Manuel d'utilisation de GénéSyst

Laboratoire LSR, équipe VASCO
681 rue de la Passerelle, BP 72 38402 St Martin d'hères Cedex

Version 1.3.2 du manuel - Compilée le 7 novembre 2005
Pour la version 0.1.7 de GénéSyst

Historique des versions

Version 1 (2002) : première rédaction de ce manuel (Nicolas Stouls - Nicolas.Stouls@imag.fr).

Version 1.0.1 (04/02/2004) : Relecture et corrections diverses.

Version 1.1 (12/05/2004) : Prise en compte des modifications de GénéSyst.

Version 1.2 (31/08/2004) : Rajout des descriptions de l'interface graphique et des possibilités de formats de sortie.

Version 1.2.1 (01/10/2004) : Relecture et corrections diverses.

Version 1.3 (20/05/2005) : Prise en compte de la version 0.1.6.2a de GénéSyst.

Version 1.3.1 (07/11/2005) : Prise en compte de la version 0.1.6.3a de GénéSyst.

Version 1.3.2 (17/10/2005) : Prise en compte de la version 0.1.7 de GénéSyst.

Contents

Introduction 1

1.1 Configuration requise

Ce programme ne tourne que sous UNIX ou LINUX (Testé uniquement sur les distributions FEDORA). Son installation nécessite d'avoir préalablement installé les logiciels suivants :

- La BoB, outil gratuit développé au LSR (pour la génération des obligations de preuve)
  http ://www-lsr.imag.fr/Les.Personnes/Nicolas.Stouls/
- L'*AtelierB*, outil payant développé par Clearsy (pour la vérification des obligations de preuve)
  http ://www.clearsy.com/
  OU B4free, l'outil gratuit (pour les possesseurs de l'*AtelierB* ou les universitaires), diffusé à l'adresse :
  http ://www.b4free.com/
- Sous LINUX seulement (car fournit avec la distribution UNIX) :
  GraphViz, développé par AT&T (pour la génération des graphiques résultats à partir des fichiers de sortie produits)
  http ://www.graphviz.org/

Une copie du logiciel GénéSyst peut être récupérée à l'adresse suivante :
http ://www-lsr.imag.fr/Les.Personnes/Nicolas.Stouls/

1.2 Description générale

GénéSyst est un programme Java diffusé sous les termes du contrat de licence Cecill-V2 [1]. Ce logiciel ne peut être diffusé autrement car il inclut le programme Grappa [2] et nécessite l'installation de l'application GraphViz [3], qui sont tous deux diffusés sous la licence GPL.

GénéSyst [PS04, BC00, Cav00, Ham02] est un outil permettant de générer un système de transitions étiquetées représentant exactement le comportement d'un système B événementiel [Abr96c, Abr96b, AM98b, Abr96a, AM98a, BP03, But99, BW96] (Spécification ou raffinement).

Pour cela, il se base sur un parseur B, le jBTools [VTH02], qui lui sert à charger les systèmes en mémoire, et sur une bibliothèque de manipulation des structures B, la **BoB**[4], qui lui permet de

[1] http ://www.cecill.info/
[2] http ://www.research.att.com/~john/Grappa/
[3] http ://www.graphviz.org/
[4] www-lsr.imag.fr/B/Documents/CNAM-2002-06-13/Resumes/Bert.pdf

générer des obligations de preuve. Ensuite, ces dernières sont donnée au prouveur automatique de l'*AtelierB* développé par ClearSy[5].

Le résultat de GénéSyst est un automate décoré. Celui-ci peut, à l'heure actuelle, être produit sous trois formes différentes :
- au format *DOT*, supporté par l'outil GraphViz [GN99] de AT&T.
- au format *HTML*, produit en utilisant les fonctionnalités de GraphViz.
- au format *GXL* (Graph eXchange Language), basé sur le XML, et reconnu par de nombreux outils (Format non encore testé).

Historiquement, le premier format a avoir été supporté était le *BCG*. Mais l'impossibilité de pouvoir représenter des automates hiérarchiques ou d'étiqueter les états avec des prédicats, nous a poussé à ne plus supporter ce format.

Cet outil est basé sur les études préalables de D. Bert et F. Cave [BC00, Cav00], S. Hamdane [Ham02] et N. Stouls et M-L Potet[PS04, MPS04, BPS05]

Enfin, son fonctionnement général peut être schématisé comme montré sur la figure 1.1.

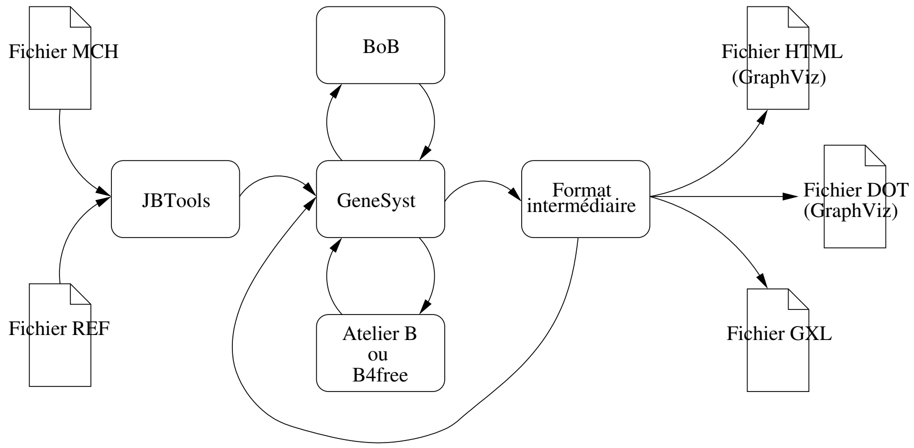

Figure 1.1: Schématisation des interactions de GénéSyst avec les autres outils

1.3 Installation

1.3.1 Prérecquis

- L'*AtelierB* ou l'outil B4free doivent être installés pour que l'outil GénéSyst puisse fonctionner : http ://www.b4free.com/
- Récupérer et installer la BoB :
  http ://www-lsr.imag.fr/Les.Personnes/Nicolas.Stouls/
- **(Installation linux seulement)**
  Si l'application graphiz n'est pas installée alors récupérer et installer le package disponible ici :
  http ://dag.wieers.com/packages/graphviz/

[5] http ://www.clearsy.com/html/clearsy.htm

1.3.2 Méthode d'installation

1. Récupérer le fichier GeneSyst.tar.gz

   http ://www-lsr.imag.fr/Les.Personnes/Nicolas.Stouls/

2. Dans le répertoire d'installation de la BoB (que nous appellerons *BoBdir* pour des raisons évidentes de comodité), faire :

   ```
   tar -xzvf GeneSyst.tar.gz
   ```

3. Vous devez obtenir un répertoire *GeneSyst* contenant 3 (ou 4) répertoires :
   - Class : Le programme en lui même
   - Licences : Contenant le contrat de licence CeCILL sous lequel est diffusé cet outil
   - icons : Les icones de GénéSyst
   - *GraphViz : Seulement dans la version UNIX livrée avec GraphViz*

   et 4 fichiers :
   - README.txt : un fichier qui rappel quelques informations sur l'utilisation de GeneSyst. Ce fichier ne peut pas être utilisé comme manuel. Il serait incompréhensible !
   - Manuel.ps.gz : le manuel de GénéSyst.
   - Install.sh : le script d'installation
   - INSTALL.txt : un manuel d'installation identique à cette section du manuel.

4. Votre variable *CLASSPATH* doit accéder au répertoire des classes de GénéSyst (GeneSyst/Class/). Pour cela, ajoutez la séquence suivante dans votre fichier .login (ou équivalent) à la suite de la séquence pour la détection des classes de la BoB :

   en csh :
   ```
   setenv CLASSPATH "$CLASSPATH":$BOBPATH/GeneSyst/Class/.
   setenv CLASSPATH "$CLASSPATH":$BOBPATH/GeneSyst/Class/GUI/.
   setenv CLASSPATH "$CLASSPATH":$BOBPATH/GeneSyst/Class/Grappa/.
   ```

   en bash :
   ```
     CLASSPATH="$CLASSPATH:$BOBPATH/GeneSyst/Class/"
     CLASSPATH="$CLASSPATH:$BOBPATH/GeneSyst/Class/GUI/."
     CLASSPATH="$CLASSPATH:$BOBPATH/GeneSyst/Class/Grappa/."
   export CLASSPATH
   ```

5. Lancez le script d'installation install.sh.

   Attention ! Vous devez lancer ce script depuis l'intérieur du dossier GeneSyst et taper la commande :

   ```
   /bin/sh ./install.sh
   ```

   Suivez les consignes qui apparaissent alors à l'écran. Ce script ne fait que générer et compiler le fichier :

   ```
   GeneSyst/Class/ConfigurationGeneSyst.java
   ```

   Celui-ci contient quelques variables de configuration de GénéSyst. Donc, en cas de problème d'utilisation du script, vous pouvez le relancer autant de fois que vous voulez, ou même le contourner et éditer manuellement le fichier.

6. *(Pour la version UNIX avec GraphViz uniquement)* Votre variable *PATH* doit accèder au répertoire des binaires exécutables de GraphViz. Pour cela, ajoutez la séquence suivante dans votre fichier .login (ou équivalent) :

```
set path=($BOBPATH/GeneSyst/GraphViz/bin ${path})
setenv MANPATH "$MANPATH":$BOBPATH/GeneSyst/GraphViz/man
```

Pour toute demande d'informations complémentaires s'adresser à :

Nicolas.Stouls@imag.fr

ou aller sur la page :

http ://www-lsr.imag.fr/Les.Personnes/Nicolas.Stouls/

Algorithmes implantés

2

Les fondements théoriques de GénéSyst ne sont pas abordés dans ce manuel. En revanche, ils sont détaillés dans [PS04, BPS05]. Dans ce chapitre, nous allons simplement décrire brièvement l'organisation des calculs effectués permettant l'obtention d'un système de transitions étiquetées.

2.1 Cas d'une spécification abstraite

Une fois la description du système chargée en mémoire par la jBTools, l'ensemble des états est chargé à partir de la clause assertion du système. En effet, l'utilisateur doit préciser, dans l'assertion, les états qu'ils souhaite voir apparaître sur l'automate. Ces états doivent alors être décrits sous la forme d'une disjonction des différents prédicats définissant chacun des états. Voici un exemple concret définissant les trois états $Feu = Vert$, $Feu = Rouge$, $Feu = Orange$:

ASSERTIONS

$$Feu = Vert \ \vee \ Feu = Rouge \ \vee \ Feu = Orange$$

Les variables qui ne sont pas précisées dans un état peuvent y prendre toutes les valeurs autorisées par l'invariant.

Ensuite, on recherche les états initiaux. C'est à dire que, pour chaque état E, on va chercher à vérifier successivement les deux obligations de preuve suivantes :

$$(1) \quad \langle Init \rangle E$$
$$(2) \quad [Init] \neg E$$

Remarque : *Si la première OP est vérifiée alors la seconde n'est pas calculée.*
Remarque : *La méthode de vérification d'une obligation de preuve sera décrite dans la section 7.*

Si (1) est vérifiée alors nous considérons que toutes les valuations possibles de l'état E sont initiales. Sinon, si (2) est vérifiée alors nous considérons qu'aucune des valuations possibles de l'état E n'est initiale. Enfin, si ni (1) ni (2) ne sont vérifiées alors nous considérons que seules certaines valuations de l'état E sont initiales.

Ensuite, nous construisons l'ensemble des états atteignables. Cette construction est réalisée de manière inductive sur les états atteignables et s'arrête lorsqu'il n'y a plus de nouvel état atteint. Au départ, seuls les états initiaux sont atteignables. Cette construction nécessite de calculer, pour chaque état, l'ensemble des transitions qui en partent. En effet, L'ensemble des états atteignables depuis un état E est l'ensemble des états atteignables depuis E par au moins une transition.

La recherche de la déclenchabilité et de l'atteignabilité donne lieu au calcul des conditions $D_{(E,Ev)}$ (condition de déclenchabilité) et $A_{(E,Ev,F)}$ (condition d'atteignabilité) qui forment la garde de chacune des transitions. Les obligations de preuves générées sont alors les suivantes :

	Obligations de preuve	Valeur des conditions
(1)	$\forall x \cdot (E \Rightarrow Garde(Ev))$	$D_{(E,Ev)} \equiv \mathsf{true}$
(2)	$\forall x \cdot (E \Rightarrow \neg Garde(Ev))$	$D_{(E,Ev)} \equiv \mathsf{false}$
(3)	$\neg((1) \lor (2))$	$D_{(E,Ev)} \equiv Garde(Ev)$
(4)	$E \land Garde(Ev) \Rightarrow \langle Action(Ev) \rangle F$	$A_{(E,Ev,F)} \equiv \mathsf{true}$
(5)	$E \land Garde(Ev) \Rightarrow \langle Action(Ev) \rangle F$	$A_{(E,Ev,F)} \equiv \mathsf{false}$
(6)	$\neg((4) \lor (5))$	$A_{(E,Ev,F)} \equiv \langle Action(Ev) \rangle F$

Comme pour l'initialisation, l'OP (2) ne sera générée et soumise au prouveur que si l'OP (1) n'est pas vérifiée. Et l'OP (3) ne sera pas calculée, car elle est la conséquence logique de la négation de (1) et (2).

De même pour (4), (5) et (6).

Remarque : *Il est tout de même possible de demander à* GénéSyst *de générer et vérifier aussi les OP* (3) *et* (6)*. Il est ainsi possible d'avoir 4 état pour une condition : Toujours vraie, gardée, impossible ou inconnue*

De manière informelle, on peut décrire ces deux conditions de la manière suivante :

- Déclenchabilité d'un événement Ev depuis un état E_1 : C'est le prédicat décrivant le sous état de E_1 permettant le déclenchement de Ev.
- Atteignabilité d'un état E_2 par un événement Ev depuis un état E_1 : C'est le prédicat décrivant celles des valeurs de la déclenchabilité permettant de mener à E_2. Autrement dit, ce prédicat décrit l'ensemble des valeurs de l'état E_1 qui permettent le déclenchement de Ev et qui mène à E_2.

Cependant, pour des raison de simplification des preuves, la description de ce dernier prédicat se fait sous hypothèse de l'état et de la déclenchabilité.

Il en résulte que :

- l'atteignabilité n'implique pas la déclenchabilité, mais que la conjonction des deux forme la garde "classique" qui orne les automates symboliques.
- la disjonction des conditions d'atteignabilité sur chaque transition d'un événement Ev sur un état E_1 est équivalente à true.
- la condition de déclenchabilité est la même sur chaque transition d'un événement Ev sur un état E_1.

Notons cependant, que le calcul de l'atteignabilité d'un événement vers les différents états se fait de la manière suivante : On cherche d'abord à prouver qu'aucun état n'est atteignable. S'il n'y a qu'un état atteignable alors l'atteignabilité de la transition qui y mène est forcément toujours vraie. Implicitement, il est normal que, si un événement peut se déclenché, il puisse allé quelque part.

2.2 Cas d'un raffinement

La génération d'un système de transitions étiquetées pour un raffinement ne peut être calculé actuellement que si le système de transitions étiquetées de son abstraction est déjà chargé en mémoire. Pour cela il y a deux possibilités : Soi les deux sont calculés dans la même session de GénéSyst, soi un Oracle est donné pour l'abstraction. Tout cela sera décrit plus en détail en section 5.3.

2.2.1 Recherche des états

L'outil va chercher à associer les différents états raffinés au états abstraits. Notons en passant que la syntaxe de définition des états du raffinement n'est pas la même que celle de l'abstraction. En effet, il faut que l'utilisateur indique l'association des différents états Raffiné avec les états abstraits. Par exemple, l'assertion suivante indique que l'état $Feu = Vert$ est décomposé en $Feu = Vert \wedge ParkingVide = \mathsf{true}$ et $Feu = Vert \wedge ParkingVide = \mathsf{false}$ et que les 2 autres états sont restés inchangés :

ASSERTIONS

$$
\begin{aligned}
&((Feu = Vert) \Leftrightarrow ((Feu = Vert \ \wedge\ ParkingVide = \mathsf{true}) \\
&\qquad\qquad\qquad \vee (Feu = Vert \ \wedge\ ParkingVide = \mathsf{false}))) \\
\wedge\ &((Feu = Rouge) \Leftrightarrow (Feu = Rouge)) \\
\wedge\ &((Feu = Orange) \Leftrightarrow (Feu = Orange))
\end{aligned}
$$

La syntaxe attendue pour l'assertion décrivant les états d'un raffinement est donc une conjonction d'équivalences. En outre, le membre de gauche de chacune d'entre elle doit être le prédicat d'un état abstrait et le membre de droite une disjonction de prédicats décrivant des états du raffinement.

Le raffinement décrit dans cet exemple a donc 4 états :
$(Feu = Rouge)$,
$(Feu = Vert \ \wedge\ ParkingVide = \mathsf{true})$,
$(Feu = Vert \ \wedge\ ParkingVide = \mathsf{false})$ et
$(Feu = Orange)$.

Remarque : *Attention à la priorité des opérateurs ! Les parenthèses sont obligatoires autour de chacune des équivalences. Il est d'ailleurs conseillé d'en mettre également autour des membre de gauche et de droite ainsi qu'autour de chacun des états, comme sur l'exemple.*

Afin de ne pas se baser uniquement sur une équivalence syntaxique des obligations de preuve sont générées pour montrées l'équivalence d'entre les états abstraits donnés dans le raffinement et ceux donnés dans l'abstraction. Ainsi, les obligations de preuve suivantes vont être générées pour notre exemple :

	Etat de l'abstraction		*Etat membre de gauche d'une équivalence du raffinement*
(1)	$Feu = Vert$	$\Leftrightarrow$	$(Feu = Vert)$
(2)	$Feu = Vert$	$\Leftrightarrow$	$(Feu = Rouge)$
(3)	$Feu = Vert$	$\Leftrightarrow$	$(Feu = Orange)$
(4)	$Feu = Rouge$	$\Leftrightarrow$	$(Feu = Vert)$
(5)	$Feu = Rouge$	$\Leftrightarrow$	$(Feu = Rouge)$
(6)	$Feu = Rouge$	$\Leftrightarrow$	$(Feu = Orange)$
(7)	$Feu = Orange$	$\Leftrightarrow$	$(Feu = Vert)$
(8)	$Feu = Orange$	$\Leftrightarrow$	$(Feu = Rouge)$
(9)	$Feu = Orange$	$\Leftrightarrow$	$(Feu = Orange)$

Ensuite, comme les OPs (1), (5) et (9) sont les seules à être vérifiées par preuve, alors nous pouvons faire l'association des états abstraits et des états raffinés.

2.2.2 Calcul des transitions

Dans un deuxième temps, l'outil va calculer les transitions du système de transitions étiquetées du raffinement. Seul le calcul vers les états raffinés est pour le moment implanté. C'est à dire que la factorisation des transition n'est pas encore supportée.

En fait, ce calcul est celui déjà implanté pour l'abstraction et ne sera donc pas redécrit. Cependant, certaines simplifications y sont faites. Tout d'abord, le calcul de recherche des états initiaux ne sera effectué que sur les sous-états des états initiaux de l'abstraction. Enfin, pour un événement déjà existant dans l'abstraction, son raffinement ne sera pas calculé sur tous les états mais seulement au niveau des états qu'il relie dans l'abstraction. Cela simplifie les calculs de sa déclenchabilité aussi bien que de son atteignabilité.

Appels de GénéSyst en ligne de commande 3

On peut faire la distinction entre deux modes d'utilisations de l'outil : Pour générer des **STE**, ou pour effacer tous les fichiers temporaires qui ont été laissés dans le dossier courant (sur demande ou par arrêt inopiné de GénéSyst).

3.1 Mode *Génération de système de transitions étiquetées*

Pour entrer dans ce mode, seul un paramètre est obligatoire : le nom du fichier contenant une machine B dont les opérations n'ont pas de pré-conditions mais des gardes. L'appel est donc le suivant :

```
java geneSyst MaMachine.mch
```

A la suite du nom de la machine on peut rajouter des paramètre optionnels dont voici la liste :

`-R` (ou `-Raffinement`) :
Ce flag doit être suivi d'un nom de fichier.
Celui-ci doit contenir un raffinement de la machine.

`-F` (ou `-Force`) [1 par défaut] :
Ce flag doit être suivi d'un entier entre 0 et 3.
Il désigne le niveau de force de preuve à employer dans l'*AtelierB* lors de la résolution des obligations de preuve.

`-M` (ou `-AffichageMinimum`) [False par défaut] :
Ce flag doit être suivi d'un booléen (true / false).
Il permet de minimiser l'affichage en ne détaillant pas chaque opération effectuée.

`-E` (ou `-OPExistentielles`) [False par défaut] :
Ce flag doit être suivi d'un booléen (true / false).
Il permet de générer les obligations de preuve (3) et (6) décrites au chapitre 2.

`-OM` (ou `-OracleMachine`) :
Ce flag doit être suivi d'un nom de fichier.
Celui-ci doit être un oracle pour les obligations de preuve de la machine.

`-OR` (ou `-OracleRaffinement`) :
Ce flag doit être suivi d'un nom de fichier.
Celui-ci doit être un oracle pour les obligations de preuve du raffinement.

`-V` (ou `-VerifierOracle`) [False par défaut] :
: Ce flag doit être suivi d'un booléen (true / false).
S'il vaut `true` et qu'un oracle est passé en paramètre alors l'oracle sera utilisé pour diriger les obligations de preuves générées afin de les minimiser.

`-N` (ou `-Nettoyer`) [True par défaut] :
: Ce flag doit être suivi d'un booléen (true / false).
Il permet de dire à l'outil de laisser les fichiers de travail en place à la fin de la génération.

`-ABP` (ou `-AtelierBenParallele`) [False par défaut] :
: Ce flag doit être suivi d'un booléen (true / false).
S'il vaut `true`, alors un processus de l'*AtelierB* sera conserver en mémoire. Dans le cas contraire, l'*AtelierB* sera relancé à chaque fois qu'une obligation de preuve devra être résolue. Ce deuxième cas semble souvent plus sûr et n'est pas plus coûteux, grace à la mise en cache de l'atelier.

`-D` (ou `-SymboleDéfaut`) ["G" par défaut] :
: Ce flag doit être suivi d'un caractère seul (sans quotes).
Ce caractère sera utilisé pour indiquer qu'une condition du graphique produit n'est pas réductible à `true`.

`-P` (ou `-SymboleProuvé`) [" " par défaut] :
: Ce flag doit être suivi d'un caractère seul (sans quotes).
Ce caractère sera utilisé pour indiquer qu'une condition du graphique produit est réductible à `true`.

`-I` (ou `-SymboleNonProuveé`) ["X" par défaut] :
: Ce flag doit être suivi d'un caractère seul (sans quotes).
Ce caractère sera utilisé (Uniquement dans le cas où le flag -E est à vrai) pour indiquer qu'une condition n'a pas put être déterminée, malgré les 3 OP générée.

`-G` (ou `-QueGenererOP`) [False par défaut] :
: Ce flag doit être suivi d'un booléen (true / false).
S'il vaut `true`, aucune preuve ne sera effectuée et toutes les machines contenant les obligations de preuve seront générées. De plus, le dossier de travail ne sera pas effacer.

`-DOT` [True par défaut] :
: Ce flag doit être suivi d'un booléen (true / false).
S'il vaut `true`, alors le résultat produit sera généré notamment dans le format DOT.

`-HTML` [True par défaut] :
: Ce flag doit être suivi d'un booléen (true / false).
S'il vaut `true`, alors le résultat produit sera généré notamment dans le format HTML. Attention, il est quand même à noter que la génération au format HTML n'est possible que grâce à l'utilisation de l'outil GraphViz de At&T et en passant par le format DOT. Sont utilisation nécessite donc que le format DOT soit aussi sélectionné.

`-GXL` [False par défaut] :
: Ce flag doit être suivi d'un booléen (true / false).
S'il vaut `true`, alors le résultat produit sera généré notamment dans le format GXL.

Exemple d'utilisation :

```
java geneSyst MaMachine.mch -R MonRaff.ref -m false -f 1 -n false -html TRUE
```

Remarque : *L'ordre dans lequel les paramètres sont donnés est totalement indifférent, sauf pour un élément : le premier paramètre doit être soi le nom de la machine à traiter soi le paramètre* `-n`.

3.2 Mode *nettoyage*

Ce mode sert à effacer automatiquement les différentes polutions qui ont pu être générées par l'outil. Pour accèder à ce mode il faut lancer l'outil de la manière suivante :

`java geneSyst -n`

Les éléments qui seront alors effacés sont :
- Les projets qui ont été créés dans l'*AtelierB* par GénéSyst par l'utilisateur courant. Ces projets sont nommés *CalculTransitionsTmpxxx*. Ce nom peut être suivi d'un suffixe numérique.
- Les dossiers qui servent à stocker toutes les informations temporaires pendant le calcul (Répertoires pour l'utilisation de l'*AtelierB*, les fichiers à prouver, etc). Les dossiers en question portent le nom *CalculDesTransitions*. Celui-ci peut-être suivi d'un suffixe numérique.

3.3 Défaut

Si ni le paramètre `-n` ni le nom d'une machine n'est trouvé à la suite de l'appel de la classe java (`java geneSyst`), alors le message d'aide suivant est affiché :

```
Méthode d'appel :
-----------------
java ganaSyst   -{N|Nettoyer} #Argument unique!!!
              | {<nomSystemeAbstrait> [-{R|Raffinement} <nomRaffinement>]
                                      [-{F|Force} {0|1|2|3}]
                                      [-{E|OPExistentielles}{TRUE|FALSE}]
                                      [-{M|AffichageMinimum} {TRUE|FALSE}]
                                      [-{OM|OracleMachine} <nomFichierOracle>]
                                      [-{OR|OracleRaffinement} <nomFichierOracle>]
                                      [-{V|VerifierOracle} {TRUE|FALSE}]
                                      [-{N|Nettoyer} {TRUE|FALSE}]
                                      [-{ABP|AtelierBenParallele} {TRUE|FALSE}]
                                      [-{G|QueGenererOP} {TRUE|FALSE}]
                                      [-{D|SymboleDéfaut} {\33..\255}]
                                      [-{P|SymboleProuvé} {\33..\255}]
                                      [-{I|SymboleNonProuve} {\33..\255}]
                                      [-{DOT} {TRUE|FALSE}]
                                      [-{GXL} {TRUE|FALSE}]
                                      [-{HTML} {TRUE|FALSE}]
                                      [-{DEBUG} {TRUE|FALSE}]
                }

  Exemple :
  ---------
     java geneSyst DISTRI.mch -R DISTRI.ref -M true

  Information :
  -------------
   {\33 ..\255 } est l'ensemble des caractères visibles.
   (tous sauf espace, tabulation, retour chariot, ...)

  Rappel :
  --------
   Si vous passez des symboles reconus par votre SHELL comme étant une
   commande (ex : * signifie l'ensemble des fichiers) alors pensez
   à les précéder d'un anti-slash (ex : \* au lieu de *).
```

Interface graphique 4

Il est désormais possible de faire appel au deux modes d'utilisation de GénéSyst au travers de l'interface graphique réalisée par Hounayda MOHAMED durant l'été 2004. Celle-ci permet de réaliser les même actions que la ligne de commande mais de manière plus agréable. Pour lancer l'interface graphique, il suffit de taper la commande :

```
java AppliGene
```

4.1 Mode *Génération de systèmes de transitions étiquetées*

4.1.1 Lancement de GénéSyst

Pour entrer dans ce mode, il est obligatoire :

- de remplir le champ : `Machine`. Ce champ doit contenir le nom d'un fichier représentant une machine B dont les opérations n'ont pas de pré-conditions mais des gardes : `MaMachine.mch` . Pour entrer le nom du fichier vous pouvez parcourir votre répertoire, mais aussi le rentrer au clavier.
- de préciser le(s) format(s) attendu(s) en sortie dans la liste `Formats de sortie`. Il suffit de cocher sur la case correspondant au(x) format(s) souhaité(s) : *"DOT"*, *"HTML"* ou *"GXL"*.

Ensuite il ne vous reste qu'à exécuter GénéSyst avec le bouton *"Lancer"*.

D'autres options peuvent être précisés :

`Raffinement` :
: Ce champ doit contenir le nom d'un fichier.
Pour entrer le nom du fichier vous pouvez parcourir votre répertoire, mais aussi le rentrer au clavier.
Celui-ci doit être un raffinement de la machine.

`Oracle Machine` :
: Ce champ doit contenir le nom d'un fichier.
Pour entrer le nom du fichier vous pouvez parcourir votre répertoire, mais aussi le rentrer au clavier.
Celui-ci doit être un oracle pour les obligations de preuve de la machine.

`Oracle Raffinement` :

Ce champ doit contenir le nom d'un fichier.
Pour entrer le nom du fichier vous pouvez parcourir votre répertoire, mais aussi le rentrer au clavier.
Celui-ci doit être un oracle pour les obligations de preuve du raffinement.

`Force` :
: Vous devez choisir, dans la liste déroulante, un entier entre 0 et 3 .
Il désigne le niveau de force de preuve à employer dans l'*AtelierB* lors de la résolution des obligations de preuve.

`Affichage Minimum` :
: Vous devez cocher la case ou pas.
Cela permet de minimiser l'affichage en ne détaillant pas chaque opération effectuée.

`Nettoyer` :
: Vous devez cocher la case ou pas.
Cela permet de dire à l'outil de laisser les fichiers de travail en place à la fin de la génération.

`Que Générer OP` :
: Vous devez cocher la case ou pas .
Si la case est cochée, aucune preuve ne sera effectuée et toutes les machines contenant les obligations de preuve seront générées. De plus, le dossier de travail ne sera pas effacer.

`Vérifier Oracle` :
: Vous devez cocher la case ou pas.
Si la case est cochée et qu'un oracle est passé en paramètre alors l'oracle sera utilisé pour diriger les obligations de preuves générées afin de les minimiser.

`AtelierBenParallele` :
: Vous devez cocher la case ou pas.
Si la case est cochée , alors un processus de l'*AtelierB* sera conserver en mémoire. Dans le cas contraire, l'*AtelierB* sera relancé à chaque fois qu'une obligation de preuve devra être résolue. Ce deuxième cas semble souvent plus sûr et n'est pas plus coûteux, grâce à la mise en cache de l'atelier.

`Debug` :
: Vous devez cocher la case ou pas.
Ce cas n'est utile que lorsque vous possédez une version de GénéSyst qui est en cours de développement. Cela permet alors d'obtenir des traces d'exécution plus détaillées.

`Symbole Défaut` :
: Vous devez choisir un caractère dans la liste déroulante.
Ce caractère sera utilisé pour indiquer qu'une condition du graphique produit n'est pas réductible à true.

`Symbole Prouve` :
: Vous devez choisir un caractère dans la liste déroulante.
Ce caractère sera utilisé pour indiquer qu'une condition du graphique produit n'est pas réductible à true.

Les valeurs par défaut sont les mêmes que dans le cas d'un appel en ligne de commande.

Remarque : *Certains remarques sont à préciser :*

- *Lorsque vous choisissez d'indiquer les noms de fichiers à partir de 'Parcourir', on vérifie si le fichier choisi est le type de fichier attendu. Si ce n'est pas le cas, vous êtes invité à rechoisir un autre.*
- *Lorsque vous souhaitez générer que les obligations de preuve alors vous verrez les champs Oracle Machine,Oracle Raffinement,Force, Vérifier Oracle se désactivés.*

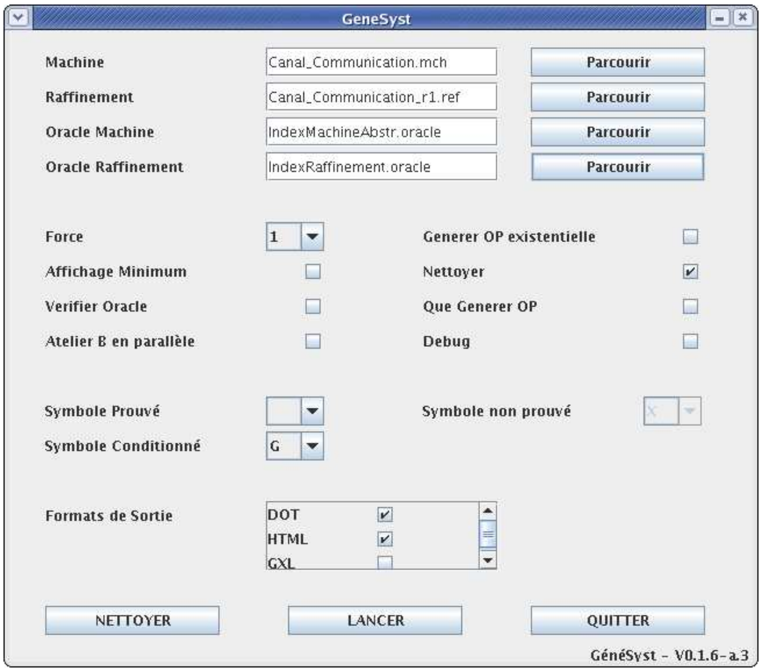

Figure 4.1: Exemple d'utilisation de l'interface graphique

- *Tant qu'aucun oracle n'est fournit, le champVérifier Oracle reste inactif.*
- *Lorsque vous choisissez en sortie un fichier au format HTML, alors automatiquement, le format Dot sera aussi généré.*

4.1.2 Exécution de GénéSyst

Après avoir choisi de lancer de GénéSyst, vous verrez apparaître une fenêtre permettant de suivre la trace de son exécution (cf Fig. 4.2). Enfin, après que celle-ci soit terminée, les deux boutons retour et visualiser deviennent actifs. Le premier permet de revenir au menu principal, tandis que le second permet d'avoir un aperçut de ce que l'outil vient de produire. Un exemple est donné en figure 4.3.

4.2 Mode *nettoyage*

Ce mode sert à effacer automatiquement les différentes "pollutions" qui ont pu être générées par l'outil. Pour accéder à ce mode, il suffit de cliquer sur le bouton *"Nettoyer"*. Il n'y a aucun champ nécessaire pour lancer ce mode.

Ensuite, vous verrez apparaître une fenêtre (Fig 4.4) constituée de deux onglets :
- *"Projets"*, qui permet le nettoyage des projets qui ont été créés dans l'*AtelierB* par GénéSyst par l'utilisateur courant. Ces projets sont nommés par défaut *CalculTransitionsTmpxxx*. Ce nom peut être suivi d'un suffixe numérique.
  Les projets vous seront listés, il vous suffit de choisir ceux que vous désirez supprimer en

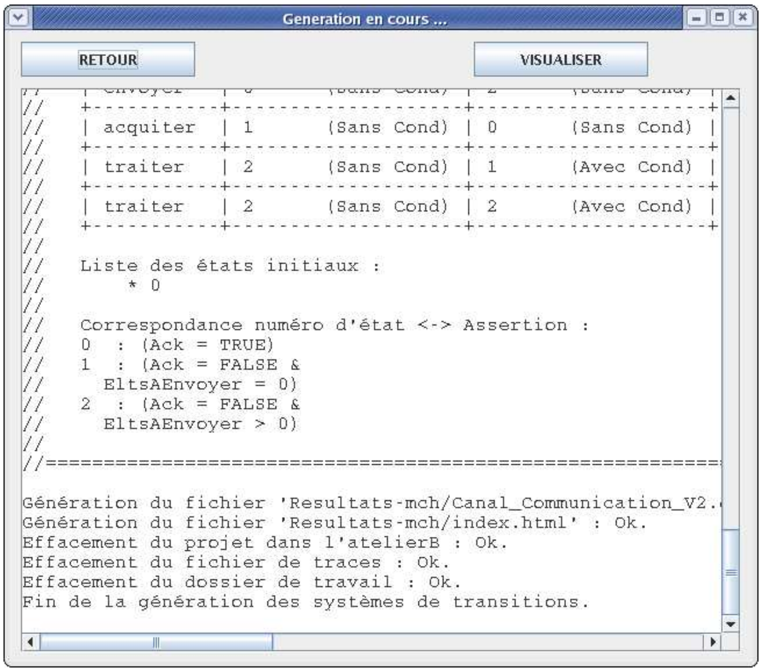

Figure 4.2: Exemple de trace d'utilisation de GénéSyst

cochant les cases correspondantes.
- *"Dossiers"*, qui permet le nettoyage des dossiers qui servent à stocker toutes les informations temporaires pendant le calcul (Répertoires pour l'utilisation de l'*AtelierB*, les fichiers à prouver, etc). Les dossiers en question se nomment *CalculDesTransitions*. Ce nom peut-être suivi d'un suffixe numérique. Les dossiers vous seront listés, il vous suffit de choisir ceux que vous désirez supprimer en cochant les cases correspondantes.

Remarque : *Lorsque aucuns projets ni dossiers n'ont été trouvés, un message vous serra fournit. La fenêtre de nettoyage va apparaître mais les boutons de suppressions sont désactivés.*

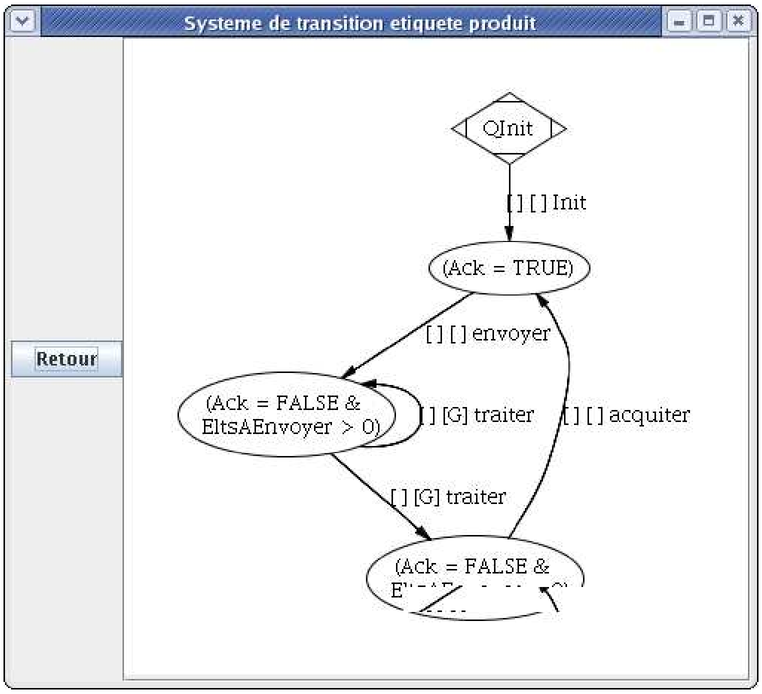

Figure 4.3: Exemple de visualisation produite par Grappa, sous GénéSyst

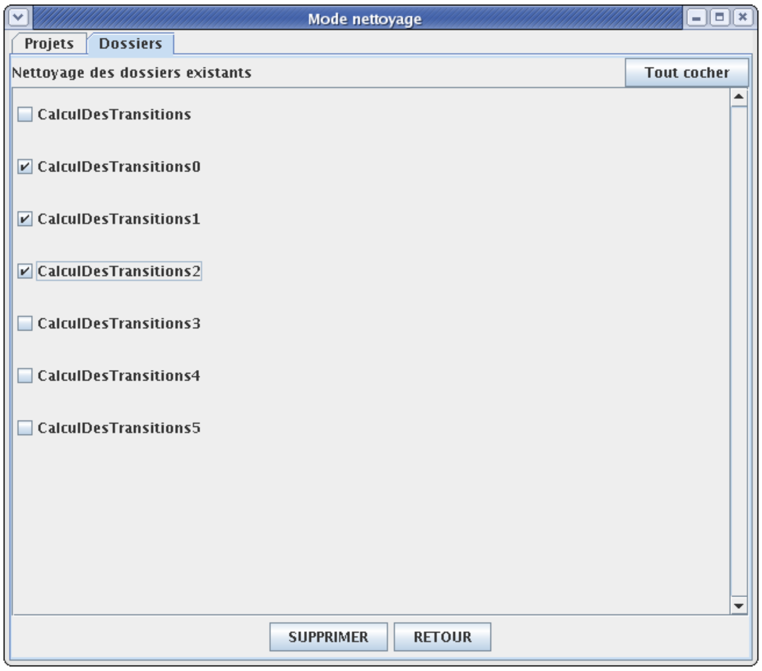

Figure 4.4: Exemple de fenêtre permettant le nettoyage par GénéSyst

Sorties et traces de GénéSyst 5

5.1 Mode *Génération de système de transitions étiquetées*

5.1.1 Fichiers générés

Pour générer un automate, GénéSyst doit effectuer un certain nombre d'obligations de preuve qui sont produites sous forme de machines B et qui sont prouvées, pour l'instant, via l'*AtelierB*. Ce dernier point nécessite notamment la présence d'un dossier BDP et d'un dossier LANG. C'est pourquoi l'outil commence par créer un répertoire de travail dans lequel il produira, par la suite, les fichiers et dossiers nécessaire à son bon fonctionnement. Le dossier de travail produit se nommera par défaut *"CalculDesTransitions"* (Ce choix est configurable lors de l'installation). Il se peut que ce nom soit suivi d'un suffixe numérique afin d'éviter tout conflit avec un dossier déjà existant.

Parmi les fichiers présents dans ce dossier, se trouve le fichier *"IndexMachinesAbstr.Oracle"* et possiblement le fichier *"IndexMachinesRaff.Oracle"*. Ces deux fichiers sont la version textuelle et éditable du format intermédiaire utilisé en interne pour représenté le système de transitions étiquetées généré. Ainsi, Ces fichiers peuvent être remis en entrée de l'outil (avec les options `-OM` et `-OR`) afin de simplifier les preuves. D'où le nom d'*oracle* qui leur est donné. Les oracles seront détaillé à la section 5.3.

De même, un nouveau projet est créé dans l'*AtelierB* lors de chacune des générations de systèmes de transitions étiquetées. Ces projets se nomment *"CalculTransitionsTmpxxx"* avec un éventuel suffixe numérique.

Enfin, un fichier traçant l'utilisation de l'*AtelierB* est créé dans le dossier courant. Celui-ci peut être assez volumineux et doit donc être effacé si l'outil ne l'a pas fait. Ce fichier se nomme par défaut *TraceGenesyst.log*. Contrairement aux noms de dossier et projet, lors de l'exécution de GénéSyst, si un fichier portant ce nom est déjà présent alors il est effacé et remplacé par celui-ci.

Par défaut, le projet, le dossier de calcul et le fichier de traces sont effacés à la fin de l'exécution du programme, si tout s'est bien passé et un fichier d'automate est généré dans le sous-dossier *Resultat-mch* (ou *Resultat-ref* pour le cas d'un raffinement) du dossier courant. Les exceptions notables sont les suivantes :

- si l'option `-N` est mise à false alors le dossier de travail et le fichier de trace ne sont pas effacés, mais le projet est effacé ;
- si l'option `-G` est mise à true alors le dossier de travail n'est pas effacé, mais le projet et le fichier de trace sont effacés ;

– si l'outil est arrêté au milieu de son travail (par `ctrl-c` par exemple) alors ni le dossier de travail ni le projet ni le fichier de trace ne sont effacés.

A la fin du calcul d'un système de transitions étiquetées, GénéSyst génère un ou deux dossiers (suivant que l'on ait, ou non, demander à étudier un raffinement). Ces dossiers sont nommés : *"Resultats-mch"* pour les informations concernant la machine et *"Resultats-ref"* pour celles du raffinement. Ces dossiers contiennent tous les fichiers de sortie permettant la visualisation du système de transitions étiquetées :

– Le fichier au format *.dot* qui est le fichier généré par défaut par GénéSyst. Ce format est textuel et peut être traduit en un grand nombre de formats vectoriels ou non par l'outil GraphViz[1] [GN99] de AT&T.
– De même qu'un fichier au format *.html* généré aussi par GénéSyst à la demande de l'utilisateur. Ce format nécessite la génération du format *.dot*. Le format *.html* permet d'avoir plus d'informations sur le graphe généré par le *.dot*, sans l'alourdir pour autant, grâce à l'utilisation d'hyperliens sur une image. La génération d'un tel résultat entrainera aussi la génération d'une image *.gif* représentant l'automate et de trois dossiers(*"Clusters"*, *"Etats"* et *"Transitions"*) qui contiennent les pages html de description des différentes transitions. Ces sont ces pages qui sont appelées lorsque l'on clique sur l'hyperimage de la représentation html.
– Le fichier au format *.gxl*, si celui-ci est demandé par l'utilisateur. Ce format, dérivé du XML, est textuel et peut être utilisé par un grand nombre d'outils.

Historiquement, le premier format supporté par GénéSyst a été le format **.bcg** qui nécessitait l'utilisation de la suite d'outils **CADP**[2] de H. Garavel [GMS01, GLM02, GM99]. Cependant, cette sortie a été temporairement desactivée. En effet, le format BCG impose certaines contraintes, dont le fait que les noms des états ne peuvent pas être autre chose que des nombres et qu'il n'est pas possible de réaliser d'automate hiérarchisé. De plus on ne peut pas mettre de couleurs ou de styles particuliers pour mettre en valeurs certaines informations.

Un exemple de résultat obtenu avec la sortie BCG est en section 6.1 et un exemple du même résultat obtenu avec la sortie DOT est en section 6.1.

5.1.2 Traces de GénéSyst

Lors de son exécution en mode génération de système de transitions étiquetées, GénéSyst commence par afficher un tableau récapitulatif des différents paramètres qu'il va utiliser pour son travail. En voici un exemple :

```
#######################################################################
# Rappel sur les données du traitement :
# ====================================
#
# Fichier du système abstrait                      : Distributeur/DISTRI3.mch
# Fichier du raffinement                           : Distributeur/DISTRI3_r1.ref
# Faire la vérification de l'oracle                : non
# Nom du projet                                    : CalculTransitionsTmpxxx7
# Nom du Dossier de calcul                         : CalculDesTransitions0
# Nom du fichier de traces de l'atelier            : TraceGenesyst.log
# Nom du fichier d'oracle machine à lire           :
# Nom du fichier d'oracle raffinement à lire       :
# Force des preuves                                : 1
# Ne faire que générer les OPs                     : non
# Effacer le dossier de travail à la fin           : oui
# Conserver un processus de l'AtelierB en parallèle : non
```

[1]http ://www.graphviz.org
[2]http ://www.inrialpes.fr/vasy/cadp/

```
# Symbole de condition réductible à true           :
# Symbole de condition non réductible à true       : G
# Formats de sortie demandés                       : Dot, Html
#
####################################################################
```

Si un oracle est utilisé alors les information qui y sont contenues sont affichées. Bien sûr seule les informations contenant un résultat de preuve sont prises en compte. Ainsi, une règle contenant une information de preuve à '?' sera ignorée. Pour plus d'information sur la syntaxe d'un oracle, référez vous à la section 5.3.3. Voici un exemple de trace de chargement d'oracle :

```
//====================================================================
// Chargement de l'oracle :
//   CalculDesTransitions/IndexMachinesAbstr.Oracle
// ------------------------
//
//   AssocNumEtat-Etat(0,"(Etat = 0)")
//   AssocNumEtat-Etat(1,"(Etat = 1)")
//   AssocNumEv-Ev(0,"Prendre_Monnaie")
//   AssocNumEv-Ev(1,"Servir_Cafe")
//   Init(0,F)
//   Init(1,T)
//   Decl(0,0,T)
//   Decl(0,1,F)
//   Decl(1,0,F)
//   Decl(1,1,F)
//   Att(0,0,0,T)
//   Att(0,1,0,G)
//
//====================================================================
```

Ensuite, l'affichage indique, pour chaque état, s'il est initial. Dans l'exemple suivant l'état E_0 est initial et l'état E_1 ne l'est pas.

```
//====================================================================
// Recherche des états initiaux ...
// --------------------------------
//
//  Statut : E0 est un état totalement initial.
//  Statut : E1 n'est pas un état initial.
//
//====================================================================
```

Enfin, pour chacun des états atteignable non encore étudié, l'outil calcule et affiche, pour chaque événement, s'il est déclenchable, et, si oui, où il mène. Dans l'exemple qui suit, la trace indique que les états numéro 0 et 1 ont été vérifiés comme atteignables et que l'on cherche les transitions qui en partent. En l'occurrence, seul un événement ($Prendre\_Monnaie$) semble déclenchable. Il donne lieu à deux transitions qui partent de l'état E_0 avec une condition réductible à true et mènent respectivement en l'état E_0 et l'état E_1 avec les conditions d'atteignabilité true et G.

```
//====================================================================
// Recherche des événements déclenchables depuis le(s) état(s) :
//  0, 1
// -------------------------------------------------------------
//
// Evénement 0 : Prendre_Monnaie
//   Statut : Evénement déclenchable dans l'état 0 (Sans cond.)
//     //============================================================
//     // Recherche des états d'arrivée ...
//     // ---------------------------------
//     //   Statut : Transition de E0 Vers E0 par Ev0 (Sans cond.)
```

```
//     //   Statut : Transition de E0 Vers E1 par Ev0 (Sous cond.)
//     //=========================================================
//   Statut : Evénement non déclenchable dans l'état 1.
//
// Evénement 1 : Servir_Cafe
//   Statut : Evénement non déclenchable dans l'état 0.
//   Statut : Evénement non déclenchable dans l'état 1.
//
//=================================================================
```

Une fois le calcul de toutes les transitions réalisé, l'ensemble des transitions est résumé dans un tableau textuel qui est affiché à la suite des traces d'exécution. L'exemple ci-dessous résume les transitions trouvées dans l'exemple précédant. De plus, les états initiaux sont rappelés et la concordance entre les numéros d'état et les prédicats est indiquée. Ainsi, ce seul résultat suffit à construire entièrement un système de transitions étiquetées. Ce sont d'ailleurs ces informations qui sont stockées dans le fichier d'oracle (étant le format intermédiaire de sortie) généré.

```
//=================================================================
// Liste des transitions existantes trouvées :
// -------------------------------------------
//
// +-----------------+--------------------+--------------------+
// | Opération       |   Etat de départ   |   Etat d'arrivée   |
// +-----------------+--------------------+--------------------+
// | Prendre_Monnaie | 0      (Sans Cond) | 0      (Sans Cond) |
// +-----------------+--------------------+--------------------+
// | Prendre_Monnaie | 0      (Sans Cond) | 1      (Avec Cond) |
// +-----------------+--------------------+--------------------+
//
// Liste des états initiaux :
//     * 0
//
// Correspondance numéro d'état <-> Assertion :
//   0  : (Etat = 0)
//   1  : (Etat = 1)
//
//=================================================================
```

Dans le cas d'un raffinement seul le calcul de recherche des équivalences d'états est ajouté au début des calculs. Dans l'exemple qui suit nous considérons que les états de la spécification ont été définis par l'assertion :

$$(Etat = 0) \ \vee \ (Etat = 1)$$

et les états du raffinement par

$$((Etat = 1) \Leftrightarrow ((Etat = 1 \ \wedge \ feu = rouge) \ \vee \ (Etat = 1 \ \wedge \ feu = vert))) \wedge$$
$$((Etat = 0) \Leftrightarrow (Etat = 0))$$

Un numéro désigne chaque état. Ce chiffre est choisi par ordre croissant à partir de 0 et attribué par ordre d'apparition des états. Ainsi la numérotation des états raffinés est 0 pour $(Etat = 1 \ \wedge \ feu = rouge)$, 1 pour $(Etat = 1 \ \wedge \ feu = vert)$ et 2 pour $(Etat = 0)$.

```
//=================================================================
// Recherche des associations entre états abstraits et du raffinement
// ----------------------------------------------------------------
//
// L'état 1 est raffiné par : 0 or 1
// L'état 0 est raffiné par : 2
//
//=================================================================
```

Enfin, à la fin de l'exécution de GénéSyst, les deux lignes suivantes sont affichées. Elles indique le nom du fichier généré contenant le système de transitions étiquetées et le fait que l'outil à terminer de travaillé. Il rend donc la main à l'utilisateur.

```
Génération du fichier 'MachineACafeClassique.dot' : terminée.
Fin de la génération des systèmes de transitions.
```

5.2 Mode Nettoyage

En mode nettoyage, le programme va d'abord regarder s'il y a des projets, puis des dossiers susceptibles d'avoir été créés par lui.

S'il n'y en a pas il dira simplement :

```
Aucun projet à effacer n'a été trouvé.

Aucun dossier à effacer n'a été trouvé.
```

S'il y a des projets qui porte comme nom :

CalculTransitionsTmpxxx$n$

(où n est un entier positif) alors ils seront tous listés. Exemple :

```
Projets trouvés :

CalculTransitionsTmpxxx
CalculTransitionsTmpxxx0
CalculTransitionsTmpxxx12556
```

Ensuite, un menu apparaît :

```
Menu :
------
   T : Tous les effacer
   L : Lister chaque projet et demander s'il faut l'effacer
   P : n'effacer aucun projet et nettoyer les Dossiers
   Q : Quitter le programme sans toucher aux projets
```

Si l'on choisit *Quitter*, alors on sort du programme sans avoir touché à rien.

Si l'on choisit de *Passer*, alors les projets resteront intouchés et l'on passera à la suite.

Si l'on choisit la première option, alors tous les projet listés ci-dessus seront effacé sans plus poser de question.

Si l'on choisit de lister chaque projet alors le contenu de chacun d'eux sera affiché et à tour de rôle on vous demandera :

```
Projet : CalculTransitionsTmpxxx


Voulez vous effacer ce projet ? (O/N)
```

Deux réponses sont alors possibles : O si l'on veut effacer le projet et N si l'on ne veut pas effacer le projet. Mais, dans la plupart des cas, le projet sera vide.

Une fois cette première partie terminée, un deuxième menu apparaît, précédé du nom des dossiers ayant put-être générés par le programme :

```
    Dossiers trouvés :

CalculDesTransitions
CalculDesTransitions0
CalculDesTransitions13215

Menu :
------
    T : Tous les effacer
    L : Lister chaque dossier et demander s'il faut l'effacer
    Q : Quitter le programme sans rien toucher
```

Cette fois ci encore, *Quitter* permet de quitter le programme sans toucher aux dossiers trouvés. Cependant, les projets qui auront été effacés précédemment ne seront pas pour autant récupérés.

Les commandes *Tous les effacer* et *Lister chaque dossier* sont identiques à celle du premier menu à ceci près que l'on parle là de projets et non de dossiers.

5.3 Format intermédiaire des système de transitions étiquetées : l'Oracle

Depuis sa version 1.3 (janvier 2004), GénéSyst permet de générer et d'exporter le format intermédiaire lui servant à générer les systèmes de transitions étiquetées. Les fichiers obtenus par cette exportation portent le doux nom d'*Oracle* car ils peuvent ensuite être donnés en entrée de l'outil afin qu'il ne refasse pas les preuves déjà effectuées ou qu'il oriente les obligations de preuves à générer pour est réduire le nombre.

La première motivation pour la création d'un fichier de format intermédiaire est la génération du système de transitions étiquetées d'un raffinement. En effet, il arrive fréquemment que des erreurs ou des imprécisions soient détectées lors de la visualisation d'un **STE**. Or la génération du système de transitions étiquetées d'un raffinement nécessite de généré d'abord celui de son abstraction. Et comme une génération est souvent longue, il est alors très intéressant de pouvoir sauter cette première étape.

De plus, une utilisation fine de l'Oracle permet de se concentrer sur une partie seulement de la génération en forçant temporairement certains résultats de preuve. Ainsi, on pourra écrire un Oracle décrivant tout un système sauf les transitions qui partent d'un état E, le temps d'affiner la spécification afin que cette partie de l'automate soit bien celle que l'on attendait.

La deuxième grande raison est de pouvoir affiner par preuve interactive les transitions. C'est à dire que l'ensemble des obligations de preuve sont générées et écrites dans des fichiers mis dans le même dossier que l'Oracle. Ainsi, si l'on pense qu'une transition devrait avec une condition réductible à false ou true et que cela n'a pas été prouvé automatiquement, alors on peut récupérer l'obligation de preuve et le prouver soit même. Ensuite, Le résultat doit être intégré dans l'Oracle pour être pris en compte à la prochaine génération du système de transitions étiquetées du système.

Dans les trois sous sections suivantes nous allons donc décrire comment obtenir un oracle, puis comment l'utiliser, avant de conclure sur comment l'éditer.

5.3.1 Génération d'un Oracle

Nous appelons un Oracle *plein* un Oracle contenant les résultats de la génération d'un système de transitions étiquetées. Il est nommé ainsi par opposition avec un Oracle *vide* qui contient toute la structure syntaxique, mais aucun résultat de preuve. Ce dernier type d'Oracle est utilisé Uniquement pour l'édition manuelle (cf. 5.3.3).

Obtention d'un Oracle "plein"

Par défaut, le format intermédiaire Oracle est toujours produit. Cependant, il est généré dans le dossier de travail. Ainsi, pour le récupérer il faut interdire à GénéSyst d'effacer ce dossier. L'option `-n false` est donc indispensable. Le fichier d'Oracle porte par défaut le nom *IndexMachinesAbstr.Oracle* ou *IndexMachinesRaff.Oracle* suivant que le système étudié sois une abstraction ou un raffinement.

Obtention d'un Oracle "vide"

L'obtention d'un Oracle vide se fait grâce à l'option `-G true`. Il n'est pas nécessaire de préciser `-n false` avec celle-ci, car le dossier ne sera pas effacer à la fin de la génération. Avec l'option `-G true`, GénéSyst ne va que générer toutes les obligations de preuves possible et un Oracle vide. Celui-ci n'aura alors d'intérêt que s'il est ensuite édité manuellement (cf 5.3.3).

5.3.2 Utilisation d'un Oracle

Mettons nous dans le cas où nous avons un système B et un Oracle partiel ou total de son système de transitions étiquetées. Cet Oracle peut avoir été généré automatiquement ou manuellement, cela est indifférent. Si notre système est une machine abstraite nommée `MaMachine.mch`, alors l'appel de l'outil pour générer son système de transitions étiquetées décrit par l'Oracle *IndexMachinesAbstr.Oracle* est le suivant :

```
java geneSyst MaMachine.mch -OM IndexMachinesAbstr.Oracle
```

L'option `-OM` indique que le paramètre suivant est le nom de l'Oracle à utiliser pour la génération du système de transitions étiquetées de la machine abstraite. On pourra ainsi demander de générer le système de transitions étiquetées d'une machine et de son raffinement avec soi un Oracle pour ne simplifier que la génération du système de transitions étiquetées de la machine :

```
java geneSyst MaMachine.mch -OM IndexMachinesAbstr.Oracle -R MonRaffinement.ref
```

Soi un Oracle pour ne simplifier que la génération du système de transitions étiquetées du raffinement :

```
java geneSyst MaMachine.mch -OR IndexMachinesRaff.Oracle -R MonRaffinement.ref
```

Soi un Oracle pour simplifier la génération du système de transitions étiquetées de la machine et un Oracle pour diriger la génération du système de transitions étiquetées du raffinement :

```
java geneSyst MaMachine.mch -OM IndexMachinesAbstr.Oracle -OR IndexMachinesRaff.Oracle -R MonRaffinement.ref
```

Enfin, dans tous les cas, si l'on veut juste que l'Oracle serve à diriger la génération des obligations de preuve, on rajoute l'option `-V TRUE`. Tous les oracles passés en paramètre seront donc vérifiés par preuve. Exemple :

```
java geneSyst MaMachine.mch -OM IndexMachinesAbstr.Oracle -OR IndexMachinesRaff.Oracle -R MonRaffinement.ref -V TRUE
```

5.3.3 Edition d'un Oracle

Un Oracle est un fichier texte ayant une syntaxe particulière. Concrètement, chaque ligne contient 0 ou 1 instruction. Un commentaire commence par // et se termine sur la fin de la ligne. Il existe 6 instructions différentes :

1. **AssocNumEtat-Etat(int** E, **String** S)
2. **AssocNumEv-Ev(int** Ev, **String** S)
3. **Init(int** E, **BoolExt** Pr)
4. **Decl(int** E, **int** Ev, **BoolExt** Pr)
5. **Att(int** $E1$, **int** $E2$, **int** Ev, **BoolExt** Pr)
6. **Equ(int** ER, **int** EA, **BoolExt** Pr)

Remarque : *Attention à l'ordre d'apparition des prédicats. Un Oracle doit toujours commencer par les prédicats* **AssocNumEtat-Etat** *puis les prédicats* **AssocNumEv-Ev**.

Les fonctions ne sont séparées que par un ou plusieurs retours à la ligne. Il n'y a donc pas de ';'. Les types utilisés pour décrire les paramètres sont **int**, **String** et **BoolExt**. Le premier désigne l'ensemble des entiers positifs ou nuls, le deuxième est l'ensemble des chaînes de caractère. Une chaîne de caractères commence et fini par une double quotte (Ex : *"Ma chaine"*). Enfin, le dernier ensemble est l'ensemble des booléens étendus avec la valeur *Gardé* et la valeur *Inconnu*. Syntaxiquement, on notera T la valeur true, F la valeur false, G la valeur *Gardé* et *?* la valeur *Inconnu*.

La sémantique de chacune de ces opérations est la suivante :

- **AssocNumEtat-Etat(int** E, **String** S) : Ce prédicat permet d'associer un numéro E à un prédicat logique décrivant un état. Le prédicat logique en question est la chaîne de caractère S le représentant. Sa syntaxe est celle du . Ce prédicat n'est actuellement utilisé dans GénéSyst que pour savoir combien il y a d'état dans le système décrit. Dans un proche avenir il pourra aussi servir à permettre à l'utilisateur de changer le numéro associer à un état, mais cela n'est actuellement pas fait. Il faut donc conserver l'ordre d'apparition des états et leur numérotation.
- **AssocNumEv-Ev(int** Ev, **String** S) : Ce prédicat permet d'associer un numéro Ev à un événement. L'événement en question représenté ici par la chiné de caractères S de son nom. Ce prédicat n'est actuellement utilisé dans GénéSyst que pour savoir combien il y a d'événements dans le système décrit. Dans un proche avenir il pourra aussi servir à permettre à l'utilisateur de changer le numéro associer à un événement, mais cela n'est actuellement pas fait. Il faut donc conserver l'ordre d'apparition des événements et leur numérotation.
- **Init(int** E, **BoolExt** Pr) : Ce prédicat permet d'associer à chaque état E un niveau d'initiabilité Pr. Un état peut être totalement initial (*T*), partiellement initial (*G*), jamais initial (*F*) ou l'on peut ne pas connaître son niveau d'initiabilité (*?*).
- **Decl(int** E, **int** Ev, **BoolExt** Pr) : Ce prédicat permet d'associer à chaque couple formé d'un état E et d'un événement Ev une condition de déclenchabilité Pr. Ev peut être totalement déclenchable depuis E (*T*), partiellement déclenchable depuis E (*G*), jamais déclenchable depuis E (*F*), ou l'on peut ne pas connaître la condition de déclenchabilité de Ev depuis E (*?*).
- **Att(int** $E1$, **int** $E2$, **int** Ev, **BoolExt** Pr) : Ce prédicat permet d'associer à chaque triplet formé d'un état de départ $E1$, d'un état d'arrivée $E2$ et d'un événement Ev une condition d'atteignabilité Pr. $E2$ peut être toujours atteignable depuis $E1$ par Ev (*T*), partiellement atteignable depuis $E1$ par Ev (*T*), jamais atteignable depuis $E1$ par Ev (*T*), ou l'on peut ne pas connaître la condition de atteignabilité de $E2$ par Ev depuis $E1$ (*?*).

– **Equ(int** ER, **int** EA, **BoolExt** Pr) : Ce prédicat sert dans le cas de l'Oracle d'un raffinement. En effet, dans ces cas là GénéSyst va chercher à retrouver la correspondance entre les états de l'abstraction et les états du raffinement. Cela permet de ne pas contraindre l'utilisateur à une comparaison syntaxique des états, mais une comparaison sémantique. Ce prédicat associe donc à un état raffiné ER et un état abstrait EA une preuve d'équivalence Pr. Celle-ci a les contraintes suivantes :

1. Chaque état doit être équivalent à 1 et 1 seul autre.
2. Pr ne peut pas prendre la valeur *?*.

Remarque : *Nous avons vu plus avant que chacune des obligations de preuve générées pour faire avancer l'outil dans la génération du système de transitions étiquetées final est conservée dans le dossier de l'Oracle. Le nom de chacun des fichier ayant été produit et utilisé est noté en commentaire dans l'Oracle, à côté du résultat qu'elle a permis d'obtenir. Par exemple, la ligne* **Init***(3,F //(Init-E3-T.mch,Init-E3-F.mch) indique que les fichiers Init-E3-T.mch et Init-E3-F.mch on servit à la preuve du résultat de cette ligne (F).*

Voici, ci-dessous, un exemple d'Oracle. Celui est l'Oracle décrivant l'intégralité du système de transitions étiquetées généré à partir de l'exemple en section 6.1 et qui décrit les figures 6.1 et 6.2.

```
//================================================
// Fichier généré automatiquement par GénéSyst
//------------------
//
// CORRESPONDANCE NUMETAT<->ETAT
// syntaxe :
// AssocNumEtat-Etat(numéro état, prédicat décrivant l'état)
  AssocNumEtat-Etat(0,"etm = arret")
  AssocNumEtat-Etat(1,"etm = pret")
  AssocNumEtat-Etat(2,"etm = select")
  AssocNumEtat-Etat(3,"etm = distri")
  AssocNumEtat-Etat(4,"etm = tropp")
  AssocNumEtat-Etat(5,"etm = remb")
//
//CORRESPONDANCE_NUMEV<->EV
// syntaxe :
// AssocNumEv-Ev(numéro événement, nom événement)
  AssocNumEv-Ev(0,"mettre_en_marche")
  AssocNumEv-Ev(1,"arreter_machine")
  AssocNumEv-Ev(2,"selectionner_boisson")
  AssocNumEv-Ev(3,"mettre_piece")
  AssocNumEv-Ev(4,"distribuer")
  AssocNumEv-Ev(5,"rendre_monnaie_1")
  AssocNumEv-Ev(6,"prendre_gobelet")
  AssocNumEv-Ev(7,"annuler")
  AssocNumEv-Ev(8,"rendre_monnaie_2")
//
//ETATS INITIAUX
// syntaxe :
// Init(numéro état, Niveau Preuve)
  Init(0,F //(Init-E0-T.mch,Init-E0-F.mch)
  Init(1,T //(Init-E1-T.mch,Init-E1-F.mch)
```

```
  Init(2,F //(Init-E2-T.mch,Init-E2-F.mch)
  Init(3,F //(Init-E3-T.mch,Init-E3-F.mch)
  Init(4,F //(Init-E4-T.mch,Init-E4-F.mch)
  Init(5,F //(Init-E5-T.mch,Init-E5-F.mch)
//
//DECLENCHABILITES
// syntaxe :
// Decl(numéro état, numero événement, Niveau Preuve)
  Decl(0,0,T //(Decl-E0-Ev0-T.mch,Decl-E0-Ev0-F.mch)
  Decl(0,1,F //(Decl-E0-Ev1-T.mch,Decl-E0-Ev1-F.mch)
  Decl(0,2,F //(Decl-E0-Ev2-T.mch,Decl-E0-Ev2-F.mch)
  Decl(0,3,F //(Decl-E0-Ev3-T.mch,Decl-E0-Ev3-F.mch)
...
...
  Decl(5,6,F //(Decl-E5-Ev6-T.mch,Decl-E5-Ev6-F.mch)
  Decl(5,7,F //(Decl-E5-Ev7-T.mch,Decl-E5-Ev7-F.mch)
  Decl(5,8,T //(Decl-E5-Ev8-T.mch,Decl-E5-Ev8-F.mch)
//
//ATTEIGNABILITES
// syntaxe :
// Att(numéro état de part, numéro état d'arrivee, numero événement, Niveau Preuve)
  Att(0,0,0,F //(Att-E0-Ev0-E0-T.mch,Att-E0-Ev0-E0-F.mch)
  Att(0,1,0,T //(Att-E0-Ev0-E1-T.mch,Att-E0-Ev0-E1-F.mch)
  Att(0,2,0,F //(Att-E0-Ev0-E2-T.mch,Att-E0-Ev0-E2-F.mch)
  Att(0,3,0,F //(Att-E0-Ev0-E3-T.mch,Att-E0-Ev0-E3-F.mch)
  Att(0,4,0,F //(Att-E0-Ev0-E4-T.mch,Att-E0-Ev0-E4-F.mch)
  Att(0,5,0,F //(Att-E0-Ev0-E5-T.mch,Att-E0-Ev0-E5-F.mch)
  Att(1,0,1,T //(Att-E1-Ev1-E0-T.mch,Att-E1-Ev1-E0-F.mch)
  Att(1,0,2,F //(Att-E1-Ev2-E0-T.mch,Att-E1-Ev2-E0-F.mch)
  Att(1,1,1,F //(Att-E1-Ev1-E1-T.mch,Att-E1-Ev1-E1-F.mch)
  Att(1,1,2,F //(Att-E1-Ev2-E1-T.mch,Att-E1-Ev2-E1-F.mch)
  Att(1,2,1,F //(Att-E1-Ev1-E2-T.mch,Att-E1-Ev1-E2-F.mch)
  Att(1,2,2,G //(Att-E1-Ev2-E2-T.mch,Att-E1-Ev2-E2-F.mch)
  Att(1,3,1,F //(Att-E1-Ev1-E3-T.mch,Att-E1-Ev1-E3-F.mch)
...
...
  Att(5,4,5,F //(Att-E5-Ev5-E4-T.mch,Att-E5-Ev5-E4-F.mch)
  Att(5,4,6,F //(Att-E5-Ev6-E4-T.mch,Att-E5-Ev6-E4-F.mch)
  Att(5,4,7,F //(Att-E5-Ev7-E4-T.mch,Att-E5-Ev7-E4-F.mch)
  Att(5,4,8,F //(Att-E5-Ev8-E4-T.mch,Att-E5-Ev8-E4-F.mch)
  Att(5,5,2,F //(Att-E5-Ev2-E5-T.mch,Att-E5-Ev2-E5-F.mch)
  Att(5,5,3,F //(Att-E5-Ev3-E5-T.mch,Att-E5-Ev3-E5-F.mch)
  Att(5,5,4,F //(Att-E5-Ev4-E5-T.mch,Att-E5-Ev4-E5-F.mch)
  Att(5,5,5,F //(Att-E5-Ev5-E5-T.mch,Att-E5-Ev5-E5-F.mch)
  Att(5,5,6,F //(Att-E5-Ev6-E5-T.mch,Att-E5-Ev6-E5-F.mch)
  Att(5,5,7,F //(Att-E5-Ev7-E5-T.mch,Att-E5-Ev7-E5-F.mch)
  Att(5,5,8,F //(Att-E5-Ev8-E5-T.mch,Att-E5-Ev8-E5-F.mch)
//——————
// FIN DU FICHIER
//===============================================
```

Exemples 6

6.1 Distributeur à café (Format BCG et DOT)

Sur l'exemple figure 6.1, nous remarquons que l'opération *mettre_piece* peut aller, depuis l'état 1, dans n'importe quel état. Cela vient du fait que rien n'a put être prouvé. Ce même exemple a été produit au format DOT en figure 6.2.

Voici le code de la machine B initiale :

MACHINE DISTRI
SETS
$BOISSONS = \{rien, cafe, the, choc\};$
$PIECES = \{demiF, unF, deuxF, cinqF\};$
$ETM = \{arret, pret, select, distri, tropp, remb\}$
CONSTANTS $prix, valeur$
PROPERTIES
$prix \in BOISSONS \longrightarrow NAT \wedge$
$valeur \in PIECES \longrightarrow NAT \wedge$
$prix(rien) = \ \wedge prix(cafe) = 300 \ \wedge \ prix(the) = 200 \ \wedge \ prix(choc) = 250 \wedge$
$valeur(demiF) = 50 \ \wedge \ valeur(unF) = 100 \ \wedge \ valeur(deuxF) = 200 \wedge$
$valeur(cinqF) = 500$
VARIABLES $boisson, somme, affichage, etm$
INVARIANT
$boisson \in BOISSONS \wedge$
$somme \in 0..max(ran(prix)) + max(ran(valeur)) \wedge$
$affichage = prix(boisson) - somme \wedge$
$etm \in ETM$
ASSERTIONS $etm = arret \ \vee \ etm = pret \ \vee \ etm = select \ \vee \ etm = distri \ \vee \ etm = tropp \ \vee \ etm = remb$
INITIALISATION $boisson, somme, affichage, etm := rien, 0, 0, arret$
OPERATIONS
mettre_en_marche =SELECT $etm = arret$ THEN $etm := pret$ END ;
arreter_machine = SELECT $etm = pret$ THEN $etm := arret$ END ;
selectionner_boisson = SELECT $etm = pret$ THEN
ANY bb WHERE $bb \in BOISSONS \ \wedge \ bb \neq rien$ THEN
$boisson := bb || somme := 0 || affichage := prix(bb) || etm := select$
END
END ;
mettre_piece = SELECT $etm = select \ \wedge \ somme < prix(boisson)$ THEN

```
    ANY  cc WHERE cc ∈ PIECES THEN
      somme := somme + valeur(cc)||affichage := affichage − valeur(cc)
    END||etm := select
  END ;
distribuer = SELECT etm = select ∧ somme ≥ prix(boisson) THEN
    IF somme = prix(boisson) THENetm := distri ELSE etm := tropp END
  END ;
yy ⟵ rendre_monnaie_1 = SELECT  etm = tropp THEN
    yy := somme − prix(boisson)||somme := prix(boisson)||affichage := 0||etm := distri
  END ;
prendre_gobelet = SELECT  etm = distri THEN
    boisson := rien||somme := 0||affichage := 0||etm := pret
  END ;
annuler = SELECT  etm = select THEN
    IF  somme = 0 THENaffichage := 0||boisson := rien||etm := pret
    ELSE  etm := remb
    END
  END ;
zz ⟵ rendre_monnaie_2 = SELECT  etm = remb THEN
    zz := somme||somme := 0||affichage := 0||boisson := rien||etm := pret
  END
END
```

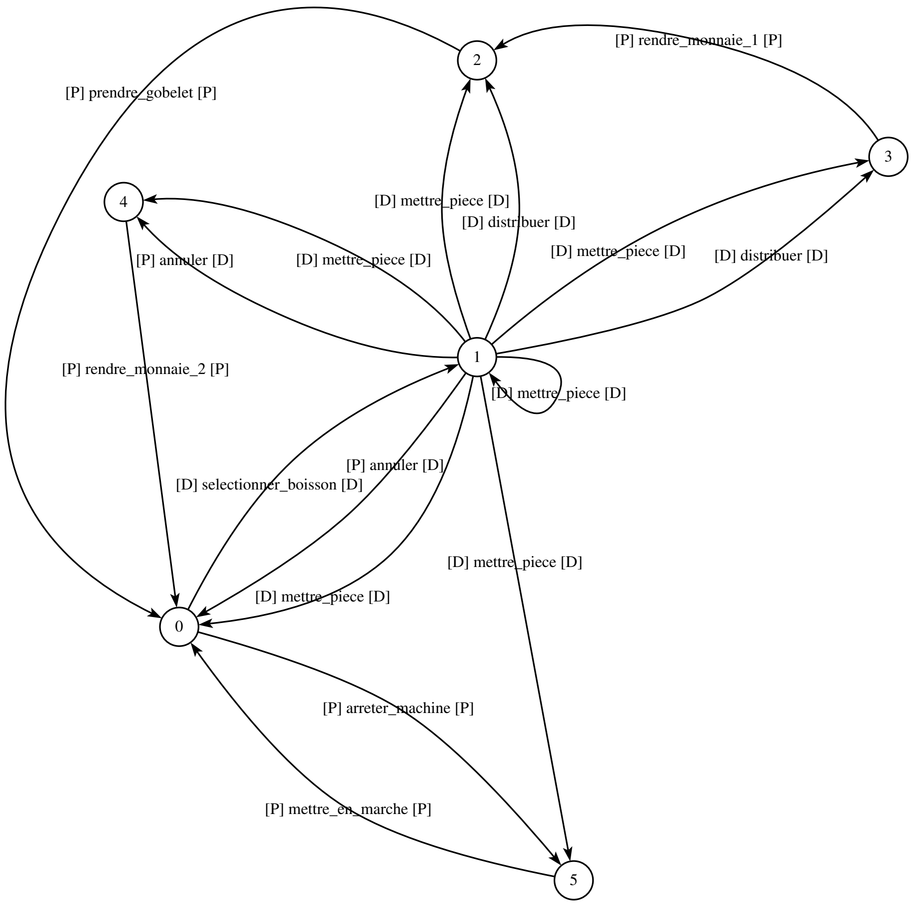

Figure 6.1: Exemple de génération BCG

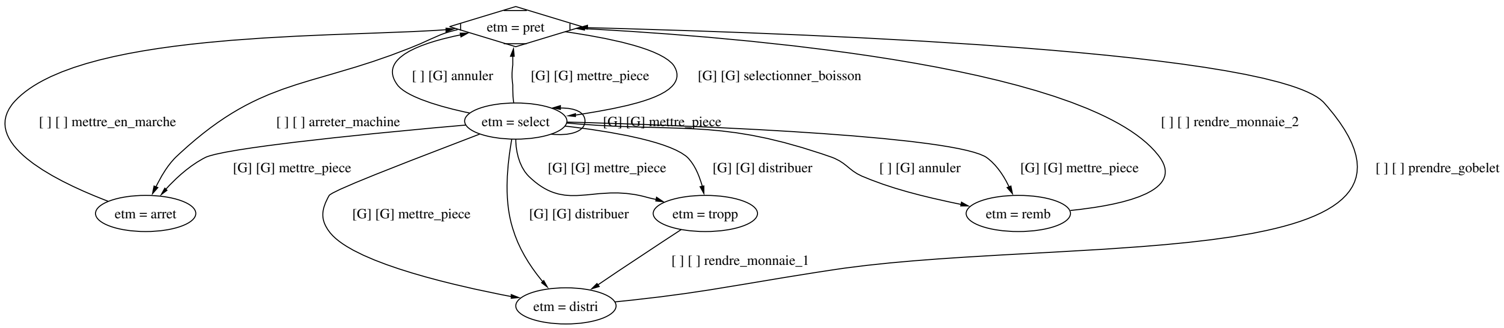

Figure 6.2: Exemple de génération DOT

7 Preuve d'une formule avec l'*AtelierB*

Dans cette section, nous allons simplement expliquer comment nous réalisons des preuves de formules grâce à l'*AtelierB*, alors que ce dernier n'est pas, à proprement parler, un prouveur de théorème tel que Coq ou PVS.

Le principe est simple : il suffit de générer une machine dont les obligations de preuve correspondent aux formules attendues et de prouver ces dernières à l'aide de l'*AtelierB*.

Pour cela, si, l'on met la formule désirée dans la clause d'assertion, alors l'obligation de preuve générée sera :

$$A \ \wedge \ P \ \wedge \ I \Rightarrow J$$

où A représente les conditions sur les ensembles déclarés, P les propriétés et I l'invariant.

En fait, il est nécessaire de mettre dans cette machine toutes les clauses du systèmeB original, exception faite des opérations.

Remarquons simplement, que toutes les obligations de preuves dont nous parlions jusqu'à maintenant avaient en hypothèses implicites la présence de ces clauses.

Voici un exemple tiré de la spécification de la machine à café décrite en section 6.1. Cet exemple est l'obligation de preuve générée pour prouver la déclenchabilité de l'événement $mettre\_en\_marche$ depuis l'état $etm = arret$.

MACHINE DISTRI

SETS

$BOISSONS = \{rien, cafe, the, choc\}$;

$PIECES = \{demiF, unF, deuxF, cinqF\}$;

$ETM = \{arret, pret, select, distri, tropp, remb\}$

CONSTANTS $prix, valeur$

PROPERTIES

$prix \in BOISSONS \longrightarrow NAT \ \wedge$

$valeur \in PIECES \longrightarrow NAT \ \wedge$

$prix(rien) = \ \wedge \ prix(cafe) = 300 \ \wedge \ prix(the) = 200 \ \wedge \ prix(choc) = 250 \ \wedge$

$valeur(demiF) = 50 \ \wedge \ valeur(unF) = 100 \ \wedge \ valeur(deuxF) = 200 \ \wedge$

$valeur(cinqF) = 500$

VARIABLES $boisson, somme, affichage, etm$

INVARIANT

$boisson \in BOISSONS \ \wedge$

$somme \in 0..max(ran(prix)) + max(ran(valeur)) \ \wedge$

$affichage = prix(boisson) - somme \wedge$

$etm \in ETM$

ASSERTIONS

*/*L'événement 0 n'est il jamais déclenchable depuis l'état 0 ?*/*

*/*Etat1 : 0 Etat2 : -1 transition n° 0 –*/*

$((etm = arret) => not(etm = arret))$

INITIALISATION $boisson, somme, affichage, etm := rien, 0, 0, arret$

OPERATIONS

mettre_en_marche = *skip* ;

arreter_machine = *skip* ;

selectionner_boisson = *skip* ;

mettre_piece = *skip* ;

distribuer = *skip* ;

yy ⟵ rendre_monnaie_1 = *skip* ;

prendre_gobelet = *skip* ;

annuler = *skip* ;

zz ⟵ rendre_monnaie_2 = *skip*

END

8 Bilan

Actuellement, la dernière version de GénéSyst est disponible en libre[1] téléchargement à l'adresse suivante :

http ://www-lsr.imag.fr/Les.Personnes/Nicolas.Stouls/

Elle est accompagnée d'un rapide fichier d'aide à l'installation. L'installation ne sera donc pas décrite dans ce document car la méthodologie peut avoir changée entre temps.

De plus, sur cette page se trouve un lien vers la documentation JavaDoc des classes de GénéSyst.

Toutes les demandes d'informations complémentaires ou les problèmes d'installation ou autres concernant l'outil GénéSyst peuvent être adressées soit à Didier Bert (didier.bert@imag.fr) soit à Nicolas Stouls (nicolas.stouls@imag.fr).

[1] L'application GénéSyst est développée sous les termes du contrat de la licence Cecill. Celle-ci permet de diffuser librement les sources d'un programme sans pour autant en perdre la propriété. Une copie de ce contrat de licence est disponible à la même adresse que GénéSyst.

Bibliography

[Abr96a] J.-R. Abrial. Extending B without Changing it (for Developping Distributed Systems). In H. Habrias, editor, *Proceedings of the 1st Conference on the B method*, 1996. ISBN 2-906082-25-2.

[Abr96b] J.R. Abrial. Extending B Without Changing it. In Nantes H. Habrias, editor, *First B conference*, 1996.

[Abr96c] J.R. Abrial. *The B-Book.* Cambridge University Press, 1996.

[AM98a] J.-R. Abrial and L. Mussat. Introducing Dynamic Constraints in B. In D. Bert, editor, *B'98 : The 2nd International B Conference, Recent Advances in the Development and Use of the B Method*, volume 1345 of *LNCS*. Springer Verlag, 1998.

[AM98b] J.R. Abrial and L. Mussat. Introducing Dynamic Constraints in B. In D. Bert, editor, *Proceedings of the Second International B Conference*, volume 1393 of *LNCS*. Springer-Verlag, 1998.

[BC00] D. Bert and F. Cave. Construction of Finite Labelled Transition Systems from B Abstract Systems. In *Integrated Formal Methods*, volume 1945 of *LNCS*. Springer-Verlag, 2000.

[BP03] D. Bert and M.-L. Potet. *Spécification en B, Support de cours de l'ENSIMAG et du DEA ISC*, 2003. Institut National Polytechnique de Grenoble.

[BPS05] D. Bert, M-L. Potet, and N. Stouls. Genesyst : a tool to reason about behavioral aspects of b event specifications. application to security properties. In *ZB 2005 : Formal Specification and Development in Z and B*, volume 3455 of *LNCS*, pages 299–318. Springer-Verlag, 2005.

[Bri02] Alexandre Brillant. *Introduction à UML*, 2002.

[But99] Michael Butler. Fm'99 : World congress on formal methods. In *An Overview of Event-Based B*, Septembre 1999.

[BW96] M. Butler and M. Waldén. Distributed System Development in B. In H. Habrias, editor, *Proceedings of the 1st Conference on the B method*, 1996. ISBN 2-906082-25-2.

[Cav00] Francis Cave. Passage du B système aux sytsèmes de transitions étiquetés finis. Rapport de DEA, Institut National Polytechnique de Grenoble, France, 2000.

[Coo00] K. M. L. Cooper. Statecharts. Document de cours, Université de Dallas, Texas, 2000.

[GLM02] H. Garavel, F. Lang, and R. Mateescu. An overview of CADP 2001. *EASST*, Newsletter 4 :13–24, 2002.

[GM99] Hubert Garavel and Radu Mateescu. Modélisation d'un système d'entrées-sorties scsi-2 v3.0. Technical report, action VERDON, 1999.

[GMS01] H. Garavel, R. Mateescu, and I. Smarandache. Parallel state space construction for model-checking. In *Proceedings of the 8th international SPIN workshop on Model checking of software*, Series-Proceeding-Article, pp.217 - 234. Springer-Verlag New York, 2001.

[GN99] E. R. Gannsner and S. C. North. An open graph visualization system and its applications to software engineering. Technical report, Laboratoires AT&T, Etats unis, 26 mai 1999.

[Ham02] Smaine Hamdane. Génération de systèmes de transition étiquetés à partir de la description d'un système d'évènements décrits avec le langage B. Rapport de maîtrise, Université Joseph Fourrier, Grenoble-1, France, mai 2002.

[Har87] David Harel. Statecharts : A visual formalism for complex systems. *Science of Computer Programming*, 8(3) :231–274, 1987.

[HM01] P. H. Hartel and L. Moreau. *Formalising the Safty of* **Java***, the Virtual Machine and* **JavaCard**, 25 mars 2001.

[Leb00] J. Lebray. Modélisation de systèmes en B : Proposition de guides méthodologiques pour la décomposition d'événements. Rapport de DEA, Institut National Polytechnique de Grenoble, France, 2000.

[MG02] Pierre-Alain Muller and Nathalie Gaertner. *Modélisation objet avec UML*. ISBN : 2-212-09122-2, Code éditeur : G09122. Editions Eyrolles, 2002.

[MPS04] Xavier Morselli, Marie-Laure Potet, and Nicolas Stouls. Génésyst : Génération d'un système de transitions étiquetées à partir d'une spécification B événementiel. In *AFADL'2004*, pages 317–320, June 2004.

[Nah01] J. Nahoum. Outils d'assistance à la construction de systèmes dans la méthode B. Rapport de DEA, Institut National Polytechnique de Grenoble, France, 2001.

[OMG01] OMG. *Unified Modeling Language Specification*, septembre 2001. Version 1.4.

[Ori00a] C. Oriat. *Invocation de méthodes en* **Java**, 2000. Polycopié ENSIMAG.

[Ori00b] C. Oriat. *Quelques éléments de* **Java**, 2000. Polycopié ENSIMAG.

[Pot00a] Marie-Laure Potet. *B Systeme. Etude de cas Parking*, 2000. Institut National Polytechnique de Grenoble, Transparents de cours.

[Pot00b] Marie-Laure Potet. *Introduction au B évènementiel. Evènement - Raffinement - Décomposition*, 2000. Institut National Polytechnique de Grenoble, Transparents de cours.

[PS04] M-L. Potet and N. Stouls. Explicitation du contrôle de développement _événementiel. In *AFADL'2004*, pages 13–27, June 2004.

[SUN00] SUN. *Java Virtual Specification*, Mai 2000.

[VTH02] J.C. Voisinet, B. Tatibouet, and A. Hammad. jBTools : An experimental platform for the formal B method. In *PPPJ'02*, pages 137–140. Trinity College, Dublin, Ireland, Juin 2002.