Dipartimento di Informatica
Università del Piemonte Orientale "A. Avogadro"
Corso Borsalino 54, 15100 Alessandria

PROGETTO MIUR - ISIDE

Stochastic modeling, analysis techniques and tools for dependable reactive systems

TECHNICAL REPORT - TR-INF-2002-10-06-UNIPMN
July 2002

drawnet2sharpe & sharpe2astra - User's manual

Author: Daniele Codetta Raiteri

Contents

Introduction

This manual is about two translators, **drawnet2sharpe** [1] and **sharpe2astra** [1], which allow to design a fault tree [2] (even in parametric form [3]) with the graphic tool named *DrawNet* [4] and analyze it with *Sharpe* [5] and *Astra* [6], tools for the automatic analysis of fault trees.

drawnet2sharpe allows to translate the file containing the fault tree designed with Drawnet from its XML format to the *Sharpe*'s formalism; on the other hand, **sharpe2astra** translates a file for *Sharpe* into the *Astra*'s formalism.

Architecture

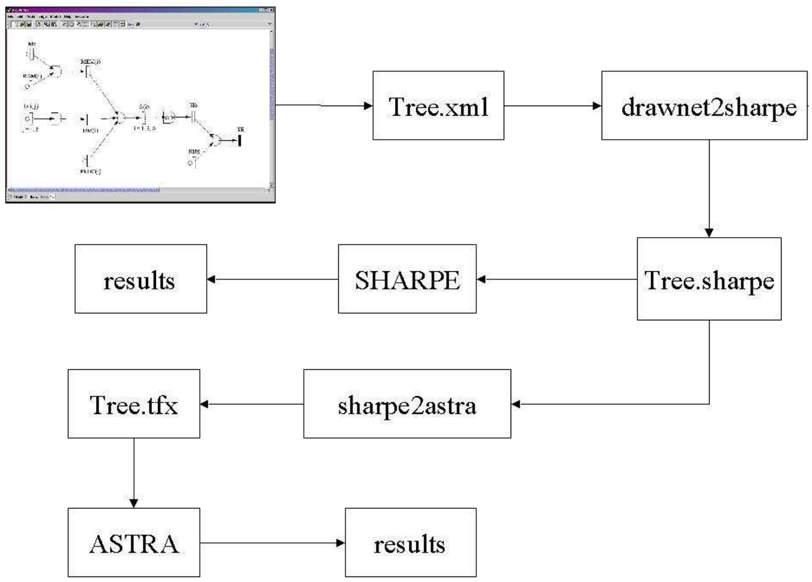


Figure 1: Software architecture

Fig. 1 shows the software architecture: *DrawNet* saves the tree in a XML file which is taken in input by **drawnet2sharpe**; this translator generates the relative file for *Sharpe*; this file can be passed to *Sharpe* for the analysis of the tree described inside or to **sharpe2astra** for the translation into the *Astra*'s formalism; the file generated by **sharpe2astra** can now be passed to *Astra*.

1 drawnet2sharpe

1.1 DrawNet

DrawNet [4] is a graphic tool able to design various types of graph and among them fault trees; *DrawNet* takes in input at the moment of its execution a XML file containing a formalism describing the features of the type of graph; in the case of parametric fault trees (PFT) [3], the formalism is the following:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE formalism SYSTEM "formalism.dtd">
<formalism parent="" name="PFT">

<propertyType name="Title" default=""/>

<nodeType parent="" name="Event0">
<propertyType name="Label" default=""/>
<propertyType name="Parameters" default=""/>
</nodeType>

<nodeType parent="Event0" name="Event"/>

<nodeType parent="Event0" name="BasicEvent">
         <propertyType name="Distribution" default="ALL EXP 1.0"/>
</nodeType>

<nodeType parent="Event" name="ReplicatorEvent">
<propertyType name="PList" default=""/>
<propertyType name="DeclaredParameters" default=""/>
<propertyType name="PredSWN" default=""/>
</nodeType>

<nodeType parent="BasicEvent" name="BasicReplicatorEvent">
<propertyType name="PList" default=""/>
<propertyType name="DeclaredParameters" default=""/>
<propertyType name="PredSWN" default=""/>
</nodeType>

<nodeType parent="" name="TopEvent">
<propertyType name="Label" default=""/>
</nodeType>

<nodeType parent="" name="Gate"/>

        <nodeType parent="Gate" name="And"/>

<nodeType parent="Gate" name="Or"/>
```

```
<nodeType parent="" name="G2of3"/>

<nodeType parent="Gate" name="KofN">
<propertyType name="K" default="2"/>
<propertyType name="N" default="3"/>
</nodeType>

<nodeType parent="" name="ColorClass">
<propertyType name="SubClassList" default=""/>
<propertyType name="Ordered" default="false"/>
</nodeType>

<nodeType parent="" name="ColorSubClass">
<propertyType name="ElementList" default=""/>
</nodeType>

<nodeType parent="" name="ParameterType">
<propertyType name="ParameterName" default=""/>
<propertyType name="ParameterColor" default=""/>
</nodeType>

<edgeType parent="" name="Arc">
<constraint fromType="Gate" fromCardinality="1"
 toType="Event" toCardinality="1"/>
<constraint fromType="G2of3" fromCardinality="1"
 toType="Event" toCardinality="1"/>
<constraint fromType="Gate" fromCardinality="1"
 toType="TopEvent" toCardinality="1"/>
<constraint fromType="G2of3" fromCardinality="1"
 toType="TopEvent" toCardinality="1"/>
<constraint fromType="Event0" fromCardinality=""
 toType="Gate" toCardinality=""/>
<constraint fromType="Event0" fromCardinality=""
 toType="G2of3" toCardinality="3"/>
</edgeType>

</formalism>
```

The types of nodes a PFT may contain are shown in Fig. 2 and they are:

- Event
- Basic Event
- Replicator Event
- Basic Replicator Event

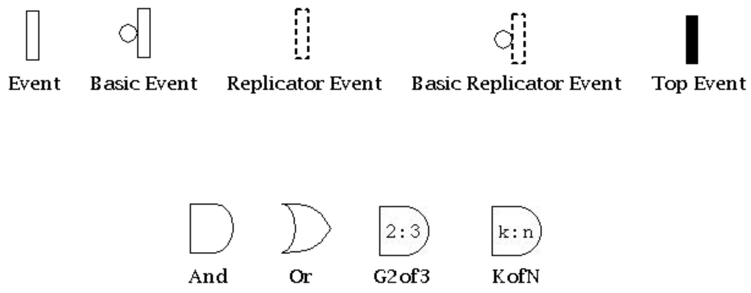


Figure 2: Types of node of a PFT

- Top Event
- And gate
- Or gate
- 2 out of 3 gate (2/3)
- K out of N gate (K/N)

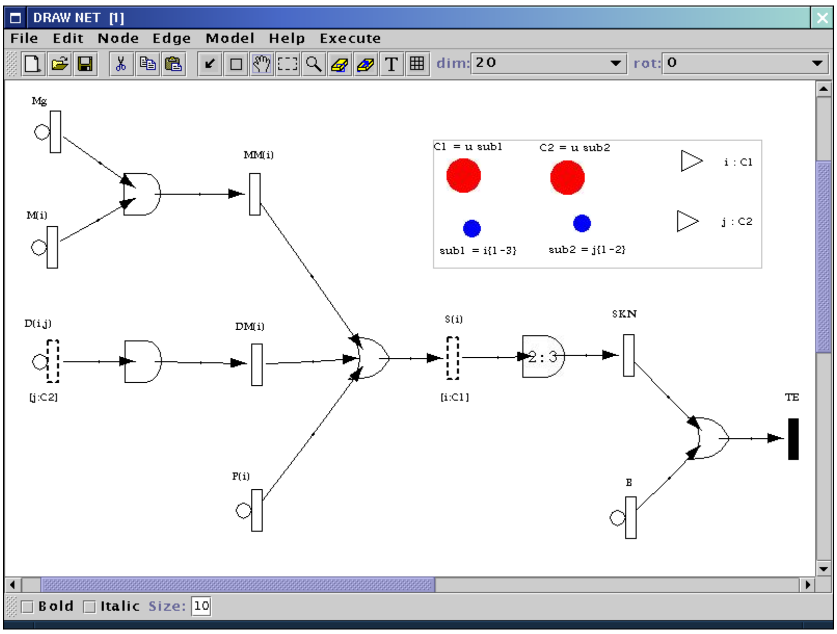
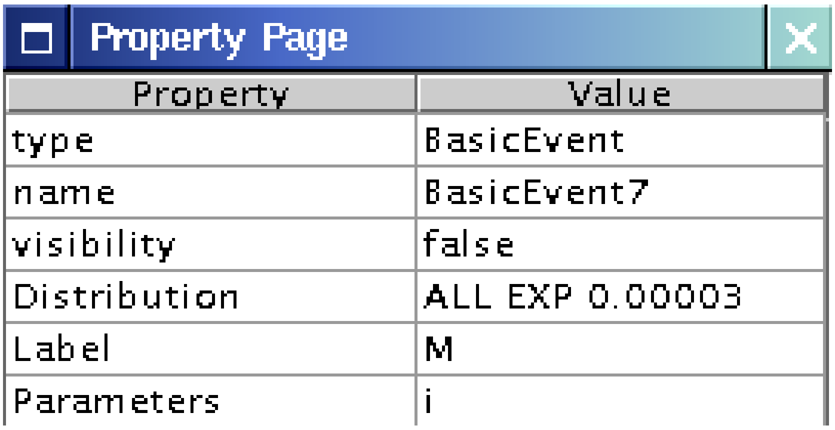


Figure 3: PFT for the multiprocessor system designed with DrawNet

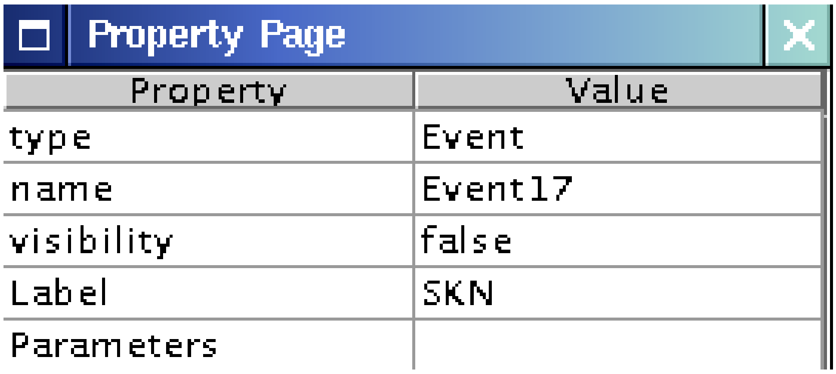
DrawNet is composed by two windows: the main window allows to design the tree and contains menus (Fig. 3) while the window named "*Property Page* " allows the user to define the features of nodes filling some fields displayed inside this window (Fig. 4, 5, 6, 7,).



Property Page

Property	Value
type	BasicEvent
name	BasicEvent7
visibility	false
Distribution	ALL EXP 0.00003
Label	M
Parameters	i

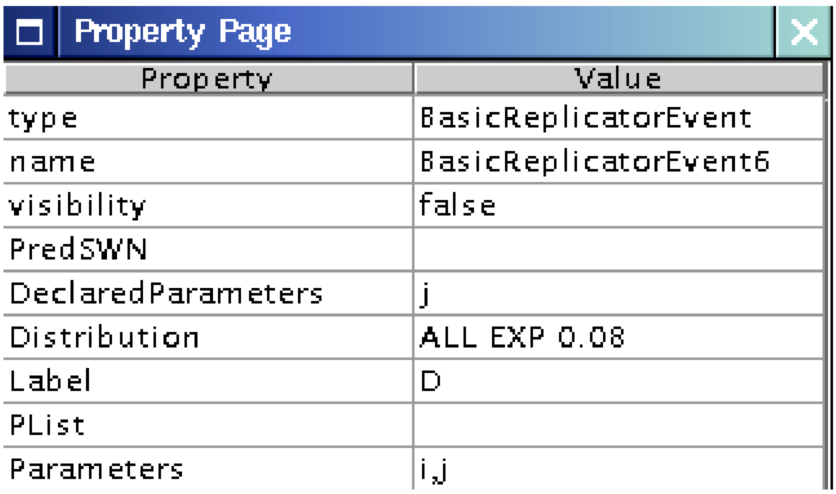
Figure 4: Property Page window for a Basic Event



Property Page

Property	Value
type	Event
name	Event17
visibility	false
Label	SKN
Parameters	

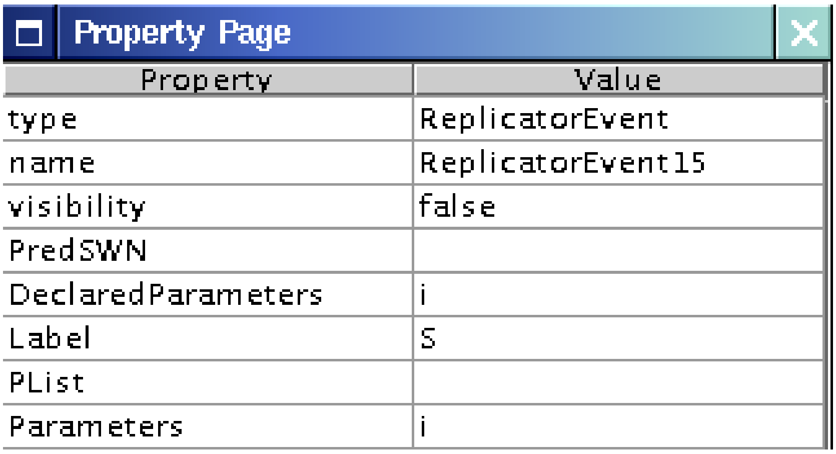
Figure 5: Property Page window for an Event



Property Page

Property	Value
type	BasicReplicatorEvent
name	BasicReplicatorEvent6
visibility	false
PredSWN	
DeclaredParameters	j
Distribution	ALL EXP 0.08
Label	D
PList	
Parameters	i,j

Figure 6: Property Page window for a Basic Replicator Event



Property Page

Property	Value
type	ReplicatorEvent
name	ReplicatorEvent15
visibility	false
PredSWN	
DeclaredParameters	i
Label	S
PList	
Parameters	i

Figure 7: Property Page window for a Replicator Event

1.2 A multiprocessor system

Let's take as example the tree of Fig. 3; it regards a multiprocessor system composed by 3 subsystems, as shown in Fig. 8, and each of them consists of:

- a cpu,
- an internal memory,
- 2 hard disks;

Furthermore, a communication bus connects every subsystem to the other ones and there is a shared memory.

Clicking on the *Execute* menu and then on *Solve*, it is possible to generate the XML representation of the PFT. In the case of the multiprocessor system, it is the following:

```
<?xml  version='1.0' encoding='UTF-8' standalone='no'?>
<PFT name='Multiproc' visibility='false' Title='Multiprocessor_System'>

   <ColorClass name='C1' visibility='false' SubClassList='sub1'
    Ordered='false'/>

   <ColorClass name='C2' visibility='false' SubClassList='sub2'
    Ordered='false'/>

   <ColorSubClass name='sub1' visibility='false'
    ElementList='i{1-3}'/>

   <ColorSubClass name='sub2' visibility='false'
    ElementList='j{1-2}'/>

   <ParameterType name='ParameterType4' visibility='false'
    ParameterColor='C1' ParameterName='i'/>
```

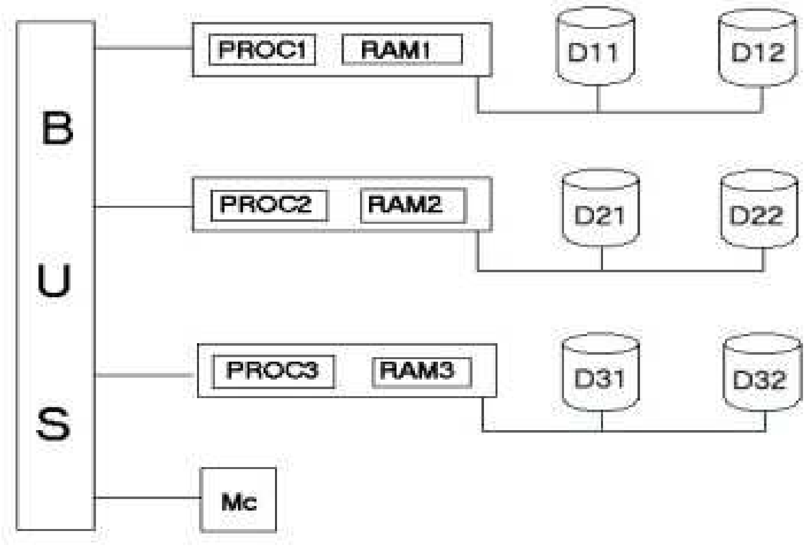


Figure 8: Multiprocessor system scheme

```
<ParameterType name=’ParameterType5’ visibility=’false’
 ParameterColor=’C2’ ParameterName=’j’/>

<BasicReplicatorEvent name=’BasicReplicatorEvent6’ visibility=’false’
 PredSWN=’’ DeclaredParameters=’j’ Distribution=’ALL EXP 0.000008’
 Label=’D’ PList=’’ Parameters=’i,j’/>

<BasicEvent name=’BasicEvent7’ visibility=’false’
 Distribution=’ALL EXP 0.000003’ Label=’M’ Parameters=’i’/>

<BasicEvent name=’BasicEvent8’ visibility=’false’
 Distribution=’ALL EXP 0.000003’ Label=’Mg’ Parameters=’’/>

<And name=’And9’ visibility=’false’/>

<And name=’And10’ visibility=’false’/>

<Event name=’Event11’ visibility=’false’ Label=’MM’ Parameters=’i’/>

<Event name=’Event12’ visibility=’false’ Label=’DM’ Parameters=’i’/>

<BasicEvent name=’BasicEvent13’ visibility=’false’
 Distribution=’ALL EXP 0.000005’ Label=’P’ Parameters=’i’/>
```

```
<Or name='Or14' visibility='false'/>

<ReplicatorEvent name='ReplicatorEvent15' visibility='false'
 PredSWN='' DeclaredParameters='i' Label='S' PList='' Parameters='i'/>

<G2of3 name='G2of316' visibility='false'/>

<Event name='Event17' visibility='false' Label='SKN' Parameters=''/>

<BasicEvent name='BasicEvent18' visibility='false'
 Distribution='ALL EXP 0.000002' Label='B' Parameters=''/>

<Or name='Or19' visibility='false'/>

<TopEvent name='TopEvent20' visibility='false' Label='TE'/>

<Arc name='Arc1' visibility='false' from='BasicEvent8' to='And9'/>

<Arc name='Arc2' visibility='false' from='BasicEvent7' to='And9'/>

<Arc name='Arc3' visibility='false' from='BasicReplicatorEvent6'
 to='And10'/>

<Arc name='Arc4' visibility='false' from='And10' to='Event12'/>

<Arc name='Arc5' visibility='false' from='And9' to='Event11'/>

<Arc name='Arc6' visibility='false' from='BasicEvent13' to='Or14'/>

<Arc name='Arc7' visibility='false' from='Event12' to='Or14'/>

<Arc name='Arc8' visibility='false' from='Event11' to='Or14'/>

<Arc name='Arc9' visibility='false' from='Or14'
 to='ReplicatorEvent15'/>

<Arc name='Arc10' visibility='false' from='ReplicatorEvent15'
 to='G2of316'/>

<Arc name='Arc11' visibility='false' from='G2of316' to='Event17'/>

<Arc name='Arc12' visibility='false' from='Event17' to='Or19'/>

<Arc name='Arc13' visibility='false' from='BasicEvent18' to='Or19'/>

<Arc name='Arc14' visibility='false' from='Or19' to='TopEvent20'/>
```

```
</PFT>
```

The name of the XML file can be assigned by the user clicking on *Model* menu, then on *Properties*; now it is possible in the *Property Page* window to assign the XML file name and optionally, a comment to the fault tree.

1.3 The translation from XML to Sharpe's formalism

Executing **drawnet2sharpe** by mean of the command

```
drawnet2sharpe file_name1.xml file_name2
```

drawnet2sharpe translates the XML file into the formalism of *Sharpe*; *file_name1.xml* is the name of the XML file generated by *DrawNet* and *file_name2* is the name of the file for *Sharpe* the user wants to assign to.

In the case of the multiprocessor system, we can launch **drawnet2sharpe** in this way:

```
drawnet2sharpe Multiproc.xml Multiproc.sharpe
```

The result is the file named Multiproc.sharpe:

```
*Multiprocessor_System

ftree Multiproc

repeat Mg exp(v2)
repeat B exp(v4)
repeat P1 exp(v3)
repeat D11 exp(v1)
repeat D12 exp(v1)
repeat M1 exp(v2)
repeat P2 exp(v3)
repeat D21 exp(v1)
repeat D22 exp(v1)
repeat M2 exp(v2)
repeat P3 exp(v3)
repeat D31 exp(v1)
repeat D32 exp(v1)
repeat M3 exp(v2)

and DM1 D11 D12
and MM1 Mg M1
or S1 P1 DM1 MM1
and DM2 D21 D22
and MM2 Mg M2
or S2 P2 DM2 MM2
and DM3 D31 D32
```

```
and MM3 Mg M3
or S3 P3 DM3 MM3
kofn SKN 2,3, S1 S2 S3
or TOP SKN B
end

bind
v1 0.000008
v2 0.000003
v3 0.000005
v4 0.000002
end

expr mean(Multiproc)
expr variance(Multiproc)

end
```

1.3.1 drawnet2sharpe stages

The translation from XML to *Sharpe*'s formalism consists of eight stages (see Fig. 10):

1. **XML file parsing** (parser's states for the main types of node are shown in Fig. 9, only exponential distribution about basic events has been considered);

2. **parameters analysis**: for each parameter, **drawnet2sharpe** verifies how many values and which values the parameter can assume and which events the parameter refers to;

3. **linking**: the fault tree structure is rebuilt in memory examining *From* and *To* fields of each arc declared in the XML file;

4. **rates research**: in this stage, all the basic events' failure rates are collected to be explicitly indicated in the *Sharpe* file after the keyword *bind*;

5. **unfolding** : transformation from PFT to FT

6. **gates bottom-up sorting**: *Sharpe* requires that every event must be declared before being referred as input of a gate;

7. ***Sharpe* file generation**: in the *Sharpe* file, **drawnet2sharpe** writes all the necessary declarations in this order:

   - the comment to the tree (if it appears in the XML file),
   - basic events with their distribution (*exponential*),
   - gates (in the order established in the previous stage and indicating for each gate its type, its output event's name and its input events' names),
   - basic events' failure rates (after the keyword *bind*),

- measures to be computed (as default, the mean and the variance of the system failure time); other measures can be requested to *Sharpe* by adding some lines to file; for example

  ```
  eval(Multiproc) 0 5000 250
  ```

  for the system failure probability at time **t** varying from 0 to 5000 hours with a step 250.

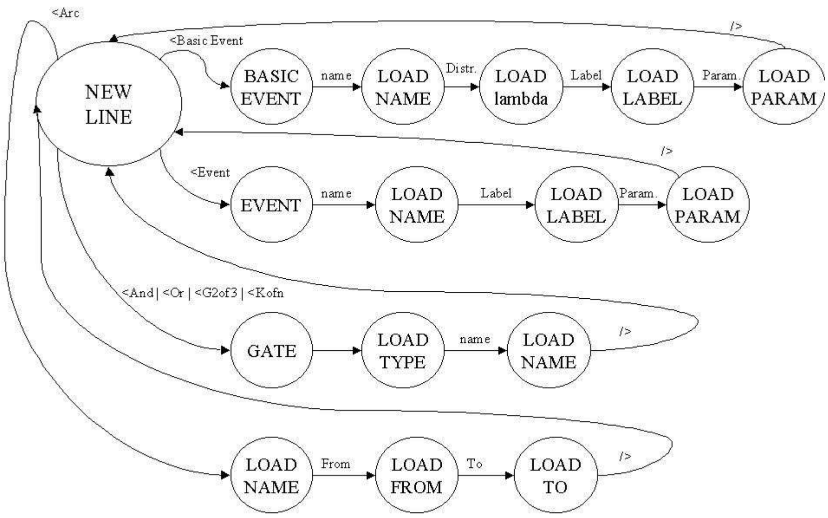


Figure 9: drawnet2sharpe parser's states

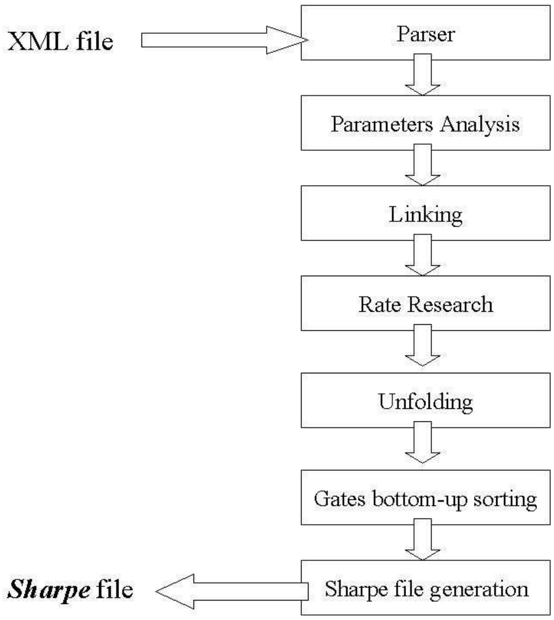


Figure 10: drawnet2sharpe stages

2 sharpe2astra

sharpe2astra can be executed by mean of the following command

```
sharpe2astra file_name1 file_name2.tfx
```

where *file_name1* is the name of the file for *Sharpe* taken in input by **sharpe2astra** and *file_name2.tfx* is the name of the ouput file containing the tree described in the *Astra*'s formalism.

In the case of the multiprocessor system, **sharpe2astra** can be executed in this way:

```
sharpe2astra Multiproc.sharpe Multiproc.tfx
```

The effect of this command is the generation of this file:

```
*Multiprocessor_System
[EVENTS]
MG 0.000003 0 0 0 0 "sharpe2astra"
B 0.000002 0 0 0 0 "sharpe2astra"
P1 0.000005 0 0 0 0 "sharpe2astra"
D11 0.000008 0 0 0 0 "sharpe2astra"
D12 0.000008 0 0 0 0 "sharpe2astra"
M1 0.000003 0 0 0 0 "sharpe2astra"
P2 0.000005 0 0 0 0 "sharpe2astra"
D21 0.000008 0 0 0 0 "sharpe2astra"
D22 0.000008 0 0 0 0 "sharpe2astra"
M2 0.000003 0 0 0 0 "sharpe2astra"
P3 0.000005 0 0 0 0 "sharpe2astra"
```

```
D31 0.000008 0 0 0 0 "sharpe2astra"
D32 0.000008 0 0 0 0 "sharpe2astra"
M3 0.000003 0 0 0 0 "sharpe2astra"
[GATES]
TOP OR GSKN B "sharpe2astra"
GSKN 2/3 GS1 GS2 GS3 "sharpe2astra"
GS3 OR P3 GDM3 GMM3 "sharpe2astra"
GMM3 AND MG M3 "sharpe2astra"
GDM3 AND D31 D32 "sharpe2astra"
GS2 OR P2 GDM2 GMM2 "sharpe2astra"
GMM2 AND MG M2 "sharpe2astra"
GDM2 AND D21 D22 "sharpe2astra"
GS1 OR P1 GDM1 GMM1 "sharpe2astra"
GMM1 AND MG M1 "sharpe2astra"
GDM1 AND D11 D12 "sharpe2astra"
$END
```

2.1 sharpe2astra stages

The translation from *Sharpe*'s formalism to *Astra*'s formalism consists of five stages (see Fig. 12):

1. ***Sharpe* file parsing** (parser's states are shown in Fig. 11, only exponential and constant distributions about basic events have been considered);

2. **events replication**: in *Sharpe*'s formalism it is possible to write declarations like these:

   ```
   and DM1 D1 D1
   kofn OUT 2,3, IN
   ```

   In the first line, a gate with two input events equal to $D1$ is declared, while in the second one there is the declaration a 2 out of 3 gate with three input events equal to IN. In the first case $D1$ must be replicated in $D11$ $D12$, while in the second case, IN must be replicated in $IN1$ $IN2$ $IN3$. This stage is not necessary if the *Sharpe* file is generated with **drawnet2sharpe**, since this translator never produce declarations of this kind which could be encountered in "hand written" *Sharpe* file;

3. **events' names conversion**: *Astra* requires that events' names must be composed only by capital letters and if an event is the output event of a gate its name must have G as first character; so, for example, Mg becomes MG and SKN becomes $GSKN$;

4. ***Astra* compatibility check**: in this stage, some checks are executed; they concern:

   - the events' names lenghts (they must be less or equal to 10 characters),

- the fault tree comment lenght (it must be less or equal to 40 characters),
- the presence of basic events with fault probability equal to 0,
- the number of gates input events (for each gate there must not be more than 8 input events)
- the total number of nodes the tree contains (it must be less or equal to 4000)

5. ***Astra* file generation**: in the *Astra* file, **sharpe2astra** writes the comment to the tree (if it appears in the *Sharpe* file) followed by the basic events declarations (after the keyword [EVENTS]), then the gates declarations (after the keyword [GATES]; the first gate to be declared is that one with TOP (Top Event) as output event. The declaration of a basic event consists of the name of the basic event followed by some values and a comment; the first value is the failure rate λ. The declaration of a gate consists of its output event's name, its type, its input events' names and a comment. The file end with the keyword [$END].

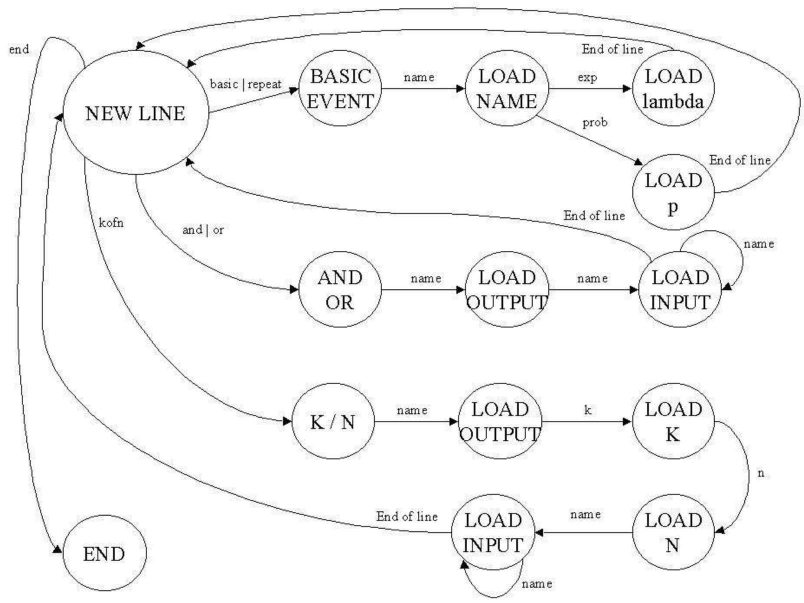


Figure 11: sharpe2astra parser's states

2.2 Results comparison

Let's refer to the multiprocessor system and request *Sharpe* the system fault probability at time **t** varying from 0 to 5000 hours with a step of 250; *Sharpe* returns the following results:

```
system Multiproc
  t              F(t)
```

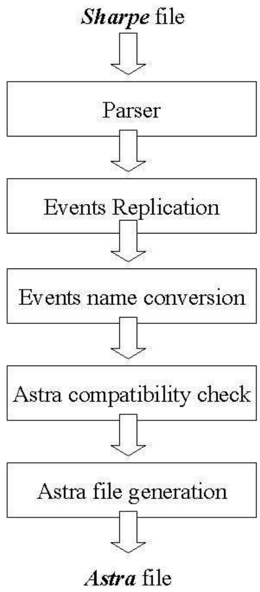


Figure 12: sharpe2astra stages

```
0.00000000 e+00  0.00000000 e+00
2.50000000 e+02  5.04585778 e-04
5.00000000 e+02  1.01843528 e-03
7.50000000 e+02  1.54168464 e-03
1.00000000 e+03  2.07446715 e-03
1.25000000 e+03  2.61691326 e-03
1.50000000 e+03  3.16915058 e-03
1.75000000 e+03  3.73130386 e-03
2.00000000 e+03  4.30349500 e-03
2.25000000 e+03  4.88584307 e-03
2.50000000 e+03  5.47846425 e-03
2.75000000 e+03  6.08147190 e-03
3.00000000 e+03  6.69497653 e-03
3.25000000 e+03  7.31908577 e-03
3.50000000 e+03  7.95390444 e-03
3.75000000 e+03  8.59953452 e-03
4.00000000 e+03  9.25607513 e-03
4.25000000 e+03  9.92362258 e-03
4.50000000 e+03  1.06022704 e-02
4.75000000 e+03  1.12921092 e-02
5.00000000 e+03  1.19932268 e-02
```

The first column indicates the value of time, while the second one indicates the system fault probability at that time.

Asking *Astra* the same kind of results, it returns the values shown in Fig. 13 where the first column indicates the value of time, the second one and the third one are two upper bounds to system fault probability and the last one is a lower bound.

The results calculated by *Sharpe*, instead, are exact and it is easy to verify that they are included between the lower bound and the upper bounds returnded by *Astra*.

List of system data as function of time-points

	Time (hours)	Unavailability	Unav. (Proschan)	Unav. (lower bound
1	0.00000000	0.000000E+00	0.000000E+00	0.000000E+00
2	250.00000000	5.045921E-04	5.045898E-04	5.045838E-04
3	500.00000000	1.018487E-03	1.018468E-03	1.018419E-03
4	750.00000000	1.541861E-03	1.541796E-03	1.541630E-03
5	1000.00000000	2.074892E-03	2.074736E-03	2.074336E-03
6	1250.00000000	2.617755E-03	2.617448E-03	2.616655E-03
7	1500.00000000	3.170626E-03	3.170091E-03	3.168699E-03
8	1750.00000000	3.733681E-03	3.732823E-03	3.730579E-03
9	2000.00000000	4.307092E-03	4.305800E-03	4.302400E-03
10	2250.00000000	4.891035E-03	4.889178E-03	4.884266E-03
11	2500.00000000	5.485683E-03	5.483111E-03	5.476275E-03
12	2750.00000000	6.091208E-03	6.087752E-03	6.078524E-03
13	3000.00000000	6.707782E-03	6.703253E-03	6.691104E-03
14	3250.00000000	7.335578E-03	7.329764E-03	7.314104E-03
15	3500.00000000	7.974765E-03	7.967435E-03	7.947608E-03
16	3750.00000000	8.625514E-03	8.616412E-03	8.591697E-03
17	4000.00000000	9.287995E-03	9.276843E-03	9.246449E-03
18	4250.00000000	9.962376E-03	9.948873E-03	9.911937E-03
19	4500.00000000	1.064883E-02	1.063264E-02	1.058823E-02
20	4750.00000000	1.134751E-02	1.132830E-02	1.127540E-02
21	5000.00000000	1.205860E-02	1.203598E-02	1.197350E-02

Figure 13: System fault probability calculated by Astra

References

[1] Daniele Codetta Raiteri. Sviluppo di formalismi per alberi dei guasti con nodi dipendenti e riparabili. Master's thesis, Università degli Studi del Piemonte Orientale "Amedeo Avogadro", A. A. 2001-2002.

[2] W.G. Schneeweiss. *The Fault Tree Method.* LiLoLe Verlag, 1999.

[3] A.Bobbio G.Franceschinis R.Gaeta L.Portinale. Parametric fault-tree for the dependability analysis of redundant systems and its high level petri nets semantics.

[4] http://www.di.unito.it/ greatspn/drawnet. DrawNet home page.

[5] R.A.Sahner K.S.Trivedi A.Puliafito. *Performance And Reliability Analysis Of Computer Systems; An Example-Based Approach Using the SHARPE Software Package.* Kluwer Academic Publishers, 1996.

[6] S.Contini. Astra - knowledge handbook - logical and probabilistic analysis method, 1998.

Index