end{aligned}$$

The idea of a periodic adjustment is that the increments of the sum s_n should not be too small compared to the sum itself in order to avoid roundoff errors. On the other hand, if adjustment is systematic, the estimation becomes: $\hat{m} \leftarrow \hat{m} + X_n/n$ and for large samples, roundoff errors may happen for the same reason.

onoff_sim – Simulation of On/Off fluid queues

SYNOPSIS

```
onoff_sim [-o <file>] [-on_law <law>] [-off_law <law>] [-capacity <real>]
          [-peak_rate <real>] [-W0 <real>] [-nbcycles <int>] [-xfrom <real>]
          [-xto <real>] [-xval <real>] [-xstep <real>] [-queue] [-log]
          [-nonewline] [-traj] [-debug] [-h] [-?]
```

DESCRIPTION

Simulates a fluid queue which input is an On/Off process. The parameters of the queue are specified using the command line arguments:

Parameter	Option	Default Value
Law of On periods	`-on_law`	Exp(1.0)
Law of Off periods	`-off_law`	Exp(1.0)
Peak Rate	`-peak_rate`	1.0
Server Capacity	`-capacity`	1.0
Initial Workload	`-W0`	0.0

The specification of the stochastic distributions uses a simple syntax like `E 2.0`, `U 0.0 1.0`, etc. The complete description of the possibilities is in the manual page for `gen_iid`.

The length of the simulation is measured in number of *cycles*, a cycle being one On period and the followin Off period. This number is controled with the option `-nbcycles`. It defaults to 100000.

There are two possible ways to use the simulation. The first one is to produce trajectories. This behavior is obtained using the `-traj` switches. The output is a list of points $(t, W(t))$ suited for plotting with `gnuplot` for instance. The initial workload may be specified with the `W0` option.

The second use is to gather statistics on the distribution of fluid in the queue: $Q(x) = \mathbf{P}(W > x)$. The result is the following empirical estimator:

$$Q^{(N)}(x) \;=\; \frac{1}{T(N)} \int_0^{T(N)} \mathbf{1}\{W(u) > x\}\; du \;.$$

where N is the number of cycles of the simulation, $T(N)$ the final time (end of the Nth Off period), and $W(u)$ the trajectory of the workload. The function is actually computed at the points x_i of a *discrete range* specified by the options `-xfrom`, `-xto` and `-xstep`. The default values are 0.0, 10.0 and 0.5, that is, the points are $x_i = 0.0, 0.5, 1.0, \ldots, 9.5, 10.0$. The result is then a list of points $(x_i, f(x_i))$, where the function f depends on command line options:

Option	Function f Computed
default	$1 - Q^{(N)}(x)$
`-queue`	$Q^{(N)}(x)$
`-queue` and `-log`	$\log(Q^{(N)}(x))$
`-queue` and `-log10`	$\log_{10}(Q^{(N)}(x))$

The results are produced on the standard output by default. They can be redirected in a file with the option `-o`.

OPTIONS

`-on_law <law>` specifies the distribution of the length of On periods. By default, this is `E 1.0`.

`-off_law <law>` specifies the distribution of the length of On periods. By default, this is `E 1.0`.

`-capacity <real>` sets the capacity of the server. 1.0 by default.

`-peak_rate <real>` sets the peak rate of the source. 1.0 by default.

`-nbcycles <int>` the number of cycles to simulate; the default is 100000.

`-traj` prints the trajectory simulated.

`-W0 <real>` sets the initial workload.

`-xfrom` sets the start of the interval on which to gather statistics. 0.0 by default.

`-xto` sets the end of the interval on which to gather statistics. 10.0 by default.

`-xstep` sets the interval between values on the interval on which statistics are computed. 0.5 by default.

`-xval` specifies a single value at whith to estimate the workload distribution.

`-queue` prints the tail of the distribution.

`-log` prints the logarithm of the tail of the distribution.

`-log10` prints the decimal logarithm of the tail of the distribution.

`-o <file>` writes the result to a file.

`-debug` produces additional information on the simulation.

`-h` produces usage information.

`-?` produces usage information.

DESIRABLE DEVELOPMENTS

There should be a way to estimate moments.

BUGS

Statistics may not work properly when the peak rate is less than the capacity. Trajectories appear to be ok in this case, however.
It is not possible to specify simulations of more than $2^{31} \equiv 2.15 \times 10^9$ cycles.

NOTES

The function $Q^{(N)}(x)$ can be computed at only certain points using the following observations. Denoting by: r the peak rate, c the capacity, a sample path of the workload is a continuous and piecewise affine function with slopes $r - c$ (On periods), c (Off periods, positive workload) and 0 (Off periods, zero workload). Then, if $N+(x)$ and $N^-(x)$ are the number of segments crossing the level x respectively upwards and downwards, $Q^{(N)}(x)$ satisfies (where it is differentiable):

$$\frac{dQ^{(N)}}{dx}(x) \;=\; -\left(\frac{1}{r-c}N^+(x) \;+\; \frac{1}{c}N^-(x)\right) \;.$$

Hence, because on every finite trajectory, $Q^{(N)}(\infty) = 0$:

$$Q^{(N)}(x) \;=\; \int_x^{\infty} \left(\frac{1}{r-c} N^+(u) \;+\; \frac{1}{c} N^-(u) \right) \, du \; .$$

Moreover, the functions N^+ and N^- have the following property:

$$\int_a^b N(u)du \;=\; (b-a)N(a) \;+\; \sum_{a_i}(b-a_i) \;+\; \sum_{d_i}(d_i - b) \tag{5.1}$$

$$\;=\; (b-a)N(b) \;+\; \sum_{a_i}(a-a_i) \;+\; \sum_{d_i}(d_i - a) \tag{5.2}$$

where the a_is are the points *in the interval* $[a, b]$ where N increases ("arrivals") and the d_is those where it decreases ("departures"). For both N^+ and N^-, the a_i are the ends of On periods, and the d_i the times at which Off periods end, or the queue becomes empty.
Consequently, the function $Q^{(N)}$ may be computed at all points $\{x_0, x_1, \ldots, x_p\}$ by keeping the count of:

- The total number of "arrivals" and "departures" in each interval $[x_k, x_{k+1}]$. This allows to compute the values $N^{\pm}(x_k)$.
- The quantity $\sum_{a_i}(b-a_i) + \sum_{d_i}(d_i - b)$. The values of $Q^{(N)}$ may then be recovered using the formulas above, and:

$$Q^{(N)}(x_k) \;=\; Q^{(N)}(x_{k+1}) \;+\; \int_{x_k}^{x_{k+1}} \left(\frac{1}{r-c} N^+(u) \;+\; \frac{1}{c} N^-(u) \right) du \; .$$

NAME
pn2c – C code generation for Petri Nets

SYNOPSIS

```
pn2c     [-bs <buf_size>] [-f <file>] [-scaling <int>] [-tb <time>] [-te
         <time>] [-tmp <dir>] [-tmax <int>] [-testing] [-gss] [-o <file>]
         [-noYstatic] [-debug] [-h] [-?]
```

DESCRIPTION
Creates a set of C-language files, which purpose is to perform an equational simulation of a Petri Net.
This is possible only if the Petri net has the following properties:

- all firing times and holding times are constant, and multiple of a certain time unit δt.
- all places with more than one output have a routing policy of type Random or Cyclic. Competition is allowed, provided that
  - all output transitions are immediate;
  - a priority order has been imposed on the transitions, in case of simultaneous enabling.
- there are no cycles of duration zero in the Net. Note that competition with priorities is considered to induce a chain of dependences with zero time, which is taken into account in the cycle analysis.

The result should be compiled with the standard, network independent piece of code: "`skeletton.c`". A copy of this file can currently be found in the directory:

`$ERSHOME/lib/pn_sim`

For instance:

```
cc skeletton.c -o simulator -lm
```

This in turn produces an executable, `simulator`, which execution provides the performance results. (If you prefer not to compile everything yourself, use `pn_sim` instead with the option `-compile`.)
The resulting executable accepts the option `-tmax <int>` which allows to change the time until which the network is simulated.
The Net is read form an PNL-file and the following files are produced:

- `pnc_graph_parameters.h` a configuration file containing the parameters of the Petri Net and of the simulation.
- `pnc_declarations.h` other declarations.
- `pnc_initial_evolution.c` the code for the beginning of the evolution, for which firing times (may) depend on initial lag times of tokens.
- `pnc_evolution.c` the code for the main loop.

These files may be written in a directory different from the current one using the `-tmp` option.
Time scale. The way `pn2c` recognizes that all (constant) firing/holding times are multiple of a certain δt is the following:

- Multiply these times by an integer number Δ.
- Check that all results are integer; if not, the Petri Net is rejected.
- Find the greatest common divisor of all results: d. The time scale δt is then d/Δ.

The value of Δ is controlled by the option `-scaling`. Its default value is 10.
Circular buffer. The simulation program uses a cyclic buffer to (potentially) increase locality and execution speed. The buffer size can be controlled with the `-bs` option. There are constraints on the minimal size of this buffer which are enforced by the program.
When making references to the internal "circular" two dimensional table, one possibility is to use moving pointers pointing to rows (this is the default). The second possibility is to use two dimensional references (like: `Y[i][j]`). This behavior is obtained using the option `-noYstatic`.

Testing. It is possible to perform only checking that the network satisfies the proper assumptions (option `-testing`). With the additional `-gss` option, the result is produced in a format suitable for use with GSS. It consists in a list of integers, separated by the keyword `_E_O_E`. The first integer is the size of the list, and the rest is a list of couples (x, n) describing the objects not satisfying the proper assumptions. Here, x is a code for the type of the object (0 for a transition, 1 for a place) and n the number of the object.

OPTIONS

`-debug` produces information on the network read.

`-f <file>` name of the input PNL file; if none is specified, the graph is read from the standard input.

`-bs <buf_size>` the size of the data buffer; 1000 by default.

`-tmax <int>` overrides the simulation length of the PNL file.

`-scaling <int>` specifies the maximum scaling factor. The default is 10.

`-testing` only checks if the assumptions permitting code generation are satisfied.

`-gss` produces a list of objects violating the assumptions, in a GSS format.

`-o <file>` redirects output in a file.

`-tmp <dir>` the directory where the C files are to be written; the default is the current directory.

`-noYstatic` prevents the use of table pointers.

`-?` produces usage information.

`-h` produces usage information.

BUGS
Unknown yet.

DESIRABLE DEVELOPMENTS
There should be less files produced.

The trace mechanism should be implemented.
There could be some more optimization in the size of the code produced and the execution time.

NAME

pn2eg – PNL to EGL translation

SYNOPSIS

`pn2eg    [-f <file>] [-debug] [-o <file>]`

DESCRIPTION

Translates a Petri Net (coded in PNL) into an Event Graph (coded in EGL), if, of course, the graph is an Event Graph.

OPTIONS

`-?` produces usage information.

`-h` produces usage information.

`-debug` produces information on the network read.

`-f <file>` name of the input PNL file; if none is specified, the graph is read from the standard input.

`-o <file>` name of the resulting EGL file; if none is specified, the result is written to the standard output.

BUGS

Unknown

NAME

pn2graph – Translate a Petri Net into a graph drawing format

SYNOPSIS

```
pn2graph  [-f <file>] [-mode dot|daVinci|scilab] [-fullname] [-debug] [-o
          <file>]
```

DESCRIPTION

This program is used for displaying a Petri Net that was not created with a graphical editor. It uses the services of one of the available Graph Layout systems.
Currently, three systems are available:

`dot/dotty` (AT&T Bell Labs)

`daVinci` (University of Brehme)

`scilab` (META2 project, INRIA)

`pned` the Petri Net editor of Ers.

In the generation of graphs, the places are labelled with the string `Px`, where `x` is its number in the PNL file. Likewise, transitions are labelled with `Tx`, `x` being their number. It is possible to make the name of the objects appear in the labels using the `-fullname` option. Note that this will produce nodes of different sizes...

OPTIONS

`-?` produces usage information.

`-mode <format>` specifies the format of the graph. The default is the daVinci format.

`-f <file>` name of the input EGL file; if none is specified, the graph is read from the standard input.

`-h` produces usage information.

`-o <file>` name of the resulting graph file; the standard output is the default.

BUGS

Submodels are not handled correctly. Not at all with `-daVinci`.

DESIRABLE DEVELOPMENTS

There should be a way to enforce the size of all objects, whatever the length of the label they have.

NAME
pn2lin – PNL to linear program translation

SYNOPSIS
`pn2lin [-f <file>] [-check] [-o <file>] [-min] [-linprog] [-objective <string>] [-mufile <file>] [-Bfile <file>] [-bound <int>] [-verbose] [-novar] [-debug]`

DESCRIPTION
Translates a Petri Net (coded in PNL) into a linear program, which solution provides bounds on some performance measures of the network.
The linear program is a set of equations, established by Z. Liu, representing various conservation laws and constraints [13]. The linear program is solved by `lp_solve` (available publicly through `ftp` at: `ftp.es.ele.tue.nl`).
The default value of the objectve function is an upper or a lower bound on the sum of the throughputs of all transitions. It can be modified by the `-objective` option, which allows to specify a replacement function. This must be a string without blank spaces containing a linear combination of the variables of the program, that is: `x*`, `q*`, `t*` and `y*_*`, where "`*`" denotes a place or transition number. The correctness of the function is left entirely to the responsibility of the user.
The entire process consists in generating a linear program, solving it, and using the results to check which constraints are tight. The normal result is a line containing the value of the objective function. This behavior may be modified by the `-linprog` option, under which only the linear program is produced, or by the `-check` option, which produces the list of equalities which are the tight constraints.
It is possible to modify the values of the transition rates by specifying them in a file, as a list of pairs (number of the transition, rate), one by line, such as:

```
2 1.0
5 3.5
```

and using the `-mufile` option. Transitions are numbered from 0.
It is also possible to tell the program that a bound on the marking exists, either by using the `-bound` option (if it is a global bound), or with the `-Bfile` option, after describing the bounds in a file as a list of pairs (number of the place, bound), one by line, such as:

```
0 5
2 4
7 1
```

Places are numbered from 0. A way to obtain such bounds is to use `pn_bound`.
The size of the linear program is proportional to the product of the number of places and the number of transitions in the net.

OPTIONS

`-Bfile <file>` the name of a file containing upper bound specifications.

`-bound <int>` specifies a global bound for places.

`-check` produces the list of tight constraints.

`-debug` produces information on the network read and some intermediate results.

`-f <file>` name of the input PNL file; if none is specified, the graph is read from the standard input.

`-h` produces usage information.

`-linprog` produces the linear program as the result. The default is to do the entire processing.

`-min` computes a lower bound. The default is to compute an upper bound.

`-mufile <file>` the name of a file containing transition rate specifications.

`-novar` inhibits the generation of variance and covariance constraints.

`-o <file>` name of the output file.

`-objective <string>` the objective function. The default is the maximum (or minimum) throughput.

`-verbose` adds more comments in the linear program.

`-?` produces usage information.

BUGS

Probably. Strange behaviors have been observed for some programs with some value functions. This is certainly due to problems (numerical instability) with the solver `lp_solve` which declares some problems unsolvable when they are.
The variance-type constraints do not take weighted arcs into account correctly.

DESIRABLE DEVELOPMENTS

It should be possible to select only some sets of equations to be included in the linear program.

Some serious testing should be done for weighted and/or inhibitor arcs.
Some known sets of equation have not been implemented yet.
There should be mode flexibility in specifying the objective function: in particular, the symbolic names of the objects (known in the PNL file) should be useable.

NAME
pn2mixt – PNL to mixt PROSIT/Equational translation

SYNOPSIS

```
pn2mixt   [-f <file>] [-bs <buf_size>] [-scaling <int>] [-tmax <int>] [-testing]
          [-O <int>] [-agat] [-nostats] [-noprosit] [-noYstatic] [-tmp <dir>]
          [-o <file>] [-op <file>] [-oc <file>] [-debug] [-h] [-?]
```

DESCRIPTION
Translates a Petri Net (coded in PNL) into two C++ files using standard simulation and equational simulation in the PROSIT environment.
The Petri Net is divided into two parts: the PROSIT part and the equational (EQ) part (see below how). The PROSIT part use the Queueing Network (Pastec) library. See pn2prosit for the principles related to this part. The generation of this part may be prevented by the option -noprosit.
The EQ part is imbedded into a special PROSIT object (of class EQ_Thing). The behavior of this object is to compute firings using evolution equations. See pn2c for the principles of this method.
The PROSIT part is written into a file called main.cc. The EQ part is in a file called EQ_Module.c. In addition, a makefile called Makefile.Prosit is produced. The program may be compiled using:

```
make -f Makefile.Prosit
```

The resulting simulator is called sim. It admits the command line option -tmax <real> allowing to change the simulation duration from its default value (read from the PNL file).
The names of the source files (but not that of the Makefile!) and their directory may be changed with the options -op, -oc and -tmp.
The use of the -agat option causes the creation of a configuration file allowing to display trajectories with the AGAT tool. This file is called sim.agat. It is used with the command:

```
agat -f sim.agat sim
```

Note that AGAT will not report stats for source objects, nor for objects that have been translated into routing stations.

Objects of the Petri Net are recognised as being in the PROSIT part if their name begins by the string "PROSIT|". The conditions imposed on the Petri Net are that of pn2prosit for the PROSIT part (mainly: only random routing at places), and pn2c for the EQ part (mainly: constants firing/holding times, restrictions on competition routing). The interface between the two parts requires in addition that EQ places with competition have all their output transitions in the EQ part as well.

The two parts interoperate nicely thanks to the following "tricks":

- All EQ objects of the Petri Net are imbedded into a single PROSIT EQ object. Queue objects send requests to this object.
- The EQ object keeps a count of the tokens sent to it by the PROSIT objects that are at the "interface". It converts incomming request into increments of firing counts into its equational engine. The identity of the origin and destination objects are recognised through the class of the request.
- Each unit time, the EQ object processes requests arrived in the last interval, uses equatinos to compute its internal evolution, and deduces the firing completions that occur at the same instant. The tokens that go to the PROSIT part are converted into requests.

By default, the PROSIT program uses the StatTool for performing basic statistics. If the option `-nostats` is used, no calls to StatTool are performed, and the output is limited to the count of services at each object. Options `-agat` and `-nostats` are incompatible.

OPTIONS

`-?` produces usage information.

`-h` produces usage information.

`-debug` produces information on the network read.

`-f <file>` name of the input PNL file; if none is specified, the graph is read from the standard input.

`-o <file>` name of the output file (usefull with `-testing`).

`-oc <file>` name of the output file for the PROSIT part. The default is `main.cc`.

`-oc <file>` name of the output file for the EQ part. The default is `EQ_Module.c`.

`-O <number>` swithes to the optimization level "number" (see `pn2prosit`).

`-agat` produces a file for AGAT.

`-nostats` prevents use of the StatTool package (see `pn2prosit`).

`-bs <buf_size>` the size of the data buffer (see `pn2c`); 1000 by default.

`-tmax <int>` overrides the simulation length of the PNL file.

`-scaling <int>` specifies the maximum scaling factor (see `pn2c`). The default is 10.

`-testing` only checks if the assumptions permitting code generation are satisfied (see `pn2c`).

`-noprosit` prevents the generation of the PROSIT part.

`-noYstatic` prevents the use of table pointers (see `pn2c`).

BUGS

The initial set-up times of initial "tokens" is ignored.

Infinite servers are actually finite servers with 100 servers.
There are problems with simultaneous events between the PROSIT part and the EQ part. It is safer that the PROSIT part does not evolve in discrete time.

DESIRABLE DEVELOPMENTS

Test the program and the resulting PROSIT models.

Provide support for the missing distributions (`g`, `H` and `D`).
Improve optimization by removing more immediate objects.

NAME

pn2pn – Perform PNL to PNL transformations

SYNOPSIS

`pn2pn [-f <file>] [-determinize] [-o <file>] [-h] [-?]`

DESCRIPTION

Translates a Petri Net (coded in PNL) into another Petri Net (also coded in PNL), performing the transformations specified by the options. By default, the input graph is simply rewritten, which provides some kind of pretty printing.

The option `-determinize` changes the distributions of firing durations and hoding times, and replaces them by a constant distribution with the same average. It does nothing about the random routing at places. This may be used to create networks suitable for analysis with `pn2c` or `pn2mixt`.

The option `-flatten` forces all submodels to be (recursively) expanded so as to produce a Petri Net included in a single file. With the additional option `-noExpandNames`, the original object names are kept, whereas the default behavior is to prefix all names with the submodel names in a UNIX-like notation: `"Sumb1/Subm2/TheTrans"`. When no submodel name is available, the string `"*/"` is used.

OPTIONS

`-?` produces usage information.

`-h` produces usage information.

`-debug` produces information on the network read.

`-determinize` replaces all timing by constants.

`-flatten` expands submodels.

`-noExpandNames` does not expand object names when flattening.

`-f <file>` name of the input PNL file; if none is specified, the graph is read from the standard input.

`-o <file>` name of the resulting PNL file; if none is specified, the result is written to the standard output.

BUGS

Unknown

NAME
pn2prosit – PNL to PROSIT translation

SYNOPSIS
```
pn2prosit [-f <file>] [-o <file>] [-agat] [-O <int>] [-tmax <int>] [-nostats]
          [-debug] [-h] [?]
```

DESCRIPTION
Translates an Petri Net (coded in PNL) into a C++ file using the Queueing Network (Pastec) library of PROSIT. The resulting file is called `main.cc`. This name may be changed with the option `-o`. In addition, a makefile called `Makefile.Prosit` is produced. This name may not be changed. The program is then compiled using:

```
make -f Makefile.Prosit
```

The resulting simulator is called `sim`. It admits the command line option `-tmax <real>` allowing to change the simulation duration from its default value (read from the PNL file).
The use of the `-agat` option causes the creation of a configuration file allowing to display trajectories with the AGAT tool. This file is called `sim.agat`. It is used with the command:

```
agat -f sim.agat sim
```

Note that AGAT will not report stats for source objects, nor for objects that have been translated into routing stations.

The Petri Net may have any topology, but if places with conflicts are present, the routing policy of this place must be random (and not cyclic not competition).
The general principles for the translation are the following:

- Transitions without input except their recycling are transformed into sources (`Pastec_Source`). Transitions without input at all are tolerated (but not recommended) and considered as sources.
- Other transitions are transformed into queueing stations (`Pastec_Server_Normal_FIFO`). If a transition is recycled (with a place with zero holding time), it becomes a finite server station. Otherwise, it is an infinite server station. If the indegree is two or more, a "split-match" station (`Pastec_FS_Station`) is prepended. If the outdegree is two or more, a split-match station is appended.
- If the transition is immediate and serial, it is transformed into a routing object (`Pastec_Routing`).
- If a transition has no output (except for its recycling), a sink (`Pastec_Sink`) is generated.
- Places are translated into infinite server stations, unless they are immediate, in which case they are translated into routing objects.
- If optimization is asked (`-O` option), places that are immediate and serial are removed. The optimization level is ignored for the time being.

The use of Prosit synchronization mechanisms necessitates the introduction of auxiliary customer classes. For simplicity, the class number expected at a fusion object is the number of the object at the other end of the link: the place if it has not been removed my optimization, and the origin transition otherwise.
The number of classes is therefore equal to the maximum of the numbers of places and transitions.

By default, the PROSIT program uses the StatTool for performing basic statistics. If the option `-nostats` is used, no calls to StatTool are performed, and the output is limited to the count of services at each object. Options `-agat` and `-nostats` are incompatible.

OPTIONS

- `-?` produces usage information.
- `-h` produces usage information.
- `-debug` produces information on the network read.
- `-f <file>` name of the input PNL file; if none is specified, the graph is read from the standard input.
- `-o <file>` name of the resulting C++ file; the default if `main.cc`.
- `-tmax <int>` overrides the simulation length of the PNL file.
- `-O <number>` switches to the optimization level “number”.
- `-agat` produces a file for AGAT.
- `-nostats` prevents use of the StatTool package.

BUGS

The initial set-up times of initial “tokens” is ignored.

Infinite servers are actually finite servers with 100 servers.
There may be trouble with the AGAT animation for places.

DESIRABLE DEVELOPMENTS

Test the program and the resulting PROSIT models.

Provide support for the missing distributions (`g`, `H` and `D`).
Improve optimization.
Make output file names configurable.

NAME

pn2qnap – PNL to QNAP translation

SYNOPSIS

`pn2qnap [-?] [-debug] [-f <file>] [-h] [-o <file>] [-O <number>] [-tb <time>] [-te <time>] [-tfile <file> ] [-tmax <int>] [-counters] [-h] [?]`

DESCRIPTION

Translates an Petri Net (coded in PNL) into an queueing network, (coded in the QNAP2 language).
WARNING: this page has been copied from `eg2qnap`. It may contain inaccuracies.
A queue object is generated for each transition and each place, unless optimization is asked (see below). The synchronization constraints are implemented in QNAP using the FISSION/FUSION constructs. To that effect, it is necessary to use auxiliary customer classes. The number of classes is equal to the maximum input degree of transitions. The routing must be probabilistic. No unrecycled transitions are allowed.
Optimization consists in trying to remove from the QNAP code unnecessary objects such as queues with 0 service, and reduce the explicit synchronization. For the time being, only places with no delay are skipped, and the optimization level is ignored.
There are two measurements available: one by "daters" (the default) and one by "counters" (option `-counters`). Both produce an estimate of the cycle time.
The simulation needs that a maximum time be specified. This is done with the option `-tmax`. When the simulation unit is "daters", this is the maximum simulation time. When "counters" are simulated, this is the maximum number of firings that are allowed to happen at the transition number 0. The default is 1000 units.
There is provision for generating a trace, in connection with the editor `ers`. Trace is selected when either of the `-tb` or `-te` options have been specified. When trace has been selected, the simulation program produces it (in the Animator language) in the file specified by `-tfile` (`trace.results` by default).
Trace and optimization are mutually exclusive. The last command read on the command line defines which one is activated.

OPTIONS

- `-?` produces usage information.
- `-h` produces usage information.
- `-debug` produces information on the network read.
- `-f <file>` name of the input PNL file; if none is specified, the graph is read from the standard input.
- `-o <file>` name of the resulting QNAP file; if none is specified, the result is written to the standard output.
- `-O <number>` switches to the optimization level "number".
- `-tmax <int>` the simulation length; 1000 by default.
- `-tb <time>` the time at which the trace begins; 0 by default.
- `-te <time>` the time at which the trace ends; 1000 by default.

`-tfile <file>` the name for the trace file; `trace.results` by default.

`-counters` produces the counts of event at each transition.

BUGS

The initial set-up times of initial "tokens" is ignored.

DESIRABLE DEVELOPMENTS

Test the program and the resulting QNAP models.

Improve optimization.

pn_bound – Bounds on Petri Net markings

SYNOPSIS

```
pn_bound  [-f <file>] [-debug] [-o <file>] [-min] [-total] [-place <int>]
          [-Bfile <file>]
```

DESCRIPTION

Computes bounds on the marking of a Petri Net (coded in PNL) using a linear program.
The linear program is actually a set of equations relating the marking vector, the firing count vector and the incidence matric of the graph. The linear program is solved by `lp_solve` (available publicly through `ftp` at: `ftp.es.ele.tue.nl`).
The default behavior of the program is to search upper bounds on the marking for all places, by generating and solving as many linear programs as there are places. This behavior may be altered by the `-min` option, under which lower bounds are sought, and the `-place` option, which allows to choose one single place to analyze.
In addition, the program may produce (if used with the option `-Bfile`) a file containing a list of pairs (place,bound). This format is compatible with the "Bfile" format of `pn2lin`.

OPTIONS

- `-?` produces usage information.
- `-h` produces usage information.
- `-debug` produces information on the network read and some intermediate results.
- `-f <file>` name of the input PNL file; if none is specified, the graph is read from the standard input.
- `-o <file>` name of a file where the output is written.
- `-place <int>` computes the bound only for the specified place. The number of the place is that of the PNL file.
- `-min` computes a lower bound. The default is to compute an upper bound.
- `-total` computes a bound for the total population of the network.
- `-Bfile <file>` the name of a file where the bounds specifications are written.

BUGS

Unknown yet.

DESIRABLE DEVELOPMENTS

There should be mode flexibility in specifying the objective function: in particular, the symbolic names of the objects (known in the PNL file) should be useable.
The system does not support multiple-valued arcs yet.
The system does not support inhibitor arcs yet.

pn_read – Parses a PNL file and display network information

SYNOPSIS

`pn_read [-f <file>] [-h] [-?]`

DESCRIPTION

Reads a Petri Net (coded in PNL) and display debugging information about place/transition connections, routing policies, initial marking and so on.
This information is more or less the one produced by any program of the “pn” family with the `-debug` option.

OPTIONS

- `-?` produces usage information.
- `-h` produces usage information.
- `-f <file>` name of the input PNL file; if none is specified, the graph is read from the standard input.

BUGS

No known bugs.

pn_sim – General purpose Petri Net simulator

SYNOPSIS

```
pn_sim    [-f <file>] [-tmax <int>] [-bs <int>] [-scale <int>] [-compile]
          [-keep] [-c] [-debug]
```

DESCRIPTION

Simulates an Petri Net (coded in PNL) by iteratively computing its evolution equations.
The simulation applies to Petri nets with the following properties:

- all firing times and holding times are constant, and multiple of a certain time unit δt.
- all places with more than one output have a routing policy of type Random or Cyclic. No competition is allowed.

Note that although no random times are allowed, certain discrete distributions may be modelled using the random routing mechanism and adequate places and transitions.
pn_sim checks that all firing times, holding times and initial lags of tokens are integers, after multiplication by a certain scaling factor. The maximum value of this factor can be controlled with the option `-scale`. The default value is 10. With this value, a firing time of 1.2 will be declared "integer" whereas a value of 1.23 will not. The program computes the smallest possible scaling factor among those that divide the maximum value.
The length of the simulation is measured in time units. It may be controled with the option `-tmax`. The default value is 1000 units of time.
The simulation uses a data buffer, the size of which may be controled with the option `-bs`. The total memory used is roughly the product of this size and the number of transitions in the Petri Net. It is probably desirable to keep it relatively small.
There are two possible ways to make the simulation. The first one is interpreted. It is the default. The second one consists in compiling a network-specific simulator using `pn2c`. It is chosen with the option `-compile`. The name of the executable is `PN_Sim`. With this option, the switch `-keep` allows to keep the temporary files and the simulator, instead of removing them at the end. This method uses resources, which may be listed and checked with the option `-c`.

OPTIONS

`-a` does only the structural analysis, and not the simulation.

`-bs <int>` sets the size of the data buffer; 1024 by default.

`-c` checks for resources.

`-compile` creates a compiled simulator and executes it.

`-debug` produces information on the network read.

`-h` produces usage information.

`-keep` keeps the source files and the simulator; the default is to clean up everything.

`-tmax <int>` simulation length. The default is 1000.

`-?` produces usage information.

DESIRABLE DEVELOPMENTS

Should be tested more throughly, as for `eg_sim`.

Does not provide trace capability.

NAME

pnanim – A graphical interface for Petri Nets

SYNOPSIS

`pnanim [<file>]`

DESCRIPTION

This application allows to animate the evolution of Petri Nets, based on a trace file generated ba some simulator. See section 3.7 for more details.

OPTIONS

`<file>` the (optional) name of a GSS file containing the model

BUGS

Unknown

DESIRABLE DEVELOPMENTS

The interface is limited to the standard “great-null-tool” provided with the distribution of the Animator. It can be much improved.

NAME
pned – A graphical interface for Petri Nets

SYNOPSIS
`pned [<file>] [-matrix] [-ts <int>] [-ps <int>] [-nolog] [-debug]`

DESCRIPTION
It allows to edit Petri Nets and do various operations, including the generation of simulation codes. For more information on the functionalities of this interface, see section 3.1.
The name of a file may be passed as argument. Normally, this is a file in the `pned` format. However, if the option `-matrix` is specified, then the file should contain a (max, +) matrix in a format recognized by ERS (standard or SCILAB). The Petri Net created is then the standard Event Graph corresponding to the matrix.

OPTIONS

- `<file>` the (optional) name of a file containing the model.
- `-matrix` indicates that the file name argument is to be interpreted as the name of a (max,+) matrix.
- `-ts <int>` sets the height of transitions. The default is 15.
- `-ps <int>` sets the radius of places. The default is 15.
- `-nolog` sets the logging style to none, thus preventing the launch of a Logger.
- `-debug` turns on a verbose debugging mode.

BUGS
Lots

DESIRABLE DEVELOPMENTS
Lots too

NAME

scscyc – Transient and periodic behavior of (max, +) matrices

SYNOPSIS

`scscyc [-f inputfile] [-o outputfile] [-transient] [-precision <int>]`

DESCRIPTION

Computes the eigenvalues, the critical graph and the transient behavior of a (max, +) integer matrix. The spectral analysis of (max, +) operators appears in [3] for instance.
The operator is described in the input file, which must contain the matrix with the following convention: first, the size of the matrix, then the matrix itself. It the `-transient` option is specified, an additional initial vector must be provided.
All entries should be real numbers, or the string `eps`, which represents $-\infty$. All the internal algorithms use integer numbers: the real numbers are therefore internally converted to integers after multiplication by the number specified by the option `-precision`. This means that if, for instance, `-precision` is 100, only two decimal digits will be taken into account. The default value of the precision is 1. Numbers smaller than -1000 times the precision are treated as $-\infty$.
The standard results are:

- the eigenvalue of the matrix, given in rational form: p/q;
- the adjacency matrix of the critical subgraph, that is, the graph of critical circuits. The convention is: 0 if there is an arc, and `eps` if not;
- the agregated critical subgraph, with strongly connected components replaced by a single node;
- the number of strongly connected components of the critical graph (scs); and the list of the nodes in each component;
- the cyclicity of the graph (cyc), that is, the least common multiple of the greatest common divisors of the lengths of the cycles in each strongly connected component.

In addition, the `-transient` option produces the transient part of the (max, +) recurrence $Y(n) = A^n \otimes Y(0)$, that is, the list of the successive iterations of A on the initial vector, until a periodic behavior is encountered. The last result is the length of the transient period.
The results are written to the standard output, or to a file using the option `-o`.

OPTIONS

`-f <file>` reads the matrix in `<file>`; if not is specified, the matrix is read from the standard input.

`-o <file>` writes the result in file `<file>`.

`-h` produces a help message.

`-transient` computes the transient part of the recurrence.

`-precision <int>` specifies the precision for converting reals to integers. 1 by default.

BUGS

The algorithm does not work perfectly when the matrix is not irreducible and there are critical circuits in more than one irreducible components.

DESIRABLE DEVELOPMENTS

The transients of the powers of the matrix should be available.

NAME

teted – A graphical interface for Heaps of Pieces (Tetris)

SYNOPSIS

`teted [<file>] [-ss <int>] [-nolog] [-debug]`

DESCRIPTION

It allows to edit set of pieces and do various operations, including playing the "Tetris game", computing bounds and performing simulations. For more information on the functionalities of this interface, see section 3.4.

OPTIONS

`<file>` the (optional) name of a file containing the model.

`-ss <int>` sets the size of squares in the interface. The default is 20.

`-nolog` sets the logging style for results to none. Results will appear in popup windows and will not be kept.

`-debug` turns on a verbose debugging mode.

BUGS

Lots

DESIRABLE DEVELOPMENTS

Lots too

tet_bound – Computes bounds for random Tetris games

SYNOPSIS

```
tet_bound [-f inputfile] [-o outputfile] [-nmax int] [-stats] [-dump] [-debug]
          [-h] [-help] [-?]
```

DESCRIPTION

Computes bounds on the Lyapunov exponent of a system given by the random stacking of pieces.
The name of the file containing the description of the heaps of pieces is given by the option `-f`. By default, the data is read on the standard input.
The format for the input data system is described in documentation for `Teted`.

Two types of bounds are computed:

- A simple lower bound on the Lyapunov exponent, given by the maximum of the expected increase in each column.
- An upper bound and a lower bound obtained by constructing a truncated Markov chain of possible stack profiles. The truncation is performed in such a way as to increase or decrease the expected growth rate of the stack.

The method based on Markov chains uses the parameter `-nmax` to indicate the size of the truncated state space. The default value is 200 (for a short computation time).
The result is written to the standard output, or a file is the option `-o` is used.
The option `-stats` provides information on the exploration and the hash table used in the exploration of the state space.
The option `-dump` just dumps the Reward Markov chain in the MCL language.

OPTIONS

- `-f <file>` name of the file containing the matrices.
- `-o <file>` sends the result in the named file.
- `-nmax` number of states in the truncated Markov chain. The default is 200.
- `-stats` produces information on the exploration of the state space and the hash table.
- `-dump` dumps the Reward Markov chain; does not solve it.
- `-h` produces usage information.
- `-help` produces usage information.
- `-?` produces usage information.

BUGS

The color specification in the input file must be in the hexadecimal form `#rrggbb`. Symbolic names are not recognized.

DESIRABLE DEVELOPMENTS

A better hash table would prove useful when exploring very large state spaces.

NAME
tganim – A graphical interface for Petri Nets

SYNOPSIS
`tganim [<file>]`

DESCRIPTION
This application allows to animate the evolution of Petri Nets, based on a trace file generated ba some simulator. See section 3.9 for more details.

OPTIONS

`<file>` the (optional) name of a GSS file containing the model

BUGS
Unknown

DESIRABLE DEVELOPMENTS
The interface is limited to the standard "great-null-tool" provided with the distribution of the Animator. It can be much improved.

NAME

tged – A graphical interface for Task Graphs

SYNOPSIS

tged [<file>] [-ts <int>] [-nolog] [-debug]

DESCRIPTION

It allows to edit task graphs and do various operations, including the generation of graphical representation code and simulation codes. For more information on the functionalities of this interface, see section 3.2.

OPTIONS

- <file> the (optional) name of a file containing the model.
- -ts <int> set the size of tasks. The default is 15.
- -nolog sets the logging style to none, thus preventing the launch of a Logger.
- -debug turns on a verbose debugging mode.

BUGS

Lots

DESIRABLE DEVELOPMENTS

Lots too

tg_read – Parses a TGL file and display network information

SYNOPSIS

`tg_read   [-f <file>] [-h] [-?]`

DESCRIPTION

Reads a Task Graph(coded in TGL) and display debugging information about task/link connections, link delays, depth of dependencies, level analysis and so on.
This information is more or less the one produced by any program of the "tg" family with the `-debug` option.

OPTIONS

`-?` produces usage information.

`-h` produces usage information.

`-f <file>` name of the input TGL file; if none is specified, the graph is read from the standard input.

BUGS

No known bugs.

tg_sim – General purpose Task Graphsimulator

SYNOPSIS

`tg_sim [-f <file>] [-nevt <int>] [-bs <int>] [-debug] [-a] [-h] [-?]`

DESCRIPTION

Simulates an Task Graph(coded in TGL) by iteratively computing its evolution equations.
If the Task Graphis repetitive (*i.e.* contains program-type dependencies), the simulation results in an estimate of the cycle time of the tasks, that is, the average time between two executions.
It the Task Graphis not repetitive, the simulation consist in a number of replicates of the graph, and an estimate of the average completion time of each task is produced.
The simulation uses a circular buffer (see also `eg2c`, p. 49), the size of which may be controlled with the `-bs` option.

OPTIONS

`-f <file>` name of the TGL file; the default is the standard input.

`-bs <int>` controls the size of the circular buffer; 1024 by default.

`-nevt <number>` specifies the number of iterations of the graph simulated; 1000 by default.

`-a` does only the structural analysis, and not the simulation.

`-?` produces usage information.

`-h` produces usage information.

`-debug` produces information on the network read (see `pn_read`).

BUGS

The circular buffer mechanism seems to fail for too small sizes.

DESIRABLE DEVELOPMENTS

Should be tested more throughly, and its performance compared with that of `eg2c`.

Not tested with min/max combinations.
Impossible to redirect the output with an option.
Does not provide trace capability.

tg_crit_path – Critical Path Analysis fot Task Graphs

SYNOPSIS

tg_crit_path [-f <file>] [-o <file>] [-ignore] [-silent] [-mean] [-paths] [-single]
[-sample] [-seed <number>] [-debug] [-h] [-?]

DESCRIPTION

Finds critical paths in a Task Graph. Critical paths are that which lead to the maximal execution time of the tasks. Tasks which finish last are critical. The links which caused them to wait before starting are also critical, as well as the tasks from which they come. And so on recursively.

In the current version, program links cannot be handled. However, even if such links exist, the analysis can be performed using the -ignore option. The task graph analyzed is the one consisting of the same tasks and data (or communication) links.

If the graph contains random delays, the analysis can be performed either using average delays (option -mean), or using samples of the random variables (option -sample). In the latter case, the option -seed may be used in order to change the random sample used. The default behavior is to give the same answer at each call.

The result of the analysis is the total execution time (makespan) of the graph. With the option -paths, the result consists in the critical paths themselves, as a list of nodes and links that may be exploited by graphical interfaces (see *e.g.* tged). There may be several critical paths. With the option -single, only one is produced.

OPTIONS

- -f <file> name of the input TGL file; if none is specified, the graph is read from the standard input.
- -o <file> name of the file containing the reault; if none is specified, the result is written to the standard output.
- -ignore ignores program links; the default is not to perform analysis if some are encountered.
- -silent does not issue warning or error messages.
- -mean uses the averages of random quantities for critical path analysis. The default is -sample.
- -paths produces as result a description of the critical paths suited for being read and displayed by the graphical interface tged. The default is to produce only the total execution time of the graph.
- -single produces (if the -paths option is selected) only one critical path. The default is to list all critical paths.
- -sample draws a sample of random quantities to perform critical path analysis. This is the default.
- -seed <number> uses the integer <number> as a seed for the random number generator.
- -? produces usage information.

`-h` produces usage information.

`-debug` produces information on the network read (see `tg_read`).

BUGS
Unknown

NAME

tg2qnap – TGL to QNAP translation

SYNOPSIS

`tg2qnap [-f <file>] [-o <file>] [-O <number>] [-tmax <number>] [-debug] [-h] [-?]`

DESCRIPTION

Translates an Task Graph(coded in TGL) into an queueing network, (coded in the QNAP2 language).
A queue object is generated for each task and each link, unless optimization is asked (see below). The synchronization constraints are implemented in QNAP using the FISSION/FUSION constructs. To that effect, it is necessary to use auxiliary customer classes. The number of classes is equal to the maximum input degree of tasks.
Optimization consists in trying to remove from the QNAP code unnecessary objects such as queues with 0 service, and reduce the explicit synchronization. For the time being, only data links with no delay and program links are skipped, and the optimization level is ignored.

OPTIONS

`-?` produces usage information.

`-h` produces usage information.

`-debug` produces information on the network read.

`-f <file>` name of the input TGL file; if none is specified, the graph is read from the standard input.

`-o <file>` name of the resulting QNAP file; if none is specified, the result is written to the standard output.

`-O <number>` swithes to the optimization level "number".

`-tmax <number>` specifies the number of iterations of the graph simulated; 1000 by default.

BUGS

There may be something inconsistant about the simulation of task graphs without "program" links (i.e. not periodic).
It is also probable that the model simulated is not FIFO, so that the results may not coincide with that of `tg_sim` or `eg_sim` for instance.

DESIRABLE DEVELOPMENTS

Test the program and the resulting QNAP models.

Confront the results with that of other simulators and compare execution times.
Improve optimization.
Add "or" synchronizations.

Bibliography

[1] F. Baccelli. Ergodic theory of stochastic petri networks. *The Annals of Probability*, 20(1):375–396, 1992.

[2] F. Baccelli and M. Canales. Parallel simulation of stochastic Petri nets using recursive equations. *ACM Transactions on Modeling and Computer Simulation*, 3(1):20–41, January 1993. Rapport INRIA RR 1520, 1993.

[3] F. Baccelli, G. Cohen, G.J. Oslder, and J.P. Quadrat. *Linear Discrete Event Systems of the Laws of Synchronization*. Wiley, 1992.

[4] F. Baccelli and D. Hong. Analytic expansions of $(max, +)$ lyapunov exponents. Technical report, INRIA, 1998.

[5] F. Baccelli, A. Jean-Marie, and Z. Liu. A survey on solution methods for task graph models. In Götz, Herzog, and Rettelbach, editors, *Second QMIPS Workshop*, volume 26 of *Arbeitsberichte der IMMD*, Erlangen, March 1993.

[6] M. Brilman. *Évaluation de performances d'une classe de resources partagées*. PhD thesis, Université Joseph Fourier - Grenoble I, September 1996.

[7] M. Canales. *Simulation parallèle de réseaux de Petri stochastiques*. PhD thesis, Université de Nice-Sophia-Antipolis, February 1994.

[8] B. Gaujal. *Parallélisme et simulation des systèmes à événements discrets*. PhD thesis, Université de Nice-Sophia Antipolis, June 1994.

[9] B. Gaujal, A. Jean-Marie, and J. Mairesse. Minimal representation of uniform recurrence equations. Technical Report 2568, INRIA, June 1995.

[10] L. Guion and V. Guion. Interface graphique pour un outil d'analyse de réseaux de Petri. Master's thesis, DESS-ISI, Université de Nice-Sophia-Antipolis, 1993.

[11] B. Heidergott. A differential calculus for random matrices with applications to (max,plus)-linear stochastic systems. Technical Report 98-10, Delft University, 1998.

[12] A. Jean-Marie. The waiting time distribution in Poisson-driven deterministic systems. RR 3083, INRIA, January 1997.

[13] Z. Liu. Performance bounds of stochastic timed petri nets by linear programming approach. In preparation., September 1994.

[14] J. Mairesse. A graphical representation for matrices in the $(\max, +)$ algebra. RR 2078, INRIA, October 1993.

[15] H. Rakotoarisoa. *Simulation distribuée de réseaux de files d'attente*. PhD thesis, Université de Nice-Sophia-Antipolis, February 1994.

[16] H. Rakotoarisoa and P. Mussi. Parseval: Parallélisation sur réseaux de *transputers* de simulations pour l'evaluation de performances. RT 131, INRIA, September 1991.

[17] G. Siegel. *Un environnement pour la programmattion de simulations à événements discrets*. PhD thesis, Université de Nice-Sophia-Antipolis, September 1997.

Contents

Index