Université d'Angers
UFR Sciences

Maitrise Informatique

Travail d'étude et de recherche «Démonstrateur de théorèmes par la méthode des tableaux sémantiques»

Mathieu Alamichel
<mathieu@info.univ-angers.fr>

Gregory Boissinot
<gregory@info.univ-angers.fr>

Olivier Moreau
<olivier@info.univ-angers.fr>

28 mars 2003

Table des Matières

1 Présentation du sujet **1**

2 Notions sous-jacentes à la Démonstration Automatique **2**
2.1 Rappels . . . 2
2.2 Tableaux Sémantiques . . . 3

3 Structures de données **5**
3.1 Données du problème . . . 5
3.1.1 Représentation du fichier . . . 5
3.1.2 Lecture du fichier . . . 6
3.2 Modélisation du problème . . . 6
3.2.1 Diagramme UML . . . 6
3.2.2 Classe Symbole . . . 8
3.2.3 Classe TableSymboles . . . 8
3.2.4 Classe Formule . . . 8
3.2.5 Classe ListeFormules . . . 8
3.2.6 Classe Noeud . . . 9
3.2.7 Classe ArbrePreuve . . . 9

4 Procédure de preuve **10**
4.1 Première stratégie . . . 10
4.2 Deuxième stratégie . . . 12
4.2.1 Affectation des poids . . . 12
4.2.2 Algorithme . . . 13

5 Interface graphique **16**
5.1 Fonctionnement de l'interface . . . 16
5.2 Dessin de l'arbre de preuve . . . 17

6 Conclusion et Perspectives **18**
6.1 Conclusion . . . 18
6.2 Perspectives . . . 18

7 Bibliographie **19**

Annexe : Fichiers sources **20**
Fichier Parseur.flex . . . 20
Fichier Parseur.y . . . 21
Fichier TabSemantique.java . . . 22
Fichier Symbole.java . . . 23
Fichier TableSymboles.java . . . 24
Fichier Formule.java . . . 26
Fichier ListeFormules.java . . . 31
Fichier Noeud.java . . . 34
Fichier ArbrePreuve.java . . . 35
Fichier BarreMenus.java . . . 39
Fichier BarreOutils.java . . . 40
Fichier BarreEtat.java . . . 41
Fichier ZoneDessin.java . . . 42
Fichier ZoneFormules.java . . . 45
Fichier Fenetre.java . . . 46
Fichier Actions.java . . . 47

Fichier Aide.java 49
Fichier makefile 50

Table des figures

1 Diagramme UML 7
2 Exemple1 avec la première stratégie 11
3 Exemple2 avec la première stratégie 12
4 Comparaison des deux procédures de preuve sur le fichier de la figure3 14
5 chargement d'un fichier 16
6 Preuve d'un ensemble de formules. 17

Liste des tables

1 Règles d'expansion α et β 3

1 Présentation du sujet

Le but de ce travail d'étude et de recherche est de réaliser un démonstrateur de théorèmes en utilisant la méthode des tableaux sémantiques.

La **Démonstration Automatique** s'intéresse à la preuve de théorèmes dans le cadre de la logique. Un démonstrateur automatique de théorèmes est un programme qui utilise des procédés de déduction, appelés *règles d'inférences*, pour inférer des conclusions qui découlent logiquement d'un ensemble d'axiomes. La recherche de preuve en logique est un problème combinatoire complexe car elle s'attaque à des problèmes NP-Complets.

La méthode des tableaux sémantiques, introduites par Robert **Smullyan**, issue des travaux de E.W.**Beth** en 1955, est une méthode analytique s'articulant autour d'une décomposition de formules en formules plus simples faisant apparaître la structure logique de la formule initiale, mais en laissant de côté les détails qui ne sont pas essentiels.

L'implémentation a été réalisée en langage Java. Ce langage objet possède de nombreux avantages dont la facilité d'utilisation et d'évolution, plus particulièrement dans le domaine graphique. Même si la portabilité du programme n'était pas essentielle pour ce projet, elle constitue un aspect non négligeable apporté par le langage JAVA. De plus, Java est capable de s'interfacer avec des outils d'analyse lexicale et syntaxique comme *JFlex* et *BYacc* similaires aux *Lex* et *Yacc* du C.

Nous commencerons par un rappel des différentes notions sous-jacentes à la Démonstration Automatique. Puis nous expliquerons nos choix de modélisation et détaillerons les aspects les plus importants de notre structure de données grâce à un schéma UML et une explication non exhaustive des différents champs et méthodes de nos classes. Nous développerons ensuite la méthode de construction de la preuve d'un problème et expliquerons les algorithmes et stratégies utilisés. Cette présentation montrera enfin, les différentes fonctionnalités de notre logiciel TabSémantique. Tout au long de ce travail seront présentés de nombreux exemples illustrant les résultats obtenus.

2 Notions sous-jacentes à la Démonstration Automatique

2.1 Rappels

Définition 1 - Théorie - *Soit $H = \{H_1, \ldots, H_n\}$ un ensemble de formules. La théorie définie par H, notée Th(H), est l'ensemble des formules F qui découlent logiquement de H ($H \vdash F$). L'ensemble de formules H représente les axiomes de la théorie.*

Les problèmes que l'on cherche à résoudre en Démonstration Automatique s'énoncent alors en ces termes :

Etant donné un ensemble d'axiomes H et une formule F, F est-elle un théorème de Th(H) ? Autrement dit, a-t-on $H_1 \wedge \ldots \wedge H_n \vdash F$?

Il résulte de la question précédente que montrer que F est un théorème de Th(H) est équivalent à montrer que F est conséquence logique de H. On a alors recours à une procédure de preuve pour montrer la possible dépendance logique existante entre H et F.

Définition 2 - Procédure de preuve - *Une procédure de preuve P est un algorithme capable d'établir si une fomrule F est un théorème d'une théorie H. Si c'est le cas, on dit que F est dérivable de H par P et on note $H \vdash_P F$. Une procédure de preuve accepte en entrée une formule ou un ensemble de clauses et fournit en sortie une réponse binaire oui/non indiquant si la formule est ou non un théorème.*

Définition 3 - Adéquation d'une procédure de preuve - *Une procédure de preuve est dite adéquate si elle ne peut prouver que des théorèmes.*

Autrement dit, si la procédure de preuve termine avec la réponse *oui*, la formule est obligatoirement un théorème.

Définition 4 - Complétude d'une procédure de preuve - *Une procédure de preuve P est dite complète si tout théorème (ou formule valide) est prouvable dans P.*

Autrement dit, la procédure de preuve termine toujours avec la réponse *oui* si la formule est un théorème.

Certaines procédures de preuve, comme le principe de résolution, font également référence à la notion de consistance.

Définition 5 - Consistance d'une procédure de preuve - *Une procédure de preuve P est consistante s'il n'existe pas de formule F telle que $\vdash_P F$ et $\vdash_P \neg F$.*

Nous pouvons à présent définir les notions connexes aux procédures de preuve que sont les notions d'espace de recherche, de raffinement et d'heuristique.

Définition 6 - Espace de recherche - *On appelle espace de recherche G, le résultat de l'application des règles d'inférence R, d'une procédure de preuve P, sur un ensemble de formules H. On a alors,*

$$\begin{gathered} \mathcal{G}_\prime = H \\ \mathcal{G}_\rangle = R(G_{i-1})\ i > 0 \\ \mathcal{G} = \cup_{i \geq 0}\ \mathcal{G}_\rangle \end{gathered}$$

Définition 7 - Restriction - *On appelle restriction d'une procédure de preuve, une procédure qui génère un sous-ensemble de l'espace de recherche G.*

Par exemple, le recours à des techniques de calcul pondéral sur les littéraux constitue une restriction. Le weighting constitue à attribuer une valeur (ou poids) aux symboles en fonction de l'importance qu'on leur accorde dans la recherche de la preuve. Cette technique permet d'éliminer les littéraux dont le poids dépasse une valeur limite fixée par l'utilisateur et sensés être inutiles pour la recherche de preuve.

Définition 8 - Stratégie - *On appelle stratégie pour une procédure de preuve, une procédure qui réordonne le développement de l'espace de recherche.*

La recherche en largeur d'abord, en profondeur d'abord ou par approfondissement successifs sont des exemples de stratégies.

Définition 9 - Raffinement - *Un raffinement d'une procédure de preuve est une restriction et/ou une stratégie.*

Un raffinement est une règle ou un ensemble de règles qui gouvernent l'utilisation des règles d'inférence.

Définition 10 - Heuristique - *On appelle heuristique un ensemble de règles(généralement empiriques) qui permet de parcourir un sous espace de l'espace de recherche.*

2.2 Tableaux Sémantiques

Avec la méthode des tableaux sémantiques, les preuves prennent la forme d'arbres ou chaque noeud est étiqueté par une formule. On définit alors des règles de réécriture sur les arbres, appelées *règles d'expansion* qui consistent à développer les branches d'un arbre pour faire apparaître la structure de la formule à démontrer. Ces règles sont classées en quatre catégories selon l'agencement des formules

- *les règles* α pour les formules conjonctives,
- *les règles* β pour les formules disjonctives,
- *les règles* γ pour les fomrules quantifiées universellemnt,
- *les règles* δ pour les fomrules quantifieés existentiellement,

Nous limiterons notre étude au calcul propositionnel, ainsi nous considérons uniquement les deux premières règles.

$\frac{\neg\neg F}{F}$	$\frac{\neg\top}{\bot}$	$\frac{\neg\bot}{\top}$	$\frac{\alpha_1 \wedge \alpha_2}{\alpha_1 \quad \alpha_2}$	$\frac{\beta_1 \vee \beta_2}{\beta_1 \mid \beta_2}$	$\frac{\beta_1 \rightarrow \beta_2}{\neg\beta_1 \mid \beta_2}$	$\frac{\beta_1 \equiv \beta_2}{\beta_1 \quad \neg\beta_1 \;\; \beta_2 \quad \neg\beta_2}$

Table 1: Règles d'expansion α et β

Définition 11 *Tout tableau T^*est obtenu par application des règles d'expansion sur un tableau T précédent.*

Pour démontrer $H_1 \wedge \ldots \wedge H_n \vdash F$, on commence par construire un tableau à une branche, puis on applique les règles d'expansion sur chaque branche du tableau jusqu'à obtenir une branche close ou ne plus pouvoir appliquer aucune règle.

Définition 12 - Branche close - *Une branche d'un tableau est dite close si F et $\neg F$ apparaissent dans cette branche ou si $\bot$ termine cette branche.*

Définition 13 - Tableau Clos - *Un tableau est clos si toutes ses branches sont closes.*

Définition 14 - Théorème d'insatisfiabilté - *Si F est un théorème de $Th(H)$ alors il existe un tableau clos pour F résultant de l'application des règles d'expansion sur un tableau initial de $H \wedge \neg F$.*

L'application des règles d'expansion se fait de manière récurrente sur chaque branche ouverte de l'arbre. L'ordre d'application des règles est en principe indifférent, mais il semble plus intéressant de donner la préférence aux règles qui ont un développement vertical (règles α) plutôt qu'horizontal (règles β) afin d'économiser des distributions sur plusieurs branches.

3 Structures de données

3.1 Données du problème

3.1.1 Représentation du fichier

Les données du problème correspondent à une liste de formules. L'utilisateur doit stocker l'ensemble de ces formules sur lequel va s'appliquer la procédure de preuve dans un fichier.

La grammaire utilisée pour décrire une liste de formules, une formule et un identificateur est la suivante :

$$
\begin{array}{l}
F \rightarrow \neg\ (F) \\
F \rightarrow F \wedge F \\
F \rightarrow F \vee F \\
F \rightarrow F \rightarrow F \\
F \rightarrow F \leftrightarrow F \\
F \rightarrow (F) \\
F \rightarrow ID \\
\\
ID \rightarrow SYMBOLE \\
ID \rightarrow \neg\ SYMBOLE
\end{array}
$$

où un symbole est défini par une chaines de caractères commençant par une lettre et suivie de plusieurs lettre ou chiffres. (La caractère *underscore* _ est considéré comme une lettre)

Priorité des opérateurs logiques : (par ordre de priorité)
- opérateur de négation
- opérateurs d'implication et d'équivalence,
- opérateur de conjonction,
- opérateur de disjonction.

Par défaut, si une formule ne comporte pas de parenthèses, on utilise d'abord les priorités des opérateurs puis l'associativité. Donc on n'est pas obligé de mettre les formules entre parenthèses.

Voici deux exemples de fichier initiaux.

Exemple 1 -Fichier initial -

```
(p->q) ;
(q->r) ;
(r->s) ;
!(p->s) ;
```

Exemple 2 -Fichier initial -

```
!p->(q|r) ;
s->(q| !r) ;
!q->(!p&s) ;   !q ;
```

3.1.2 Lecture du fichier

Afin de récupérer chacune des formules dans notre structure de données, nous utilisons les outils lexicaux et syntaxiques *JFlex et BYacc/J*. Ces outils offrent une grande similitude avec *Lex et Yacc* (destinés au langage C et C++) dont l'efficacité et la souplesse d'utilisation ne sont plus à démontrer. Effectivement *JFlex et BYacc/J* acceptent comme *Lex et Yacc* les grammaires LALR (Look Ahead Left Right). Ceci est un atout primordial comparé à l'analyseur lexical et syntaxique *JavaCC* destiné à Java qui accepte uniquement les grammaires LL(1) (Left Left one) ; initialement, nous avions ce parseur ignorant l'existence de *JFlex et BYacc/J*.

Une grammaire LL(1) est une grammaire non ambigüe, sans récursion à gauche et factorisée à gauche. L'implémentation d'une telle grammaire se révèle alors très délicate, rapidement illisible et difficilement modifiable ce qui a pour effet de retirer tout intérêt à l'utilisation d'un parseur.

En revanche une grammaire LALR(1) est une grammaire très simple à mettre en oeuvre, lisible et facilement modifiable. L'utilisaton de *JFlex et BYacc/J* permet alors d'écrire et de lire le fichier de formules de manière naturelle.

3.2 Modélisation du problème

Nous avons utilisé une structure de données simple et suffisamment polyvalente pour recouvrir l'ensemble du problème. Notre structure est décrite par le diagramme UML suivant.

3.2.1 Diagramme UML

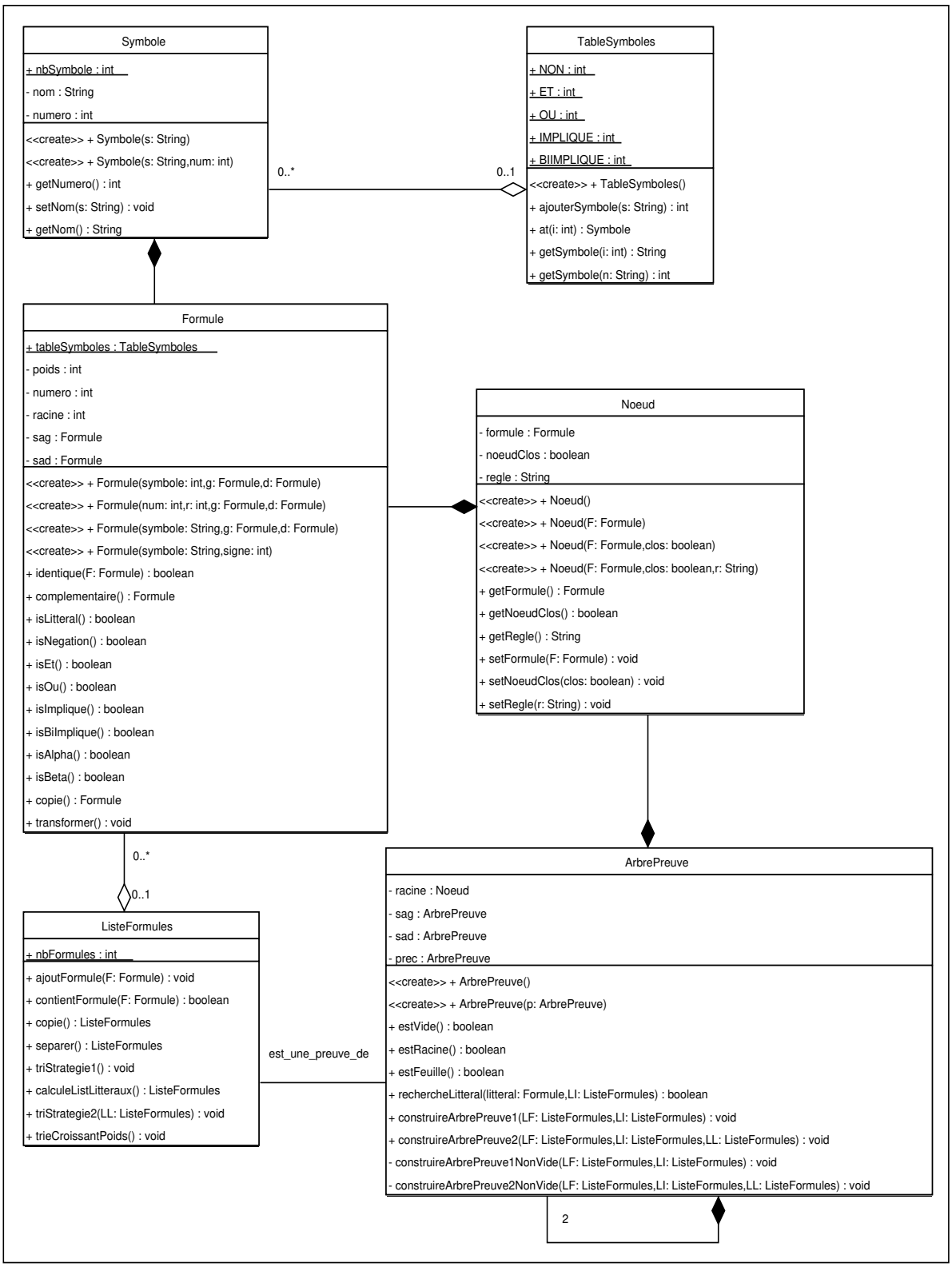

Figure 1: Diagramme UML

Voici une description non exhaustive de cette modélisation.

3.2.2 Classe Symbole

Premièrement, nous utilisons une classe nommée *Symbole* qui permet de représenter un symbole d'une formule. Un symbole est composé de son nom (p, q, ...) et d'un numéro unique permettant d'identifier chaque formule sans ambiguïté. Une variable statique *nbSymboles* permet l'affectation d'un numéro unique.

3.2.3 Classe TableSymboles

L'ensemble des symboles sont rassemblés dans une classe *TableSymboles*. Par défaut, une instance de la classe TableSymboles contient les symboles prédéfinis non, ou, et, implique, équivalence. Notre problème contient donc une seule table de symboles.

3.2.4 Classe Formule

L'entité de base de notre problème est la classe *Formule*. Elle peut contenir des sous-formules ou peut également représenter une formule constituée d'un unique littéral.

Cette classe à pour champs principaux :

- *racine* qui est l'indice du symbole de la formule dans la table de symboles ; ce symbole peut être un littéral ou alors une valeur par défaut de la table des symboles (non, et, ou, implique, équivalence). Dans ce dernier cas, à l'exception du non qui n'a qu'une partie gauche, une formule est alors constituée de deux sous-formules : la partie gauche de la formule et la partie droite.
- *sag* qui est une formule représentant si nécessaire la partie gauche de la formule
- *sad* qui est de la même manière la partie droite de la formule

Pour pouvoir identifier chaque formule de manière unique, on utilise un champ *numero* indicé de manière unique. De plus, dans certaines stratégies, il a été nécessaire d'ajouter un champ *poids* pour déterminer l'ordre d'application des formules. Une formule a une valeur attribuée pour ces deux champs uniquement si elle appartient à une liste de formules. En effet, une formule peut être formée de sous-formules ; ces dernières n'ont pas de poids et ne sont pas indicées. Cette solution d'indexation permet de garder un certaine cohérence lors de l'affichage de la preuve et des règles appliquées sur ces formules.

Cette classe contient également certaines méthodes importantes :

- *identique* qui vérifie que deux formules sont identiques,
- *complementaire* qui calcule le complémentaire d'une formule composée d'un unique littéral,
- *isX* qui teste si une formule est du type X (alpha, beta, non, ...),
- *transformer* qui transforme la formule selon les règles d'expansion (cf Table1),
- *copie* qui renvoie une copie d'une formule.

3.2.5 Classe ListeFormules

Les données de départ de la méthode des tableaux sémantiques forment un ensemble de formules.

Pour cela, il est nécessaire d'utiliser une classe *ListeFormules*. Cette classe correspond donc à une collection de formule implémentée en JAVA par un vecteur (vector) d'objets de la classe *Formule*.

Cette classe contient les méthodes suivantes :

- *contientFormule* qui teste si une formule appartient à la liste de formules
- *calculeListeLitteraux* qui recherche tous les littéraux inclus dans la liste de formules
- *triStrategie* qui tri la liste de formule selon une stratégie définie
- *copie* qui renvoie une copie d'une liste de formules

3.2.6 Classe Noeud

La classe *Noeud* est nécessaire pour la construction de l'arbre de preuve (cf plus bas). Celle-ci permet de stocker pour un nœud de l'arbre de preuve la formule du noeud grâce au champ *formule*, le champ *noeudClos* permettant de déterminer si un nœud est clos, et le champ *regle* indique la règle appliquée au nœud pour pouvoir afficher la trace de la preuve.

3.2.7 Classe ArbrePreuve

Après avoir décrit les entités de base de notre modélisation, nous allons maintenant décrire l'entité qui permettera de construire la preuve selon les différentes stratégies choisies. Cette preuve se construit à partir d'un ensemble de formules, c'est à dire à partir d'une instance de la classe *ListeFormules*.

La structure de la classe *ArbrePreuve* est basée sur un arbre binaire. Elle est composée des différents champs suivants :
- *racine* qui décrit le nœud (formule, closité du nœud, règle appliquée)
- *prec* qui représente le père du nœud et permet de remonter dans l'arbre de preuve
- *sag* qui représente le sous-arbre gauche
- *sad* qui représente le sous-arbre droit

Les sous-arbres permettent de représenter la suite de la preuve selon les alternatives suivantes :
- **si la formule est de type alpha :** la suite de la preuve est effectuée sur le sous-arbre gauche.
- **si la formule est de type beta :** la suite de la preuve est effectuée sur les deux sous-arbres.

La construction de la preuve est décrite dans le paragraphe suivant.

4 Procédure de preuve

A partir d'une liste de formules intiales, on commence par construire un tableau(ou un arbre) à une branche, puis on applique les règles d'expansion sur chaque branche du tableau jusqu'à obtenir une branche close ou ne plus pouvoir appliquer aucune règle.

Nous avons implémenter deux stratégies de procédures de preuves.

4.1 Première stratégie

C'est le rôle de la méthode *construireArbrePreuve1* qui s'applique sur une instance de la classe *ArbrePreuve*, qui prend en paramètre une liste de formules LF et qui retourne une instance de la classe *ArbrePreuve*. Le principe général est de construire l'arbre de preuve en favorisant simplement les formules du type α par rapport aux formules du type β.

Cette méthode suit le schéma récursif suivant :

1. Si la liste de formules LF est vide, on s'arrête.
2. On ordonne la liste de formules LF de telle sorte que les formules composées d'un unique littéral apparaissent en tête de la liste, suivis des formules du type α, puis des formules de type β.
3. On extrait la première formule F de la liste LF qu'on transforme afin d'obtenir un littéral, une conjonction, ou une disjonction.
4. On retire cette formule F de la liste LF
5. On distingue trois cas :

   (a) La formule est un littéral.
   On relance la construction de l'arbre sur la liste de formules LF.

   (b) La formule est de type α :

      i. On extrait les deux sous formules α_1 et α_2 ;
      ii. On crée le sous arbre gauche AG de l'arbre courant, qui contiendra la formule α_1 en racine ;
      iii. On affecte *l'arbre vide* au sous arbre droit de l'arbre courant.
      iv. On crée le sous arbre gauche AGG de l'arbre AG, cet arbre contiendra la formule α_2 en racine. Le sous arbre droit de l'arbre AG sera vide ;
      v. On teste si la branche créée est close. Si l'une ou l'autre des formules, α_1 ou α_2, sont des littéraux, on vérifie pour chacun si leur complémentaire est présent dans l'ensemble des pères du noeud de l'arbre AG. Si c'est le cas, on met à jour les informations de closité du noeud de l'arbre AGG et on s'arrête ;
      vi. Pour chaque sous formule α_1 et α_2, si ce n'est pas des littéraux, on les ajoute à la liste de formules LF ;
      vii. On réapplique la méthode *construireArbrePreuve1* sur l'arbre AGG avec en paramètre la liste de formules LF.

   (c) La formule est de type β :

      i. On extrait les deux sous formules α_1 et α_2 ;
      ii. On crée le sous arbre gauche AG de l'arbre courant, qui contiendera en racine la formule α_1 ;

iii. On crée le sous arbre droit AD de l'arbre courant, contenant en racine la formule α_2 ;

iv. On réalise une copie $LFCopie$ de la liste de formules LF ;

v. Si α_1 est un littéral et que son complémentaire est présent en remontant dans l'arbre, alors on met à jour l'information de closité pour le noeud contenant α_1 et on va en *viii.*

vi. Si α_1 n'est un littéral alors on l'ajoute à la liste de formules LF ;

vii. On relance la construction sur l'arbre de preuve AG avec la liste de formules LF ;

viii. Si α_2 est un littéral et que son complémentaire est présent en remontant dans l'arbre de preuve, on met à jour l'information de closité du noeud contenant α_2 et on s'arrête ;

ix. Si α_2 n'est pas un littéral alors on l'ajoute à la liste de formules $LFCopie$;

x. On réapplique la méthode *construireArbrePreuve1* sur l'arbre AD avec en paramètre la liste de formules $LFCopie$.

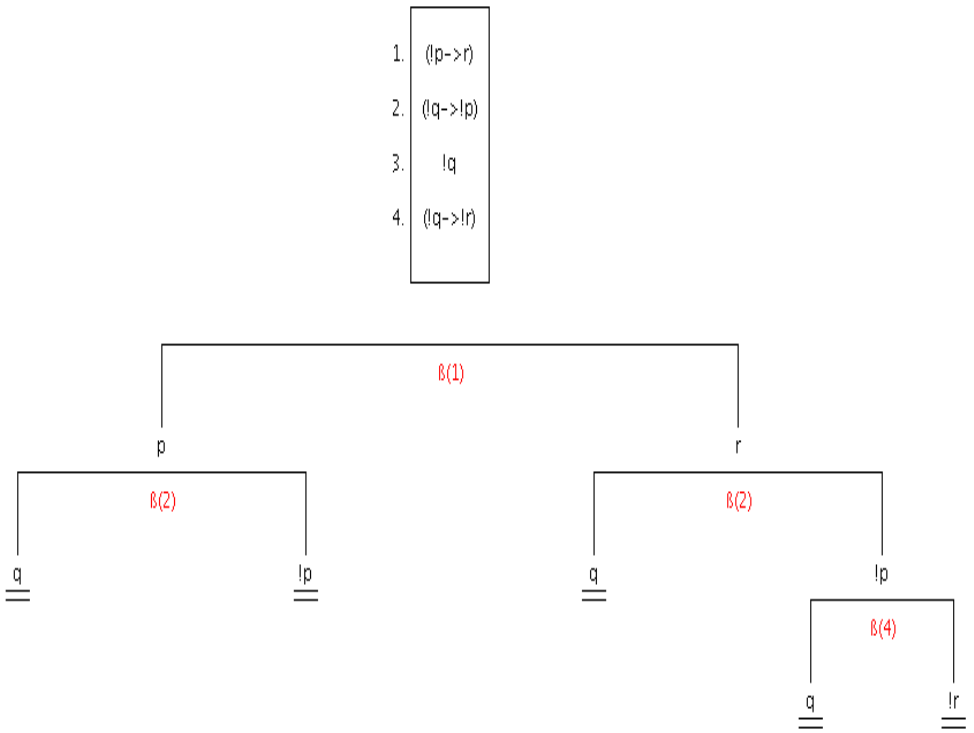

Figure 2: Exemple1 avec la première stratégie

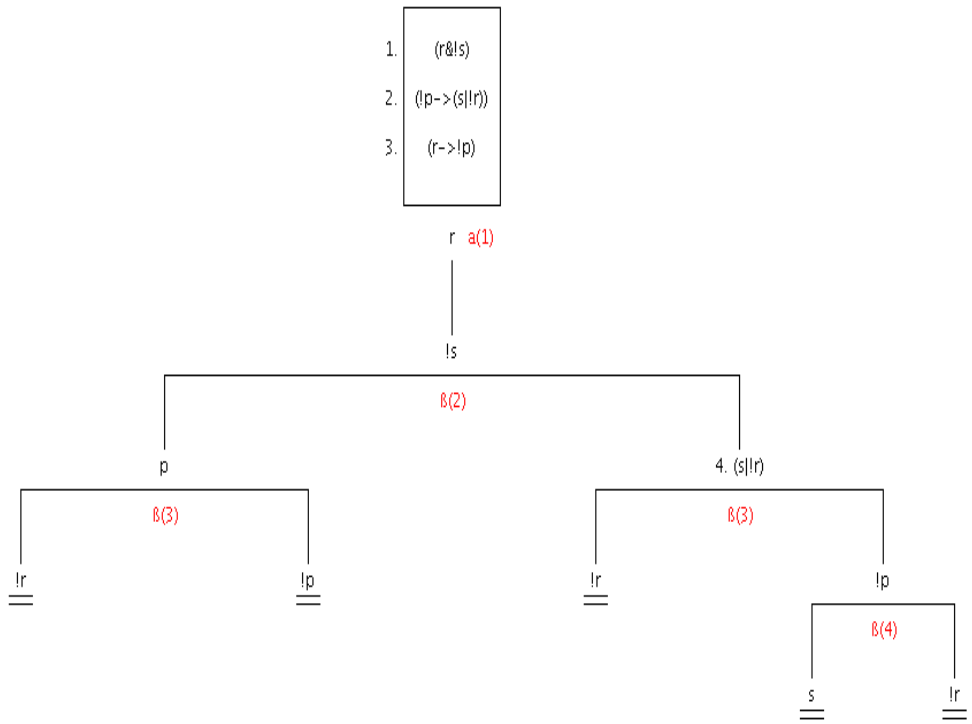

Figure 3: Exemple2 avec la première stratégie

4.2 Deuxième stratégie

C'est le rôle de la méthode *construireArbrePreuve2* qui s'applique sur une instance de la classe *ArbrePreuve*. Cette méthode prend en paramètre une liste de formules LF et une liste de littéraux LL et retourne une instance de la classe *ArbrePreuve*. Cette deuxième stratégie s'appuie sur un calcul pondéral des formules en fonction de l'importance de chaque formule dans la procédure de preuve.

4.2.1 Affectation des poids

On affecte un poids à chaque formule de la liste de formules LF. Les formules ayant un poids moins élevé sont les plus prioritaires. On dispose d'une borne commme poids maximal. Par défaut, le poids d'une formule de type α (*BorneAlpha*) est la valeur de la borne maximale divisée par quatre. De même, par défaut, le poids d'une formule de type β (*BorneBeta*) est la valeur de la borne maximale divisée par deux.

Pour chaque formule F :

- F est un littéral.
  On lui affecte la valeur zéro.

- F est une formule du type α :
  - On extrait les deux sous formules α_1 et α_2 ;
  - Si α_1 et α_2 sont deux littéraux et que leurs complémentaires sont présents dans la liste de littéraux LL, alors on affecte comme poids à la formule F, la valeur de la borne α divisée par quatre ;
  - Sinon, si α_1 est un littéral et que son complémentaire est dans la liste des littéraux LL, alors on affecte à la formule F comme poids la valeur de la borne α divisée par deux ;

  - Sinon, si α_2 est un littéral et que son complémentaire est dans la liste des littéraux LL, alors on lui affecte comme poids la valeur de la borne α divisée par deux ;
  - Sinon, on affecte comme poids, à la formule F la valeur de la borne α.

- F est une formule du type β :
  - On extrait les deux sous formules β_1 et β_2 ;
  - Si β_1 et β_2 sont deux littéraux et que leurs complémentaires sont présents dans la liste de littéraux LL, alors on affecte comme poids à la formule F, la valeur de la borne β divisée par quatre ;
  - Sinon, si β_1 est un littéral et que son complémentaire est dans la liste des littéraux LL, alors on lui affecte comme poids la valeur de la borne β divisée par deux ;
  - Sinon, si β_2 est un littéral et que son complémentaire est dans la liste des littéraux LL, alors on lui affecte comme poids la valeur de la borne β divisée par deux ;
  - Sinon, on affecte comme poids, à la formule F la valeur de la borne β.

4.2.2 Algorithme

Cette méthode suit le schéma récursif suivant :

1. Si la liste de formules LF est vide, on s'arrête.
2. On affecte un poids à chaque formule de la liste de formules LF
3. On ordonne LF par poids croissant.
4. On extrait la première formule F de la liste LF qu'on transforme pour obtenir un littéral, une conjonction, ou une disjonction.
5. On retire cette formule F de la liste LF
6. On distingue trois cas :

   (a) La formule est un littéral.
   On relance la construction de l'arbre sur le reste de la liste de formules LF et la liste de littéraux LL.

   (b) La formule est de type α :

      i. On extrait les deux sous formules α_1 et α_2 ;
      ii. On crée le sous arbre gauche AG de l'arbre courant, qui contiendra la formule α_1 en racine ;
      iii. On affecte l'arbre vide au sous arbre droit de l'arbre courant ;
      iv. On crée le sous arbre gauche AGG de l'arbre AG, cet arbre contiendra la formule α_2 en racine. Le sous arbre droit de l'arbre AG sera vide ;
      v. On teste si la branche créée est close. Si l'une ou l'autre des formules, α_1 ou α_2, sont des littéraux, alors on vérifie si pour chacun leur complémentaire est présent dans l'ensemble des pères du noeud. Si c'est le cas, on s'arrête ;
      vi. Si α_1 est un littéral alors on l'ajoute à la liste de littéraux LL, sinon on l'ajoute à la liste de formules LF.
      vii. Si α_2 est un littéral alors on l'ajoute à la liste de littéraux LL, sinon on l'ajoute à la liste de formules LF.
      viii. On réapplique la méthode *construireArbrePreuve2* sur l'arbre AGG avec en paramètre la liste de formules LF.

   (c) La formule est de type β :

i. On extrait les deux sous formules α_1 et α_2 ;

ii. On crée le sous arbre gauche AG de l'arbre courant, qui contiendra en racine la formule α_1 ;

iii. On crée le sous arbre droit AD de l'arbre courant, contenant en racine la formule α_2 ;

iv. Si α_1 est un littéral et que son complémentaire est présent en remontant dans l'arbre, alors on met à jour l'information de closité pour le noeud contenant α_1 et on va en *viii.* ;

v. On réalise une copie *LFCopie* de la liste de formules LF ;

vi. On réalise une copie *LLCopie* de la liste de littéraux LL ;

vii. Si la formule α_1 n'est pas un littéral alors on l'ajoute à la liste de formules LF, sinon on l'ajoute à la liste de littéraux LL ;

viii. On relance la construction sur l'arbre de preuve AG avec la liste de formules LF et la liste de littéraux LL.

ix. Si α_2 est un littéral et que son complémentaire est présent en remontant dans l'arbre de preuve, on met à jour l'information de closité du noeud contenant α_2 et on s'arrête ;

x. Si la formule α_2 n'est pas un littéral alors on l'ajoute à la liste de formules *LFCopie*,sinon on l'ajoute à la liste de littéraux *LLCopie* ;

xi. On réapplique la méthode *construireArbrePreuve2* sur l'arbre AD avec en paramètre la liste de formules *LFCopie* et la liste de littéraux *LLCopie*.

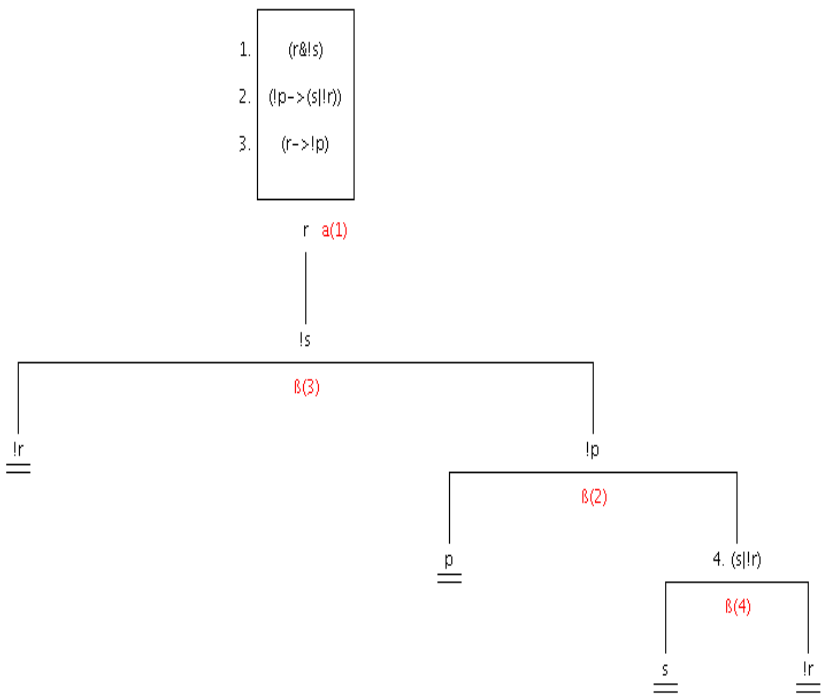

Figure 4: Comparaison des deux procédures de preuve sur le fichier de la figure3

Pour la première stratégie, on obtient la trace $(\alpha(1),\beta(2),\beta(3),\beta(4))$, alors que pour la deuxième stratégie, on obtient la trace $(\alpha(1),\beta(3),\beta(2),\beta(4))$.

Explication de la trace de la première stratégie :
La liste de formules LF est triée de telle sorte que les littéraux soient suivis des formules de type α, puis des formules de type β. Cette stratégie consiste à prendre les règles les unes après les autres.
Dans cette stratégie, on applique la première règle $\alpha(1)$ de la liste de formules LF, puis on vérifie si la branche est close, comme ce n'est pas le cas alors on applique la formule $\beta(2)$, qui est la seconde règle de la liste de formules LF, sur la branche de l'arbre. Ensuite, on vérifie si les branches de l'arbre sont closes et on applique la formule suivante si nécessaire. La stratégie s'arrête quand la liste de formules LF est vide ou quand toutes les branches de l'arbre sont closes.

Explication de la trace de la seconde stratégie :
On affecte un certain poids à chaque formule de telle sorte que les formules de type α aient un poids moins important que les formules de type β. Puis on applique la règle ayant le plus faible poids (sachant que le poids est recalculé après chaque application de formule).
Dans la deuxième stratégie, on applique en premier la règle $\alpha(1)$ de la liste de formules LF, car elle a un poids moins important que les formules du type β. Ensuite, on vérifie si la branche est close, comme ce n'est pas le cas, on applique la formule $\beta(3)$, qui a un poids moins important que la formule $\beta(3)$. Ceci a pour résultat de rendre le sous arbre gauche clos. On applique la formule $\beta(2)$ sur le sous arbre droit, qui conduit directement à une branche close. Il ne reste donc plus qu'a appliquer la sous formule $\beta(4)$ de $\beta(2)$ pour obtenir un arbre clos.

5 Interface graphique

Afin de visualiser, la procédure de preuve d'un ensemble de formules, notre logiciel, *TabSemantique*, est composé d'une interface graphique.

5.1 Fonctionnement de l'interface

Notre logiciel permet, à partir d'un fichier de formules, de visualiser l'arbre de preuve en choisissant la première stratégie ou la deuxième stratégie.

Cette interface graphique est composée d'une barre de menus, une barre d'outils, une zone d'affichage de la liste des formules initiales et une zone de visualisation de la preuve.

Exemple 3 -Exemple -

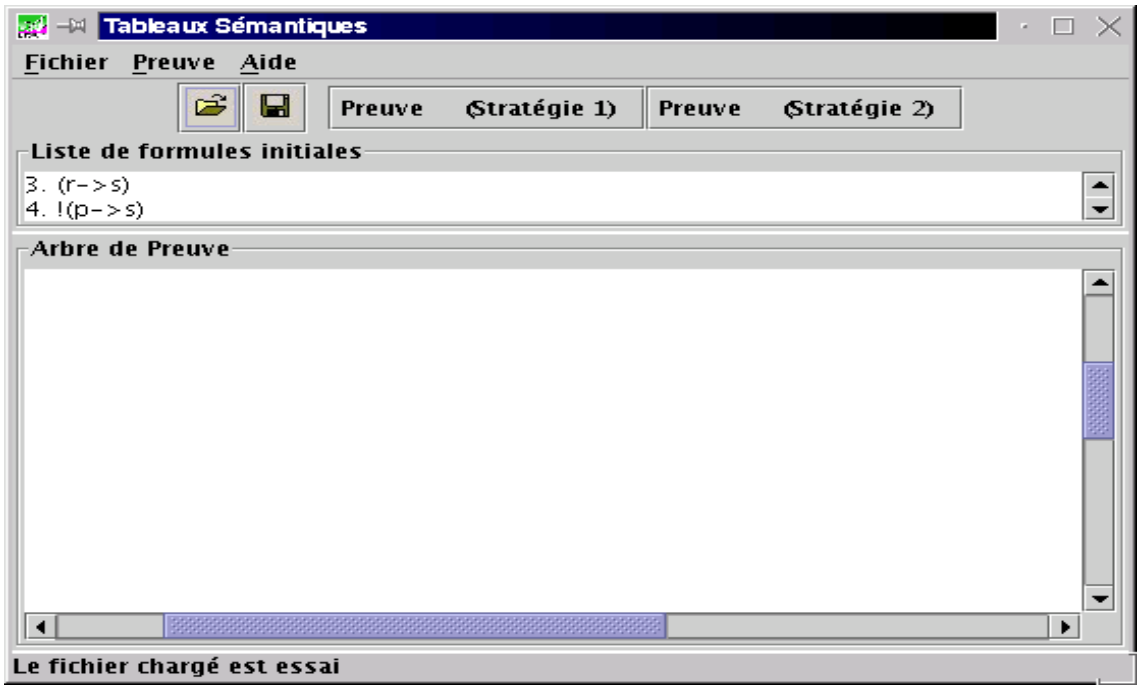

Figure 5: chargement d'un fichier

Détails des différents menus :

Le menu Fichier :
L'utilisateur a la possibilité de charger un fichier à partir de ce menu en choisissant l'option ouvrir, en utilisant les touches de raccourcis *Ctrl-O*, ou en cliquant directement sur l'icône d'ouverture de fichier. Cette action ouvre une autre fenêtre qui permet de sélectionner un fichier. De même, il peut choisir l'option fermer (les touches de raccourcis *Ctrl-F*), qui permet de fermer la preuve en cours ou l'option quitter (les touches de raccourcis *Ctrl-Q*), qui permet de quitter le programme.

Le menu Preuve :
Une fois un fichier chargé, l'utilisateur peut choisir la *stratégie1* ou la *stratégie2* pour résoudre le problème. Ces deux options sont présentes dans la barre d'outils.

Le menu Aide :
Dans ce menu les options ne sont pas nécessaires au déroulement du programme, ou de la preuve, mais permettent d'apporter des aides (l'option *Aide*) sur la marche à suivre pour utiliser le logiciel et des informations complémentaires (l'option *A Propos*) sur le logiciel.

Exemple 4 -Exemple -

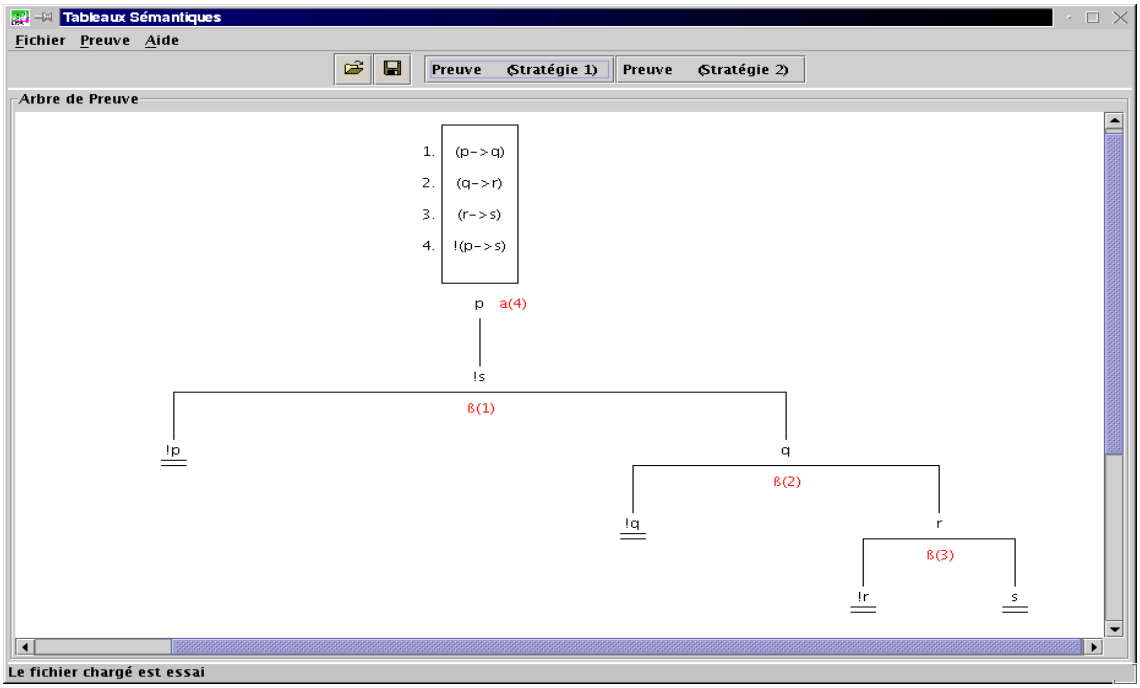

Figure 6: Preuve d'un ensemble de formules.

5.2 Dessin de l'arbre de preuve

La preuve s'affiche dans un composant Java *JPanel* de la bibliothèque *Swing*, avec la prise en compte d'ascenseurs (Scrollbars), dans le cas où la dimension de l'arbre dépasserait les dimensions initiales du composant (1024X768 pixels).

Le dessin de l'arbre est réalisé de manière récursive. Le principe général est d'afficher la racine, puis de réappliquer la méthode de dessin sur le sous-arbre gauche, puis sur le sous-arbre droit.
Afin d'éviter que deux branches de l'arbre ne se superposent, nous divisons par deux la dimension de chaque branche pour chaque niveau de l'arbre. La dimension de départ est calculée selon le nombre maximal de branches comportant des formules de type β.
De plus, nous affichons en couleur rouge les règles appliquées en chaque noeud afin que l'utilisateur puisse visualiser la trace de la procédure de preuve.

6 Conclusion et Perspectives

6.1 Conclusion

Notre logiciel *tabSemantique* permet d'obtenir la procédure de preuve d'un fichier de formules valides.

Ce projet nous a permis d'utiliser certains aspects de la méthode de conception UML. Il a d'abord été nécessaire de discuter sur les structures de données à utiliser et les différents algorithmes à implanter. La modélisation du projet grâce à la méthode UML a d'ailleurs été un aspect intéressant de cette étude.

Nous avons implanté deux méthodes de construction de l'arbre de preuve. La première correspond à une application des règles en privilégiant simplement les règles α par rapport aux règles β, et la seconde fait appel à une stratégie de choix des différentes règles basée sur un calcul pondéral des formules. Les différents tests effectués ont révélé l'efficacité de la deuxième méthode par rapport à la première.

Toutefois, au cours de ce projet, nous avons rencontré plusieurs difficultés. Premièrement, nous avons commencé par utiliser l'outil lexical et syntaxique standard du langage JAVA, *JavaCC*, qui nous a posé de grandes difficultés pour implanter la grammaire. Nous avons donc rapidement utilisé les outils *JFlex* et *Byacc/J* qui se révèlent bien plus pratiques. L'interface graphique et plus particulièrement la gestion de la taille (dynamique) de la zone de dessin de l'arbre de preuve nous a également posé quelques problèmes.

6.2 Perspectives

Il nous semble que ce travail doit être poursuivi et complété dans les directions suivantes :

- On pourrait améliorer l'affectation des poids pour la deuxième stratégie de procédure, en prenant en compte la gestion des sous formules qui ne sont pas des littéraux.
- Une autre amélioration porterait sur l'interface graphique avec la mise en place d'onglets pour comparer directement les arbres de preuves selon les différentes stratégies employées.

7 Bibliographie

References

[Del02] Claude Delannoy. *Programmer en Java.* Eyrolles, 2002.

[Hor01] Ivor Horton. *Maîtrisez Java2.* Wrox, 2001.

[Lam94] Leslie Lamport. *LaTeX a document preparation system(2e édition).* Addison-Wesley, 1994.

[Ric99] Jean-Michel Richer. *Sacre, une Approche de Résolution en Logique fondée sur des techniques de Satisfaction de Contraintes.* PhD thesis, Université de Bourgogne, 1999.

Fichier Parseur.flex

```
%%

%byaccj

%{
        private Parser yyparser;

        public Ylex(java.io.Reader r, Parser yyparser) {
                this(r);
                this.yyparser = yyparser;
        }
%}

EOL  = \n | \r | \r\n

CHIFFRE = [0-9]
/* une lettre peut etre de a-Z ou un underscore */
LETTRE = [a-zA-Z_]

IDENT = {LETTRE}+ ({CHIFFRE}|{LETTRE})*

/* OPERATEURS LOGIQUES */
NON =               "!"
OU =                "|"
ET =                "&"
IMPLIQUE =          "->"
BI_IMPLIQUE =       "<->"
POINT_VIRG =        ";"
PAR_OUVR =          "("
PAR_FERM =          ")"

%%

{EOL}               {}

{IDENT}             { yyparser.yylval = new ParserVal(yytext()); return Parser.IDENT; }

{NON}               { return Parser.NON; }
{OU}                { return Parser.OU; }
{ET}                { return Parser.ET; }
{IMPLIQUE}          { return Parser.IMPLIQUE; }
{BI_IMPLIQUE}       { return Parser.BI_IMPLIQUE; }
{POINT_VIRG}        { return Parser.POINT_VIRG; }
{PAR_OUVR}          { return Parser.PAR_OUVR; }
{PAR_FERM}          { return Parser.PAR_FERM; }


[^]                 { /*System.err.println("Erreur: ècarcatre inconnu '"+yytext()+"'");*/ return -1; }
```

Fichier Parseur.y

```
%{
        import java.io.*;
        import javax.swing.*;
%}

%token EOL
%token NON OU ET IMPLIQUE BI_IMPLIQUE
%token POINT_VIRG PAR_OUVR PAR_FERM
%token <sval> IDENT

%type <obj> Formule

%left OU
%left ET
%left IMPLIQUE BI_IMPLIQUE
%left NON

%%

ListeFormules: ListeFormules Formule POINT_VIRG{
                        Actions.listeFormules.ajoutFormule((Formule)$2);
                }
        | Formule POINT_VIRG{
                        Actions.listeFormules.ajoutFormule((Formule)$1);
                }
        ;

Formule: IDENT                                  { $$ = (Object) new Formule($1,1); }
        | NON IDENT                             { $$ = (Object) new Formule($2,-1); }
        | Formule ET Formule                    { $$ = (Object) new Formule("&",(Formule)$1,(Formule)$3); }
        | Formule OU Formule                    { $$ = (Object) new Formule("|",(Formule)$1,(Formule)$3); }
        | Formule IMPLIQUE Formule              { $$ = (Object) new Formule("->",(Formule)$1,(Formule)$3); }
        | Formule BI_IMPLIQUE Formule           { $$ = (Object) new Formule("<->",(Formule)$1,(Formule)$3); }
        | NON PAR_OUVR Formule  PAR_FERM        { $$ = (Object) new Formule("!",(Formule)$3,null);; }
        | PAR_OUVR Formule PAR_FERM             { $$ = $2; }
        ;

%%

        private Yylex lexer;
        private ZoneDessin zone;
        private boolean erreur = false;


        private int yylex () {
                int yyl_return = -1;
                try {
                        yylval = new ParserVal(0);
                        yyl_return = lexer.yylex();
                }
                catch (IOException e) {
                        // Erreurs Lexicales
                        JOptionPane.showMessageDialog(
                        zone.getParent(),
                        "Erreur d'Entree/Sortie !",
                        "Erreur !",
                        JOptionPane.ERROR_MESSAGE);
                        Actions.listeFormules = new ListeFormules();
                }
                return yyl_return;
        }


        public void yyerror (String error) {
                if (!erreur){
                        // Erreurs Syntaxiques
                        JOptionPane.showMessageDialog(
                        zone.getParent(),
                        "Le contenu du fichier de preuve n'est pas correct !",
                        "Erreur !",
                        JOptionPane.ERROR_MESSAGE);
                        Actions.listeFormules = new ListeFormules();
                }
                erreur = true;
        }


        public Parser(Reader r,ZoneDessin Z) {
                zone = Z;
                lexer = new Yylex(r, this);
        }
```

Fichier TabSemantique.java

```
public class TabSemantique {

    public static void main(String args[]){
            Fenetre fenetre = new Fenetre();
            fenetre.setVisible(true);
    } //fin main

} // fin classe TabSemantique
```

Fichier Symbole.java

```
/**
 * Classe <B>Symhole</B>:
 * un objet de la classe Symbole éreprsente un symbole du èproblme
 * Chaque Symbole est éstock dans une table de symbole
 *
 * @author ALAMICHEL Mathieu
 * @author BOISSINOT Gregory
 * @author MOREAU Olivier
 *
 * @see TableSymboles
 */

public class Symbole{
        static int nbSymbole = 0;
        private String nom;
        private int numero;


/**
 * Constructeur par édfaut
 * @param s nom du nouveau symbole
 */
        public Symbole(String s){
                ++nbSymbole;
                this.nom = s;
                this.numero = nbSymbole;
        }

/**
 * Constructeur
 * @param s nom du nouveau symbole
 * @param num énumro du nouveau symbole
 */
        public Symbole(String s, int num){
                ++nbSymbole;
                this.nom = s;
                this.numero = num;
        }

/**
 * @return le énumro éassociâ  un symbole
 */
        public int getNumero(){
                return this.numero;
        }

/**
 * éMhode d'affection du nom d'un symbole
 * @param s nouveau nom
 */
        public void setNom(String s){
                this.nom = s;
        }

/**
 * @return le nom éassociâ  un symbole
 */
        public String getNom(){
                return this.nom;
        }
}
```

Fichier TableSymboles.java

```
import java.util.Vector;

/**
 * Classe <B>TableSymboles</B>:
 * contient un ensemble de symboles tous distincts
 * Ces symboles sont éutiliss pour édcrire des formules
 *
 * @author ALAMICHEL Mathieu
 * @author BOISSINOT Gregory
 * @author MOREAU Olivier
 *
 * @see Symbole
 * @see Formule
 */

public class TableSymboles extends Vector{

        static final int NON            = 1;
        static final int ET             = 2;
        static final int OU             = 3;
        static final int IMPLIQUE       = 4;
        static final int BIIMPLIQUE     = 5;

/**
 * Constructeur par édfaut
 */
        TableSymboles(){
                this.add(new Symbole("!",NON));
                this.add(new Symbole("&",ET));
                this.add(new Symbole("|",OU));
                this.add(new Symbole("->",IMPLIQUE));
                this.add(new Symbole("<->",BIIMPLIQUE));
        }


/**
 * Ajoute un nouveau Symboleà  la table des symboles
 * Si le symbole existe éàdj on ne le écre pasà  nouveau
 * @param s nom du nouveau symboleà é crer
 * @return l'indice du symbole s dans la table des symboles
 */
        public int ajouterSymbole(String s){
                int num = getSymbole(s); // num de s dans le table des symboles, -1 si n'existe pas

                if (num == -1){
                        Symbole newSymb = new Symbole(s);
                        this.add(newSymb);
                        return newSymb.getNumero();
                }
                else{
                        // s existe éàdj
                        return num;
                }
        }


/**
 * éMthode d'èaccs au symbole d'indice i
 * @param i indice
 * @return le symbole d'indice i
 *
 */
        public Symbole at(int i){
                return (Symbole) this.elementAt(i);
        }

/**
 * Retourne le nom du symbole de numero i
 * @param int indiceà  rechercher
 * @return le nom
 */
        public String getSymbole(int i){
                int j = 0;

                while (j!=this.size()){
                        if ( this.at(j).getNumero() == i){
                                return this.at(j).getNom();
                        }
                        j++;
                }
                return "";
        }

/**
 * Retourne le numero du symbole de nom n
 * @param String nomà  rechercher
 * @return le numero
 */
        public int getSymbole(String n){
                int j = 0;

                while (j!=this.size()){
                        if ( (this.at(j).getNom()).equals(n) ){
                                return this.at(j).getNumero();
                        }
                        j++;
                }

                return -1;
        }
```

```
/**
 * éMthode d'affichage
 * <i>éMthode qui n'est plus éutlise pour l'affichage graphique</i>
 */
        public void afficher(){
                for (int i=0; i<this.size(); i++)
                        System.out.println(i+"\t"+at(i).getNumero()+"\t"+at(i).getNom());
        }

}
```

Fichier Formule.java

```
/**
 * Classe <B>Formule</B>:
 *
 * @author ALAMICHEL Mathieu
 * @author BOISSINOT Gregory
 * @author MOREAU Olivier
 *
 * @see Symbole
 * @see TableSymboles
 * @see ListeFormules
 */

public class Formule{

        /**
         * Table de symbole communeà  toutes les formules
         */
        static TableSymboles tableSymboles = new TableSymboles();

/**
 * indique le poids éattribuà  une formule
 */
        private int poids;

/**
 * numero d'une formule
 */
        private int numero;

/**
 * indice du symbole dans la table de symboles en racine
 */
        private int racine;

/**
 * Formule correspondantà  la partie gauche de la formule
 */
        private Formule sag;
/**
 * Formule correspondatnà  la partie droite de la formule
 */
        private Formule sad;

/**
 * Constructeur
 * @param r indice du symbole
 * @param g sous-arbre gauche
 * @param d sous-arbre droit
 */
        Formule(int symbole, Formule g, Formule d){
                this.racine = symbole;
                this.sag = g;
                this.sad = d;
                this.numero=-1;
        }

/**
 * Constructeur
 * @param num numero de la formule
 * @param r indice du symbole
 * @param g sous-arbre gauche
 * @param d sous-arbre droit
 */
        Formule(int num,int r, Formule g, Formule d){
                this.numero = num;
                this.racine = r;
                this.sag = g;
                this.sad = d;
        }

/**
 * Constructeur
 * @param symbole nom du symbole en racine
 * @param g sous-arbre gauche
 * @param d sous-arbre droit
 */
        Formule(String symbole, Formule g, Formule d){
                this.racine = tableSymboles.ajouterSymbole(symbole);
                this.sag = g;
                this.sad = d;
                this.numero=-1;
        }

/**
 * Constructeur pour un élittral
 * @param symbole nom du symbole ne racine
 * @param signe -1 pour un élittral éngatif, autre chose pour un élittral positif
 */
        Formule(String symbole, int signe){
                if (signe==-1)
                        this.racine = -tableSymboles.ajouterSymbole(symbole);
                else
                        this.racine = tableSymboles.ajouterSymbole(symbole);
                this.sag = null;
                this.sad = null;
        }

/**
 * renvoie la racine de la formule
 * @return la racine de la formule
 */
        public int getRacine() {
```

```
                return this.racine;
        }

/**
 * renvoie le nom du symbole de la formule
 * @return le nom du symbole de la racine
 */
        public String getSymbole(){
                return tableSymboles.getSymbole(this.racine);
        }

/**
 * renvoie la partie gauche de la formule
 * @return la partie gauche de la formule
 */
        public Formule getSag() {
                 return this.sag;
        }

/**
 * renvoie la partie droite de la formule
 * @return la partie droite de la formule
 */
        public Formule getSad() {
                return this.sad;
        }

/**
 * @renvoie le énumro de la formule
 * @return le énumro de la formule
 */
        public int getNumero(){
                return this.numero;
        }

/**
 * @renvoie le poids de la formule
 * @return le poids de la formule
 */
        public int getPoids(){
                return this.poids;
        }

/**
 * Affecteà  la racine l'indice du symbole correspondant
 * @param racine indice du symbole dans la table de symbole
 */
        public void setRacine(int racine) {
                this.racine=racine;
        }

/**
 * Affecteà  la racine une formule en partie gauche
 * @param g nouvelle partie gauche
 */
        public void setSag(Formule g) {
                this.sag=g;
        }

/**
 * Affecteà  la racine une formule en partie droite
 * @param g nouvelle partie droite
 */
        public void setSad(Formule d) {
                this.sad=d;
        }

/**
 * Affecte le numeroà  la formule
 * @param i nouveau énumro
 */
        public void setNumero(int num){
                this.numero = num;
        }

/**
 * Affecte le poidsà  la formule
 * @param p nouveau poids
 */
        public void setPoids(int p){
                this.poids = p;
        }
        /*-----Fin Ascesseurs-----*/

/**
 * Teste si deux formules sont identiques
 * @param F formuleà  comparer
 * @return vrai si les 2 formules sont identiques, faux sinon
 */
        public boolean identique(Formule F){

                if (F == null)
                        return false;
                if (this.getRacine() != F.getRacine())
                        return false;
                Formule sag = this.getSag();
                Formule sad = this.getSad();
                if ((sag == null)  && (sad == null)){
                        return ((F.getSag() == null) && (F.getSad() == null));
                }
                if ((sag == null)  && (sad != null)){
                        return ((F.getSag() == null) && (sad.identique(F.getSad())));
                }
                if ((sag != null)  && (sad == null)){
                        return ((sag.identique(F.getSag())) && (F.getSad() == null));
```

```
                }
                return sag.identique(F.getSag()) && sad.identique(F.getSad());
        }

/**
 * Calcule le écomplmentaire d'une formule
 * @return le écomplmentaire
 */
        public Formule complementaire() {
                if ((this.sag == null) && (this.sad == null)) {
                        Formule F= new Formule(-this.getRacine(),null,null);
                        return F;
                }
                else {
                        Formule F= this.getSag();
                        return F;

                }

        }

/**
 * Teste si une formule est un élittral
 * @return vrai si c'est un élittral, faux sinon
 */
        public boolean isLitteral(){
                //this.transformer();
                if (isNegation()){
                        Formule g = this.sag;
                        if (g.isLitteral()){
                                this.racine = - g.getRacine();
                                this.sag = g.getSag();
                                this.sag = g.getSad();
                        }
                }
                if ((this.sag==null) && (this.sad==null))
                        return true;
                //else if ( (sag!=null) && (this.sag.isLitteral()) && (this.isNegation()))
                //       return true;
                return false;
        }

/**
 * Teste si une formule est une éngation
 * @return vrai si c'est une éngation, faux sinon
 */
        public boolean isNegation(){
                return (this.getRacine()== tableSymboles.NON);
        }

/**
 * Teste si une formule est un Et
 * @return vrai si c'est le cas, faux sinon
 */
        public boolean isEt(){
                return this.getRacine()== tableSymboles.ET;
        }

/**
 * Teste si une formule est un Ou
 * @return vrai si c'est le cas, faux sinon
 */
        public boolean isOu(){
                return this.getRacine()== tableSymboles.OU;
        }
/**
 * Teste si une formule est un implique
 * @return vrai si c'est le cas, faux sinon
 */
        public boolean isImplique(){
                return this.getRacine()== tableSymboles.IMPLIQUE;
        }

/**
 * Teste si une formule est un bi-implique
 * @return vrai si c'est le cas, faux sinon
 */
        public boolean isBiImplique(){
                return this.getRacine()== tableSymboles.BIIMPLIQUE;
        }

/**
 * Teste si une formule est une formule de type <i>alpha</i>
 * @return vrai si c'est le cas, faux sinon
 */
        public boolean isAlpha(){
                //si la racine est un non, il faut que le sous arbre soit une formule beta
                if (this.isNegation())
                        return (this.sag.isBeta());
                else
                        return ((this.isEt()) || (this.isBiImplique()));
        }//fin isAlpha

/**
 * Teste si une formule est une formule de type <i>beta</i>
 * @return vrai si c'est le cas, faux sinon
 */
        public boolean isBeta(){
                if (this.isNegation())
                        return (this.sag.isAlpha());
                else
                        return ((this.isOu()) || (this.isImplique())) ;
        }
```

```
/**
 * @return une copie de la formule
 */
        public Formule copie(){
                if (this == null)
                        return null;
                if ((this.getSag() == null) && (this.getSad() == null))
                        return new Formule(this.getNumero(),this.getRacine(),null,null);
                if ((this.getSag() != null) && (this.getSad() == null))
                        return new Formule(this.getNumero(),this.getRacine(),this.getSag().copie(),null);
                if ((this.getSag() == null) && (this.getSad() != null))
                        return new Formule(this.getNumero(),this.getRacine(),null,this.getSad().copie());
                return new Formule(this.getNumero(),this.getRacine(),this.getSag().copie(),this.getSad().copie());
        }

/**
 * Transforme une formule de telle sorte
 * que la formule érsultante soit de la forme <i>"F1 et F2"</i> ou <i>"F1 ou F2"</i>
 *
 */
        public void transformer(){
                if (this.isNegation()){
                        Formule g = this.sag;
                        if (g.isLitteral()){
                                this.racine = - g.getRacine();
                                this.sag = g.getSag();
                                this.sag = g.getSad();
                        }
                        if (g.isNegation()){
                                Formule gg = g.getSag();
                                this.racine = gg.getRacine();
                                this.sag = gg.getSag();
                                this.sag = gg.getSad();
                        }
                        if (g.isEt()){  // g= F1 & F2
                                Formule F1 = g.getSag();
                                Formule F2 = g.getSad();
                                Formule nvg = new Formule("!",F1,null);
                                Formule nvd = new Formule("!",F2,null);
                                this.racine = tableSymboles.OU;
                                this.sag    = nvg;
                                this.sad    = nvd;
                        }
                        if (g.isOu()){
                                Formule F1 = g.getSag();
                                Formule F2 = g.getSad();
                                Formule nvg = new Formule("!",F1,null);
                                Formule nvd = new Formule("!",F2,null);
                                this.racine = tableSymboles.ET;
                                this.sag    = nvg;
                                this.sad    = nvd;
                        }
                        if (g.isImplique()){
                                //on aurait pu utiliser recursivement la methode
                                Formule F1 = g.getSag();
                                Formule F2 = g.getSad();
                                Formule nvd = new Formule("!",F2,null);
                                this.racine = tableSymboles.ET;
                                this.sag    = F1;
                                this.sad    = nvd;
                        }
                }//fin if (this.isNegation())
                //else if (this.isLitteral()){
                //      return this;
                //}
                else if (this.isOu()){
                        Formule F1 = this.sag;
                        Formule F2 = this.sad;
                        F1.transformer();
                        F2.transformer();
                }
                else if (this.isEt()){
                        Formule F1 = this.sag;
                        Formule F2 = this.sad;
                        F1.transformer();
                        F2.transformer();
                }
                else if (this.isImplique()){
                        Formule F1 = this.sag;
                        Formule F2 = this.sad;
                        Formule nvg = new Formule("!",F1,null);
                        this.racine =  tableSymboles.OU;
                        this.sag    =  nvg;
                }//fin else if (this.isImpplique())
                else if (this.isBiImplique()){
                        Formule F1 = this.sag;
                        Formule F2 = this.sad;
                        Formule nv1 = new Formule("->",F1,F2);
                        Formule nv2 = new Formule("->",F2,F1);
                        this.racine = tableSymboles.ET;
                        this.sag =  nv1;
                        this.sad = nv2;
                }
        }//fin transformer

/**
 * éMthode d'affichage
 * @return une chaine édcrivant la formule: <i>symbole, numero...</i>
 */
        public String toString(){
                if (this.isLitteral()){
                        if (racine>0)
                                return tableSymboles.getSymbole(racine);
                        else
                                return "!"+tableSymboles.getSymbole(-racine);
```

```
                }
                else {
                        String r1 = sag.toString();
                        if (this.isNegation()){
                                return "!"+r1;
                        }
                        else {
                                String r2 = sad.toString();
                                return "("+r1 + tableSymboles.getSymbole(racine) + r2+")";
                        }
                }
        }//fin toString


/**
 * éMthode d'affichage
 * <i>éMthode qui n'est plus éutlise pour l'affichage graphique</i>
 */
        public void afficher(){
                System.out.println(this.toString());
        }

}//fin classe Formule
```

Fichier ListeFormules.java

```
import java.util.Vector;


/**
 * Classe <B>ListeFormules</B>:
 * un objet de la classe <i>ListeFormules</i> contient un ensemble d'objets de la classe <i>Formule</i>
 *
 * @author ALAMICHEL Mathieu
 * @author BOISSINOT Gregory
 * @author MOREAU Olivier
 *
 * @see Formule
 */

public class ListeFormules extends Vector{

/**
 * permet de édterminer combien de formule onté ét éajoutes
 */
        static int nbFormules =0;

/**
 * ééRcupration de la iè me formule
 * @param i indice de la formuleà  chercher
 * @return la formule d'indice i
 */
        public Formule at(int i){
                return (Formule) this.elementAt(i);
        }

/**
 * Teste si c'est vide
 * @return vrai si le liste de formules est vide
 */
        public boolean estVide(){
                return this.size() == 0;
        }

/**
 * Ajoute une formule en édbut de liste
 * @param F la formuleà  ajouter
 */
        public void ajoutFormule(Formule F){
                ++nbFormules;
                F.setNumero(nbFormules);
                this.add(F);
        }

        // ajout d'une formule dans la listeà  la position n
        //public void ajoutFormule(int n,Formule f){
        //      this.add(n,f);
        //} //fin ajoutFormule


/**
 * Teste si une formule appartientà  la liste de formules
 * @param F formuleà  chercher
 * @return true ou false
 */
        public boolean contientFormule(Formule F){
                int i =0;
                while ((i < this.size())){
                        Formule F1 = this.at(i);
                        if (F.identique(F1)){
                                return true;
                        }
                        i++;
                }
                return false;
        }

/**
 * éMhode de copie d'une liste de formules
 * @return une copie ècomplte de la liste de formules
 */
        public ListeFormules copie(){
                ListeFormules nvL = new ListeFormules();
                for (int i =0;i<this.size();i++){
                        Formule F = this.at(i);
                        Formule FCopie = F.copie();
                        nvL.add(FCopie);
                }
                return nvL;
        }

/**
 * Cette émthode éspare de la liste de formules
 * les formules de type <i>alpha</i> (+litteraux)
 * et les formules de type <i>beta</i>
 * <b>Le liste de formules ne contient plus que des formules <i>beta</i></b>
 * @return une liste de formules <i>alpha</i>
 */
        public ListeFormules separer(){
                int max = this.size();
                ListeFormules listeAlpha = new ListeFormules();

                for (int i=0; i!=max; i++){
                        Formule F = this.at(i);
                        //Formule F1 = new Formule(F.getRacine(),F.getSag(),F.getSad());
                        //F1.transformer();
                        F.transformer();
                        if (F.isAlpha()){
```

```
                                listeAlpha.add(F);
                                this.removeElementAt(i);
                                --i;
                                --max;
                        }

                        if (F.isLitteral()){
                                listeAlpha.add(0,F);
                                this.removeElementAt(i);
                                --i;
                                --max;
                        }
                }
                return listeAlpha;
        }

/**
 * Trie une liste de formules
 * pour que les litteraux apparaissent en premier,
 * puis les formules de type <i> alpha</i>
 * et en dernier les formules de type <i>beta</i>
 */
        public void triStrategie1(){
                ListeFormules listeAlpha = this.separer();
                this.addAll(0,listeAlpha);
        }


/**
 * Recherche les élittreaux de la liste de formules
 * @return la liste des élittraux de la liste de formules
 */
        public ListeFormules calculeListLitteraux (){
                ListeFormules nv = new ListeFormules();
                for (int i=0;i<this.size();i++){
                        Formule F = this.at(i);
                        if (F.isLitteral()){
                                nv.add(F);
                        }
                }
                return nv;
        }


/**
 * Trie la liste selon la èdeuxime éstratgie
 */
        public void triStrategie2(ListeFormules LL) {
                //borne maximale
                int borne = 100;
                int borneAlpha = borne /4;
                int borneBeta  = borne /2;

                //Affectation de poids
                for (int i=0;i<this.size();i++){
                        //ééRcupration de la èpremire formule
                        Formule F = this.at(i);
                        F.transformer();

                        //La formule F est de type alpha
                        if (F.isAlpha()){
                                //On lui affecte par édfaux le poids d'une formule alpha
                                F.setPoids(borneAlpha);
                                //ééRcupration des deux sous formules
                                Formule alpha1 = F.getSag();
                                Formule alpha2 = F.getSad();
                                if (LL!=null) {
                                        if ((alpha1.isLitteral()) && (LL.contientFormule(alpha1.complementaire()))){
                                                F.setPoids(10);
                                        }
                                        else if ((alpha2.isLitteral()) && (LL.contientFormule(alpha2.complementaire()))) {
                                                F.setPoids(10);
                                        }
                                }
                        }//fin if (F.isAlpha())

                        else if (F.isOu()){
                                F.setPoids(50);
                                Formule alpha1 = F.getSag();
                                Formule alpha2 = F.getSad();

                                if (LL!=null){

                                        if( (alpha1.isLitteral()) && (alpha2.isLitteral())  &&
                                            (LL.contientFormule(alpha1.complementaire()))   &&
                                            (LL.contientFormule(alpha1.complementaire()))){
                                                F.setPoids(15);
                                        }
                                        if((alpha1.isLitteral()) && (LL.contientFormule(alpha1.complementaire()))) {
                                                F.setPoids(30);
                                        }
                                        else if ((alpha2.isLitteral()) && (LL.contientFormule(alpha2.complementaire()))){
                                                F.setPoids(30);
                                        }
                                }
                        }//fin if (F.isOu())

                        else if (F.isImplique()){
                                F.setPoids(50);
                                Formule alpha1 = F.getSag();
                                Formule alpha2 = F.getSad();

                                if (LL!=null){
                                        if((alpha1.isLitteral()) && (LL.contientFormule(alpha1))) {
                                                F.setPoids(30);
```

```
                                        }
                                        else if ((alpha2.isLitteral()) && (LL.contientFormule(alpha2))){
                                                F.setPoids(30);
                                        }
                                }
                        }//fin if (F.isImplique())

                        //On fixe les élittreaux avec un poids de ézro.
                        else {
                                F.setPoids(0);
                        }
                }//fin for
                //Fin Affectation des poids
                this.trieCroissantPoids();

        }//fin strategie2


/**
 * éRalise un tri croissant des poids de la liste de formules LF
 */
        public void trieCroissantPoids() {
                int taille = this.size();
                for (int i=0;i<taille;i++){
                        for (int j=i+1;j<taille;j++){
                                Formule F1 = this.at(i);
                                Formule F2 = this.at(j);
                                if (F1.getPoids() > F2.getPoids()) {
                                        this.add(i,F2);
                                        this.removeElementAt(i+1);
                                        this.add(j,F1);
                                        this.removeElementAt(j+1);
                                }
                        }//fin for
                }//fin for
        }//fin triCroissentPoids


/**
 * éMthode d'affichage
 * <i>éMthode qui n'est plus éutlise pour l'affichage graphique</i>
 */
        public void afficher() {
                for (int i=0;i<this.size();i++){
                        System.out.println("formule " + i + " : ");
                        this.at(i).afficher();
                }
         }
}
```

Fichier Noeud.java

```
/**
 * Classe <B>Noeud</B>:
 * un objet de la classe noeud éreprsente un noeud de l'arbre de preuve
 *
 * @author ALAMICHEL Mathieu
 * @author BOISSINOT Gregory
 * @author MOREAU Olivier
 */

public class Noeud{
        private Formule formule;
        private boolean noeudClos;
        private String regle;

/**
 * Constructeur par édfaut
 */
        public Noeud(){
                this.formule   = null;
                this.noeudClos = false;
                this.regle = "";
        }

/**
 * Constructeur pour une formule F
 * @param F formule constituant le noeud
 */
        public Noeud(Formule F){
                this.formule   = F;
                this.noeudClos = false;
                this.regle = "";
        }

/**
 * Constructeur pour une feuille
 * @param F formule constituant le noeud
 * @param clos determine si le noeud est clos ou non
 */
        public Noeud(Formule F, boolean clos){
                this.formule    = F;
                this.noeudClos = clos;
                this.regle="";
        }

/**
 * Constructeur de la classe Noeud pour une noeud clos
 *
 * @param F formule constituant le noeud
 * @param clos édtermine si le noeud est clos ou non
 * @param r èrgle éapplique au noeud
 */
        public Noeud(Formule F,boolean clos,String r){
                this.formule    = F;
                this.noeudClos = clos;
                this.regle = r;
        }

/**
 * @return la formule du noeud
 */
        public Formule getFormule(){
                  return this.formule;
        }

        public boolean getNoeudClos(){
                 return this.noeudClos;
        }
        public String getRegle(){
                 return this.regle;
        }
        public void setFormule(Formule F){
                 this.formule = F;
        }

        public void setNoeudClos(boolean clos){
                 this.noeudClos = clos;
        }
        public void setRegle(String r){
                 this.regle = r;
        }

}//fin classe Noeud
```

Fichier ArbrePreuve.java

```
class ArbrePreuve {

        private Noeud racine;
        private ArbrePreuve sag;
        private ArbrePreuve sad;
        private ArbrePreuve prec;


        /* CONSTRUCTEURS */

        public ArbrePreuve(){
                this.racine = null;
                this.sag    = null;
                this.sad    = null;
                this.prec   = this;
        }

        public ArbrePreuve(ArbrePreuve p){
                this.racine = null;
                this.sag = null;
                this.sad = null;
                this.prec = p;
        }


       /*----------Ascesseurs---------*/
        public ArbrePreuve  getSag(){
                return this.sag;
        }

        public ArbrePreuve getSad(){
                return this.sad;
        }

        public Noeud getRacine(){
                return this.racine;
        }

        public ArbrePreuve getPrecedent(){
                return prec;
        }

        public void setRacine(Noeud n){
                this.racine = n;
        }

        public void setSag(ArbrePreuve g) {
                this.sag = g;
        }

        public void setSad(ArbrePreuve d) {
                this.sad = d;
        }
       /*---------Fin Ascesseurs---------*/


        // renvoie vrai si l'arbre de preuve est vide
        public boolean estVide(){
                return ((this.racine == null)&&(sag==null)&&(sad==null));
        }

        // renvoie vrai si on està la racine de l'arbre de preuve
        public boolean estRacine(){
                return (prec == this);
        }

        // renvoie vrai si l'arbre de preuve est une feuille
        public boolean estFeuille(){
            return ((this.racine != null) && (sad == null) && (sag==null));
        }


        // recherche d'un litteralà la émonte dans un arbre de preuve
        // ou dans la liste de formules intiales
        public boolean rechercheLitteral(Formule litteral,ListeFormules LI){
                ArbrePreuve abPrec = this.prec;
                if (abPrec == null)
                        return LI.contientFormule(litteral);
                if (abPrec.getRacine() == null){
                        return LI.contientFormule(litteral);
                }
                if (abPrec.getRacine().getFormule() == null){
                        return LI.contientFormule(litteral);
                }
                if (abPrec.estRacine()){
                        return (abPrec.getRacine().getFormule().identique(litteral));
                }
                else{
                        Formule formulePrec = abPrec.getRacine().getFormule();
                        // on teste si la formule éé prcdente est un litteral
                        // Si c'est un litteral on teste si il esté galà l
                        // sinon on teste au dessus dans l'arbre
                        if ((formulePrec != null)&&(formulePrec.identique(litteral))) {
                                return true;
                        }
                        return abPrec.rechercheLitteral(litteral,LI);
                }
        }


        //Renvoie la hauteur de l'arbre de preuve
        public int hauteur(){
```

```
        if (this.estVide())
                return 0;
        if ((this.sag == null) && (this.sad == null))
                return 1;
        if ((this.sag!=null) && (this.sad == null))
                return 1 + this.sag.hauteur();
        if ((this.sag==null) && (this.sad != null))
                return 1 + this.sad.hauteur();
        return 1 + Math.max(this.sag.hauteur(), this.sad.hauteur());
}

//Renvoie la largeur Maximale de l'arbre
public int largeurMax(){
        if (this.estFeuille()){
                return 1;
        }
        else{
                if (this.sad != null){
                        int Hsad = this.sad.hauteur();  // hauteur du sous-arbre droit
                        int Hsag = this.sag.hauteur();  // hauteur du sous-arbre gauche
                        if (Hsad >= Hsag)
                                return 2*this.sad.largeurMax();
                        else
                                return 2*this.sag.largeurMax();
                }
                else
                        return this.sag.largeurMax();
        }
}


//Renvoie le nombre de feuilles
public int nbFeuilles(){
        if (this.estVide())
                return 0;
        if ((this.sag == null) && (this.sad == null))
                return 1;
        if ((this.sag != null) && (this.sad == null))
                return this.sag.nbFeuilles();
        if ((this.sag == null) && (this.sad != null))
                return this.sad.nbFeuilles();
        return this.sag.nbFeuilles() + this.sad.nbFeuilles();
}//fin nbFeuilles


//Renvoie le nombre de noeuds
public int nbNoeuds(){
        if (this.estVide())
                return 0;
        if ((this.sag == null) && (this.sad == null))
                return 1;
        if ((this.sag != null) && (this.sad == null))
                return 1+this.sag.nbFeuilles();
        if ((this.sag == null) && (this.sad != null))
                return 1+this.sad.nbFeuilles();
        return 1+this.sag.nbFeuilles() + this.sad.nbFeuilles();
}

//Renvoie le nombre de disjonction
public int nbOuMax(){
        if (this.estVide())
                return 0;
        if ((this.sag == null) && (this.sad == null))
                return 0;
        if ((this.sag != null) && (this.sad == null))
                return this.sag.nbOuMax();
        if ((this.sag == null) && (this.sad != null))
                return this.sad.nbOuMax();
        return 1+ Math.max(this.sag.nbOuMax(), this.sad.nbOuMax());
}



//Construit un arbre de preuve avec la èpremire éstratgie
public void construireArbrePreuve1(ListeFormules LF,ListeFormules LI){
        if (!LF.estVide()){
                this.construireArbrePreuve1NonVide(LF,LI);
        }
}

//Construit un arbre de preuve avec la èdeuxime éstratgie
public void construireArbrePreuve2(ListeFormules LF,ListeFormules LI,ListeFormules LL){
        if (!LF.estVide()){
                this.construireArbrePreuve2NonVide(LF,LI,LL);
        }
}

//Sous émthode de construireArbrePreuve1
private void construireArbrePreuve1NonVide(ListeFormules LF,ListeFormules LI){
        // trie la liste pour obtenir littreaux,alpha,beta
        LF.triStrategie1();
        // extraction de la èpremire formule
        Formule F = LF.at(0);
        // on transforme la formule en conjonction ou disjonction si possible
        F.transformer();
        // on retire cette formule de la liste de formules
        LF.removeElementAt(0);
        //La formule F est une conjonction

        if (F.isEt()){
                //Mise a jour de la racine en stockant la èrgle éapplique
                String regle = "a("  + String.valueOf(F.getNumero())+")";
                if (this.racine == null){
                        this.racine = new Noeud(null,false,regle);
                }
```

```
                    else{
                            this.racine = new Noeud(this.racine.getFormule(),false,regle);
                    }
                    //Extraction des deux sous formules
                    Formule alpha1 =  F.getSag();
                    Formule alpha2 =  F.getSad();
                    ArbrePreuve AG,AGG;
                    //éCration  du premier sous arbre gauche pour alpha1
                    AG = new ArbrePreuve(this);
                    Noeud nAlpha1 = new Noeud(alpha1);
                    AG.setRacine(nAlpha1);
                    this.sag = AG;
                    this.sad = null;
                    //éCrattion du èdeuxime sous arbre gauche pour alpha2
                    AGG = new ArbrePreuve(AG);
                    Noeud nAlpha2 = new Noeud(alpha2);
                    AGG.setRacine(nAlpha2);
                    AG.setSag(AGG);
                    AG.setSad(null);
                    //Test de éclosit
                    if ((alpha1.isLitteral()) && (AG.rechercheLitteral(alpha1.complementaire(),LI))){
                            (AGG.racine).setNoeudClos(true);
                    }
                    else
                    if ((alpha2.isLitteral()) && (AGG.rechercheLitteral(alpha2.complementaire(),LI))){
                            (AGG.racine).setNoeudClos(true);
                    }
                    //sinon relance de la construction
                    else {
                            //On ajoute alpha1 et alpha2 a liste de formules
                            //seulement si chacun ne sont pas des élittraux
                            if (!alpha1.isLitteral())
                                    LF.ajoutFormule(alpha1);
                            if (!alpha2.isLitteral())
                                LF.ajoutFormule(alpha2);
                            AGG.construireArbrePreuve1(LF,LI);
                    }
            }//fin if (F.isEt())

            //La formule Fest une disjonction
            else if (F.isOu()){
                    String regle = ß"(" + String.valueOf(F.getNumero()+")");
                    //Mise a jour de la racine en stockant la èrgle éapplique
                    if (this.racine==null)
                            this.racine = new Noeud(null,false,regle);
                    else
                            this.racine = new Noeud(this.racine.getFormule(),false,regle);
                    //Extraction des deux sous formules
                    Formule alpha1 =  F.getSag();
                    Formule alpha2 =  F.getSad();
                    alpha1.transformer();
                    alpha2.transformer();
                    ArbrePreuve AG,AD;
                    //éCration  du premier sous arbre gauche pour alpha1
                    AG = new ArbrePreuve(this);
                    //éCration  du èdeuxime sous arbre gauche pour alpha2
                    AD = new ArbrePreuve(this);
                    Noeud nAlpha1 = new Noeud(alpha1);
                    Noeud nAlpha2 = new Noeud(alpha2);
                    AG.setRacine(nAlpha1);
                    AD.setRacine(nAlpha2);
                    this.sag = AG;
                    this.sad = AD;
                    //Copie de la liste de formules
                    ListeFormules LFCopie = LF.copie();
                    if ((alpha1.isLitteral()) && (AG.rechercheLitteral(alpha1.complementaire(),LI))){
                            (AG.racine).setNoeudClos(true);
                    }
                    else {
                            if (!alpha1.isLitteral()) LF.ajoutFormule(alpha1);
                            AG.construireArbrePreuve1(LF,LI);
                    }
                    if ((alpha2.isLitteral()) && (AD.rechercheLitteral(alpha2.complementaire(),LI))){
                            (AD.racine).setNoeudClos(true);
                    }
                    else {
                            if (!alpha2.isLitteral()) LFCopie.ajoutFormule(alpha2);
                            AD.construireArbrePreuve1(LFCopie,LI);
                    }
            }//fin if (F.isOu())

            //Sinon la formule est un élittral
            //on relance la construction sur le reste de la liste de formules
            else {
                            this.construireArbrePreuve1(LF,LI);
            }
    }//fin construireArbrePreuve1NonVide



    //Sous émthode de construireArbrePreuve2
    private void construireArbrePreuve2NonVide(ListeFormules LF,ListeFormules LI,ListeFormules LL){
            //Trie la liste LF sous la èdeuxime éstratgie
            LF.triStrategie2(LL);
            //Extraction de la èpremire formule
            Formule F = LF.at(0);
            //On transforme cette formule en une conjonction ou disjocntion si possible
            F.transformer();
            //On retire cette formule de la liste de formules
            LF.removeElementAt(0);

            //La formule F est une conjonction
            if (F.isEt()){
                    //Mise a jour de la racine en stockant la èrgle éapplique
```

```
                        String regle = "a("  + String.valueOf(F.getNumero())+")";
                        if (this.racine == null){
                                this.racine = new Noeud(null,false,regle);
                        }
                        else{
                                this.racine = new Noeud(this.racine.getFormule(),false,regle);

                        }
                        //Extraction des deux sous formules
                        Formule alpha1 =  F.getSag();
                        Formule alpha2 =  F.getSad();
                        ArbrePreuve AG,AGG;
                        //éCration du premier sous arbre gauche pour alpha1
                        AG = new ArbrePreuve(this);
                        Noeud nAlpha1 = new Noeud(alpha1);
                        AG.setRacine(nAlpha1);
                        this.sag = AG;
                        this.sad = null;
                        //éCration du èdeuxime sous arbre gauche pour alpha2
                        AGG = new ArbrePreuve(AG);
                        Noeud nAlpha2 = new Noeud(alpha2);
                        AGG.setRacine(nAlpha2);
                        AG.setSag(AGG);
                        AG.setSad(null);
                        //Test de la éclosit
                        if ((alpha1.isLitteral()) && (AG.rechercheLitteral(alpha1.complementaire(),LI))){
                                (AGG.racine).setNoeudClos(true);
                        }
                        else
                        if ((alpha2.isLitteral()) && (AGG.rechercheLitteral(alpha2.complementaire(),LI))){
                                (AGG.racine).setNoeudClos(true);
                        }
                        //sinon on relance la construction
                        else {
                                if (alpha1.isLitteral()) LL.add(alpha1);
                                else LF.ajoutFormule(alpha1);
                                if (alpha2.isLitteral()) LL.add(alpha2);
                                else LF.ajoutFormule(alpha2);
                                AGG.construireArbrePreuve2(LF,LI,LL);
                        }
                }//fin if (F.isEt())

                //La formule F est une disjonction
                else if (F.isOu()){
                        //Mise a jour de la racine en stockant la èrgle éapplique
                        String regle = ß"(" + String.valueOf(F.getNumero()+")");
                        if (this.racine==null)
                                this.racine = new Noeud(null,false,regle);
                        else
                                this.racine = new Noeud(this.racine.getFormule(),false,regle);

                        //Extraction des sous formules
                        Formule alpha1 =  F.getSag();
                        Formule alpha2 =  F.getSad();
                        alpha1.transformer();
                        alpha2.transformer();
                        ArbrePreuve AG,AD;
                        //éCration du sous arbre gauche
                        AG = new ArbrePreuve(this);
                        //éCration du sous arbre droit
                        AD = new ArbrePreuve(this);
                        Noeud nAlpha1 = new Noeud(alpha1);
                        Noeud nAlpha2 = new Noeud(alpha2);
                        AG.setRacine(nAlpha1);
                        AD.setRacine(nAlpha2);
                        this.sag = AG;
                        this.sad = AD;
                        //Copie de la liste de formules
                        ListeFormules LFCopie = LF.copie();
                        ListeFormules LLCopie = LL.copie();
                        if ((alpha1.isLitteral()) && (AG.rechercheLitteral(alpha1.complementaire(),LI))){
                                (AG.racine).setNoeudClos(true);
                        }
                        else {
                                if (alpha1.isLitteral()) LL.add(alpha1);
                                else LF.ajoutFormule(alpha1);
                                AG.construireArbrePreuve2(LF,LI,LL);
                        }
                        if ((alpha2.isLitteral()) && (AD.rechercheLitteral(alpha2.complementaire(),LI))){
                                (AD.racine).setNoeudClos(true);
                        }
                        else {
                                if (alpha2.isLitteral()) LLCopie.add(alpha2);
                                else     LFCopie.ajoutFormule(alpha2);
                                AD.construireArbrePreuve2(LFCopie,LI,LLCopie);
                        }
                }//fin if (F.isOu())

                //Sinon la formule F est un élittral, on relance sur le reste de la liste LF
                else {
                        this.construireArbrePreuve2(LF,LI,LL);
                }
        }//fin construireArbrePreuveNonVide2


}//fin classe ArbrePreuve
```

Fichier BarreMenus.java

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class BarreMenus extends JMenuBar {

	//Le menu Fichier
	private JMenu menuFichier = new JMenu("Fichier");
	private JMenuItem itemNouveau, itemOuvrir,itemEnregistrer,itemImprimer,itemFermer,itemQuitter;
	//Le menu Preuve
	private JMenu menuPreuve = new JMenu("Preuve");
	private JMenuItem itemStrategie1;
	private JMenuItem itemStrategie2;
	//Le menu d'aide
	private JMenu menuAide = new JMenu("Aide");
	private JMenuItem itemAPropos, itemAide;

	public BarreMenus(ZoneDessin zone,Fenetre fen){

		/*---------Menu Fichier---------*/
		//Creation des items
		itemOuvrir      = new JMenuItem("Ouvrir",new ImageIcon("./images/ouvrir1.jpg"));
		itemEnregistrer = new JMenuItem("Enregistrer",new ImageIcon("./images/enregistrer1.jpg"));
		itemImprimer    = new JMenuItem("Imprimer", new ImageIcon("./images/imprimer1.jpg"));
		itemFermer      = new JMenuItem("_______Fermer");
		itemQuitter     = new JMenuItem("_______Quitter");
		//Ajout des items
		menuFichier.add(itemOuvrir);
		menuFichier.addSeparator(); //insere une ligne
		menuFichier.add(itemEnregistrer);
		menuFichier.addSeparator();
		menuFichier.add(itemFermer);
		menuFichier.addSeparator();
		menuFichier.add(itemQuitter);
		menuFichier.setMnemonic('F');
		itemOuvrir.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,InputEvent.CTRL_MASK));
		itemEnregistrer.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,InputEvent.CTRL_MASK));
		itemFermer.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F,InputEvent.CTRL_MASK));
		itemQuitter.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Q,InputEvent.CTRL_MASK));
		//Ajout du menu
		this.add(menuFichier);
		/*-----------Fin Menu Fichier-----------*/

		/*---------------Menu Preuve---------------*/
		//éCration des items
		itemStrategie1 = new JMenuItem("Preuve (Strategie 1)");
		itemStrategie2 = new JMenuItem("Preuve (Strategie 2");
		//Ajout de l'item
		menuPreuve.add(itemStrategie1);
		menuPreuve.add(itemStrategie2);
		//Racourcis claviers
		menuPreuve.setMnemonic('P');
		//Ajout du menu
		this.add(menuPreuve);
		/*---------Fin Menu Preuve---------------*/

		/*-------------Menu d'aide---------------*/
		//Creation des items
		itemAPropos = new JMenuItem("A propos...");
		itemAide = new JMenuItem("Aide",new ImageIcon("./images/aide.jpg"));
		//Ajout des items
		menuAide.add(itemAide);
		menuAide.addSeparator(); //insere une ligne
		menuAide.add(itemAPropos);
		//Police
		//menuAide.setFont(new Font("Helvetica", Font.PLAIN,14));
		//Racourcis clavier
		menuAide.setMnemonic('A');
		//Ajout du menu
		this.add(menuAide);
		/*--------Fin Menu Aide-------------------*/

		//gestion desé èvnements
		itemOuvrir.addActionListener(new Actions(Actions.OUVRIR ,zone,fen));
		itemFermer.addActionListener(new Actions(Actions.FERMER,zone,fen));
		itemStrategie1.addActionListener(new Actions(Actions.PREUVE1,zone,fen));
		itemStrategie2.addActionListener(new Actions(Actions.PREUVE2,zone,fen));
		itemAide.addActionListener(new Actions(Actions.AIDE,zone,fen));
		itemAPropos.addActionListener( new Actions(Actions.APROPOS, zone,fen));
		itemQuitter.addActionListener( new Actions(Actions.QUITTER, zone, fen));
		this.init(0);
	}//fin constructeur BarreMenus

	public void init(int etat){
		if (etat == 0) {
			this.itemFermer.setEnabled(false);
			this.itemImprimer.setEnabled(false);
			this.itemEnregistrer.setEnabled(false);
			this.itemStrategie1.setEnabled(false);
			this.itemStrategie2.setEnabled(false);
		}
		else {
			this.itemFermer.setEnabled(true);
			this.itemImprimer.setEnabled(true);
			this.itemEnregistrer.setEnabled(true);
			this.itemStrategie1.setEnabled(true);
			this.itemStrategie2.setEnabled(true);
		}
	}
}//fin classe BarreMenus
```

Fichier BarreOutils.java

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class BarreOutils extends JToolBar {

             private JButton boutonNouveau , boutonOuvrir , boutonEnregistrer, boutonPreuve1, boutonPreuve2;

          public BarreOutils(ZoneDessin zone,Fenetre fen) {

                   this.setFloatable(false);
                   GridBagLayout gbl = new GridBagLayout();
                   GridBagConstraints gbc = new GridBagConstraints();
                   setLayout(gbl);
                   //ajout et diposition de bouton
                          gbc.gridwidth=1;
                          gbc.gridheight=1;
                     gbc.ipadx=-3;
                     gbc.ipady=-3;
                   //insertion ouvrir
                   Icon iconouvrir0= new ImageIcon("images/ouvrir0.jpg");
                   Icon iconouvrir1= new ImageIcon("images/ouvrir1.jpg");
                   Icon iconouvrir2= new ImageIcon("images/ouvrir2.jpg");
                   boutonOuvrir= bouton(iconouvrir1,gbl,gbc);
                   boutonOuvrir.setRolloverIcon(iconouvrir0);
                   boutonOuvrir.setSelectedIcon(iconouvrir2);
                   boutonOuvrir.setToolTipText("ouvrir un fichier");
                   //insertion enregistrer
                   Icon iconenregistrer0= new ImageIcon("images/enregistrer0.jpg");
                   Icon iconenregistrer1= new ImageIcon("images/enregistrer1.jpg");
                   Icon iconenregistrer2= new ImageIcon("images/enregistrer2.jpg");
                   boutonEnregistrer= bouton(iconenregistrer1,gbl,gbc);
                   boutonEnregistrer.setRolloverIcon(iconenregistrer0);
                   boutonEnregistrer.setSelectedIcon(iconenregistrer2);
                   boutonEnregistrer.setToolTipText("sauver le dessin de l'arbre de preuve");
                   //insertion Preuve
                   addSeparator();
                   boutonPreuve1 = new JButton("Preuve (éStratgie 1)");
                   boutonPreuve2 = new JButton("Preuve (éStratgie 2)");
                   this.add(boutonPreuve1);
                   this.add(boutonPreuve2);
                   boutonPreuve1.setToolTipText("Appliquer la èpremire éstratgie.");
                   boutonPreuve2.setToolTipText("Appliquer la èdeuxime éstratgie.");
                   //association des boutons avec leurs evenements
                   boutonOuvrir.addActionListener(new Actions(Actions.OUVRIR, zone, fen));
                   boutonPreuve1.addActionListener(new Actions(Actions.PREUVE1, zone, fen));
                   boutonPreuve2.addActionListener(new Actions(Actions.PREUVE2, zone, fen));
                   this.init(0);
              } //fin constructeur BarreOutils

          //methode de creation de bouton
            public JButton bouton (Icon icone, GridBagLayout gbl, GridBagConstraints gbc){
                       JButton b= new JButton(icone);
                   gbl.setConstraints(b,gbc);
                   add(b);
          return b;
          }//fin bouton


            public void init(int etat) {
                         if (etat == 0){
                           boutonEnregistrer.setEnabled(false);
                           boutonPreuve1.setEnabled(false);
                           boutonPreuve2.setEnabled(false);
                   }
                   else{
                           boutonEnregistrer.setEnabled(true);
                           boutonPreuve1.setEnabled(true);
                           boutonPreuve2.setEnabled(true);
                   }
    }

} // fin classe BarreOutils
```

Fcihier BarreEtat.java

```
import java.awt.*;
import javax.swing.*;

public class BarreEtat extends JPanel {

        private JLabel info = new JLabel();             //zone d'information

        public BarreEtat(){
                this.setBackground(Color.lightGray);  //couleur
                this.setLayout(new GridLayout(1,1));  //1 ligne et 1 colonne
                this.add(info);
                this.init();
        } //fin constructeur BarreEtat

        public void init(){
                this.afficher("Aucun fichier n'est écharg.");
        }


        public void afficher(String message)  {
                info.setText(message);
        }//fin afficheMessage

} //fin classe BarreEtat
```

Fcihier ZoneDessin.java

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

class ZoneDessin extends JPanel{


        private int dimZoneX;
        private int dimZoneY;
        private ArbrePreuve arbrePreuve = new ArbrePreuve();
        private int dimBranche=250;         //dimension d'une branche
        private int hauteurDessinArbre;     //hauteur de l'arbre dans le dessin
        private int largeurDessinArbre;     //largeur de l'arbre dans le dessin
        private BarreEtat barreEtat;        //Barre dé'tat en bas de la zone de dessin
        private static Fenetre fen;



        public ZoneDessin(BarreEtat barreEtat, Fenetre fen){
                this.setBackground(Color.white);
                this.barreEtat= barreEtat;
                this.fen=fen;
        } //fin constructeur ZoneDessin



        public void setArbrePreuve(ArbrePreuve AP){
                this.arbrePreuve = AP;
        }

        public void paintComponent(Graphics g){
                this.dimBranche = 250;
                super.paintComponent(g);
                this.setPreferredSize(new Dimension(1024,768));
                this.revalidate();
                dessinArbrePreuve(g);
        }

        public void essai(int posX){
                Graphics g = getGraphics();
                dessineListeFormulesInitiales(g,posX);
        }


        public void dessineListeFormulesInitiales(Graphics g, int posX){
                int decaYFormule =30;    //édcalage verticale entre les formules
                int debY = 20;           //édbut verticale de l'arbre
                int x = posX;
                int y = debY;
                int longFormule;         //longeur de la formule
                int longNumero;
                int longMax =0;
                FontMetrics fm = g.getFontMetrics();
                ListeFormules LI = Actions.listeFormules;
                for (int i=0;i<LI.size();i++){
                        y+=decaYFormule;
                        Formule F = LI.at(i);
                        String chaineFormule =  F.toString();
                        longFormule = fm.stringWidth(chaineFormule);
                        x = x - (longFormule/2);
                        int longChaine = longFormule;
                        if (longChaine > longMax) {
                                longMax =  longChaine;
                        }
                        g.drawString(chaineFormule,x,y);
                        x =posX;
                }//fin for

                //Afiichage du cadre autour de la liste des formules intiales
                //et des énumros de formules correspondant
                y = debY;
                for (int i=0;i<LI.size();i++){
                        int decaXCadreFormule = 10;
                        int largeurCadre = longMax+decaXCadreFormule*2;
                        int hauteurCadre = (LI.size() + 1)*decaYFormule + 1;
                        Formule F = LI.at(i);
                        String chaineNumero  =  String.valueOf(F.getNumero())+ ".␣";
                        longNumero  = fm.stringWidth(chaineNumero);
                        y+= decaYFormule;
                        g.drawString(chaineNumero,posX-largeurCadre/2 -longNumero,y);
                        g.drawLine(posX-largeurCadre/2,debY,posX+largeurCadre/2,debY);
                        g.drawLine(posX-largeurCadre/2,debY,posX-largeurCadre/2,debY+hauteurCadre);
                        g.drawLine(posX+largeurCadre/2,debY,posX+largeurCadre/2,debY+hauteurCadre);
                        g.drawLine(posX-largeurCadre/2,debY+hauteurCadre,posX+largeurCadre/2,debY+hauteurCadre);
                }//fin for
        }//fin dessineFormuleInitiales


        public void dessinArbrePreuve(Graphics g){


                ArbrePreuve AP =  this.arbrePreuve;
                int milieuX ;
                if (!AP.estVide()){
                        //fen.getZoneFormules().setVisible(false);
                        //fen.getScrollZoneFormules().setVisible(false);
                        hauteurDessinArbre =   AP.hauteur() * 80;
                        int s = 0;
                        for (int i=1;i <= AP.nbOuMax()-1;i++){
                                s =  s + (this.dimBranche / (2 * i));

                        }
```

```
                largeurDessinArbre = (s+this.dimBranche)*2 + 100;
                if ((largeurDessinArbre < 1024) &&(hauteurDessinArbre < 768)) {
                        this.setPreferredSize(new Dimension(1024,768));
                        milieuX = 1024 / 2;
                }

                else if ((largeurDessinArbre > 1024) &&(hauteurDessinArbre < 768)) {
                        this.setPreferredSize(new Dimension(largeurDessinArbre,768));
                        milieuX = largeurDessinArbre / 2;

                }

                else if ((largeurDessinArbre < 1024) &&(hauteurDessinArbre > 768)) {
                        this.setPreferredSize(new Dimension(1024,hauteurDessinArbre));
                        milieuX = 1024 / 2;
                }

                else  {
                        this.setPreferredSize(new Dimension(largeurDessinArbre,hauteurDessinArbre));
                        milieuX = largeurDessinArbre / 2;

                }
                this.revalidate();      //miseà jour des scrollbars
                int debY = (Actions.listeFormules.size()+2)*30;//édbut de l'arbre de preuve
                this.dessineListeFormulesInitiales(g,milieuX);
                this.dessinerPreuve(g,AP,milieuX,debY,0,0);

        }
}//fin paintComponent


public void init(){
        this.arbrePreuve    = new ArbrePreuve();
        Actions.init();
        repaint();
}



public void dessinerPreuve(Graphics g, ArbrePreuve AP, int x, int y, int xa, int ya) {

        String chaineNumero ;   // édestin a contenir le énumro de la èrgle éapplique
        int longNumero=0;       // longeur du énumro
        String racine;          // édestin a contenir la racine de l'arbre
        int longRacine=0;       // longeur de la racine
        int decaX =this.dimBranche;   //édcalage du point X
        int decaY = 70;     //édcalage du point Y
        int decaS = 15;     //édcalage Y pour afficher la racine par rapport au point x,y

        if (! (AP.estVide())){

                Noeud n = AP.getRacine();
                FontMetrics fm = g.getFontMetrics();
                if (n!=null) {
                        Formule F=n.getFormule();
                        if (F != null) {

                        racine =   F.toString();
                        longRacine =  fm.stringWidth(racine);
                        if (F.isLitteral()) {
                                        g.drawString(racine,x-longRacine/2,y+decaS);
                                }
                                else {
                                        chaineNumero  =  String.valueOf(F.getNumero())+"␣";
                                        longNumero  = fm.stringWidth(chaineNumero);


                                        g.drawString(chaineNumero+racine,x-(longRacine+longNumero)/2,y+decaS);

                                }
                        }
                }
                //Si ce n'est pas le premier noeud
                if ((xa !=0) && (ya !=0) ){
                        //affichage des liens reliant deux noeuds
                        g.drawLine(x,ya,xa,ya);
                        g.drawLine(x,ya,x,y);
                        //Affichage de la éclosit
                        if (AP.getRacine().getNoeudClos() == true){
                        //on trace deux traits èparallles en dessous du noeud courant
                                int decaClosX = 10;
                                int decaClosY=decaS+5;
                                g.drawLine( x-decaClosX, y+decaClosY,    x+decaClosX, y+decaClosY);
                                g.drawLine( x-decaClosX, y+decaClosY+5,  x+decaClosX, y+decaClosY+5);
                        }
                }
                //La branche droite est vide, il resteà dessiner le sous arbre gauche
                int intY = decaS + 10; //édcalage produit par y + un espace de éclart
                if ((AP.getSad() == null)  && (AP.getSag() != null)){
                        //Affichage de la èrgle éapplique
                        g.setColor(Color.red);
                        String espaceRegle = "␣␣␣";
                        int longRegle = fm.stringWidth(espaceRegle);
                        g.drawString(AP.getRacine().getRegle(),x+(longNumero+longRacine)/2+longRegle,y+decaS);
                        g.setColor(Color.black);
                        //Relance
                        dessinerPreuve(g, AP.getSag(), x, y+decaY, x, y+intY);
                }

                //cas éégnral, il resteà dessiner les deux sous-arbres
                else if ((AP.getSad() != null)  && (AP.getSag() != null)){
                        this.dimBranche = this.dimBranche / 2;
                        int temp= this.dimBranche;
                        g.setColor(Color.red);
                        g.drawString(AP.getRacine().getRegle(),x-10,y+decaY-25);
```

```
                                g.setColor(Color.black);
                                dessinerPreuve(g,AP.getSag(), x-decaX, y+decaY, x, y+intY);
                                this.dimBranche = temp;
                                dessinerPreuve(g,AP.getSad(), x+decaX, y+decaY, x, y+intY);
                            }
                        }//fin (!(AP.estVide())){
                }//fin dessinerArbreBis

} //fin classe ZoneDessin
```

Fcihier ZoneFormules.java

```
import javax.swing.*;
import java.awt.*;

class ZoneFormules extends JTextArea{


        public ZoneFormules(){
                super(2,10);
                this.setEditable(false);
                this.init();
        }//fin ZoneFormules


        public void init() {
                this.setText("La liste des formules est vide.");
        }


        public void ajoutFormule(String formule){
                String texte= this.getText() + formule;
                this.setText(texte);
        }



        public void afficher(ListeFormules list){
                for (int i=0;i<list.size();i++){
                        String numero = String.valueOf(list.at(i).getNumero());
                        String stringFormule = numero + ". " + list.at(i).toString();
                        if (i!=0)
                                this.ajoutFormule("\n"+stringFormule);
                        else
                                this.setText(stringFormule);
                }

        }

} //fin classe ZoneFormules
```

Fichier Fenetre.java

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;


class Fenetre extends JFrame{

    final static int HTAILLE = 900;
    final static int VTAILLE = 700;
    private ZoneDessin arbreDes ;                                   //Zone de dessin pour l'arbre
    private ZoneFormules zoneFormules;                              //Zone comportant la liste de formules
    private BarreEtat barreEtat = new BarreEtat();                   //Barre d etats
    private BarreMenus barreMenus;                                   //Barre de menus
    private BarreOutils barreOutils;                                 //Barre d'outils
    private JScrollPane scrollZoneDessin;                            //Barre de édfilement pour la zone de Dessin
    private JScrollPane scrollZoneFormules ;                         //Barre de édfilement pour la zone d e formules
    private Container contenu;

    public Fenetre(){

        contenu = this.getContentPane();                             //conteneur
        this.setTitle("Tableaux␣éSmantiques");                       //titre de la êfentre
        this.setSize(HTAILLE,VTAILLE);                               //dimension de la êfentre
        this.setIconImage(Toolkit.getDefaultToolkit().getImage("./images/logoFac.gif"));
        arbreDes = new ZoneDessin(barreEtat,this);                   //écration de la zone de dession d el'arbre
        zoneFormules = new ZoneFormules();
        scrollZoneDessin = new JScrollPane(arbreDes,ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,ScrollPaneConstants.HORIZONTAL_S
        scrollZoneDessin.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEtchedBorder() ,"Arbre␣de␣Preuve" ) );
        barreMenus = new BarreMenus(arbreDes,this);                  //écration de la barre de menus
        this.setJMenuBar(barreMenus);                                //Insertion du menu
        contenu.setBackground(Color.white);
        contenu.setLayout( new BorderLayout(2,2));                   //barre d etat en bas de page
        contenu.add(scrollZoneDessin,"Center");                                 //zone pour le dessin de l'arbre
        JPanel haut = new JPanel();
        haut.setLayout(new  BorderLayout());
        barreOutils = new BarreOutils(arbreDes,this);
        haut.add(barreOutils, "North");                              //barre d outils
        scrollZoneFormules = new JScrollPane(zoneFormules);
        scrollZoneFormules.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEtchedBorder() ,"Liste␣de␣formules␣initiale

        haut.add(scrollZoneFormules,"South");
        contenu.add(haut,"North");
        contenu.add(barreEtat,"South");

        //gestion de la Fermeture de la êfentre
        addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent e) {
            System.exit(0);
                        }});
        //center la êfentre
        this.centrerFenetre();

    } //fin constructeur Fenetre


    public ZoneFormules getZoneFormules(){
            return zoneFormules;
    }
    public JScrollPane getScrollZoneFormules(){
            return scrollZoneFormules;
    }
    public BarreEtat getBarreEtat(){
            return barreEtat;
    }

    public JScrollPane getScrollZoneDessin(){
            return scrollZoneDessin;
    }



    //Centrer les êfentresà  lé'cran
    public void centrerFenetre() {
            Dimension dimension = getToolkit().getScreenSize();
            Rectangle abounds = this.getBounds();
            setLocation((dimension.width - abounds.width) / 2,(dimension.height - abounds.height) / 2);
   }//fin émthode centerFenetre


   public void initComposants(int etat){

            if (etat == 0){
                    Actions.init();
                    arbreDes.init();
                    zoneFormules.init();
                    barreEtat.init();
                    barreMenus.init(0);
                    barreOutils.init(0);
            }
            else{
                    barreMenus.init(1);
                    barreOutils.init(1);
  }
}//fin initComposants


}//fin classe Fenetre
```

Fichier Actions.java

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.io.*;


public class Actions implements ActionListener {

        //éDfinition des variables de commandes

        static final int OUVRIR      = 1;
        static final int FERMER      = 2;
        static final int PREUVE1     = 3;
        static final int PREUVE2     = 4;
        static final int IMPRIMER    = 5;
        static final int QUITTER     = 6;
        static final int AIDE        = 7;
        static final int APROPOS     = 8;
        static ListeFormules listeFormules = new ListeFormules();
        private Fenetre fen;
        private  ZoneDessin zone;
        private int commande;


        //Constructeur
        public Actions(int commande, ZoneDessin zone,Fenetre fen){
                this.fen=fen;
                this.commande = commande;
                this.zone = zone;
        } //fin constructeur



        //éDfinition des actionsà  effectuer
        public void actionPerformed(ActionEvent e){
                switch(commande){
                        case OUVRIR :
                                this.ouvrir(fen,zone);
                                break;
                        case FERMER :
                                this.fermer(fen,zone);
                                break;
                        case PREUVE1 :
                                this.preuve1(fen,zone);
                                break;
                        case PREUVE2 :
                                this.preuve2(fen,zone);
                                break;
                        case QUITTER :
                                System.exit(0);
                                break;
                        case AIDE :
                                this.aide(fen);
                                break;
                        case APROPOS :
                                this.apropos(zone);
                                break;
                }//fin switch
        } //fin actionPerformed


         public void lireFichier(String nomFichier) throws IOException{

                //Reinitialisation du nombre de formules
                ListeFormules.nbFormules = 0;
                Parser yyparser;
                Actions.listeFormules = new ListeFormules();
                // lecture d'un fichier
                yyparser = new Parser(new FileReader(nomFichier),zone);
                yyparser.yyparse();
        }



        //permet le chargement d'un fichier deja éenregistr
        public  void ouvrir(Fenetre fen,ZoneDessin zone){
                try {
                        JFileChooser files = new JFileChooser(".");
                        files.setDialogTitle("Ouverture du fichier de formule");
                        int resultat = files.showOpenDialog(fen);
                        if (resultat == JFileChooser.APPROVE_OPTION){
                                this.initComposants(fen,zone);
                                String nomFichier = files.getSelectedFile().getName();
                                this.lireFichier(nomFichier);
                                fen.getZoneFormules().afficher(Actions.listeFormules);
                                fen.getBarreEtat().afficher("Le fichier échargé est " +nomFichier);
                                fen.initComposants(1);

                        }//fin if  (resultat == JFileChooser.APPROVE_OPTION)

                }//fin try
                catch(Exception ex){
                        JOptionPane.showMessageDialog(
                                        zone.getParent(),
                                        "Le contenu du fichier de preuve n'est pas correct !",
                                        "Erreur !",
                                        JOptionPane.ERROR_MESSAGE);
                }
        }//fin ouvrir
```

```
        public  void fermer(Fenetre fen, ZoneDessin zone){
                this.initComposants(fen,zone);
        }


        public static void init(){
                Actions.listeFormules = new ListeFormules();
        }


        public  void initComposants(Fenetre fen, ZoneDessin zone){
                fen.initComposants(0);
        }



        public void  preuve1(Fenetre fen,ZoneDessin zone){


                ArbrePreuve AP = new ArbrePreuve();
                ListeFormules LF = Actions.listeFormules.copie();
                ListeFormules LI = Actions.listeFormules.copie();
                ListeFormules.nbFormules = LF.size();
                AP.construireArbrePreuve1(LF,LI);
                if ((AP.getSag() != null) && (AP.getSad() == null)) {
                        Noeud temp = AP.getRacine();
                        AP.setRacine(AP.getSag().getRacine());
                        AP.setSad(AP.getSag().getSad());
                        AP.setSag(AP.getSag().getSag());
                        //L'arbre courant maintenant
                        AP.getRacine().setRegle(temp.getRegle());
            }
            zone.setArbrePreuve(AP);
            zone.repaint();
    }


    public void  preuve2(Fenetre fen,ZoneDessin zone){

            ArbrePreuve AP = new ArbrePreuve();
            ListeFormules LF = Actions.listeFormules.copie();
            ListeFormules LI = Actions.listeFormules.copie();
            ListeFormules LL = LF.calculeListLitteraux();
            ListeFormules.nbFormules = LF.size();
            AP.construireArbrePreuve2(LF,LI,LL);
            if ((AP.getSag() != null) && (AP.getSad() == null)) {
                    Noeud temp = AP.getRacine();
                    AP.setRacine(AP.getSag().getRacine());
                    AP.setSad(AP.getSag().getSad());
                    AP.setSag(AP.getSag().getSag());
                    //L'arbre courant maintenant
                    AP.getRacine().setRegle(temp.getRegle());
            }
            zone.setArbrePreuve(AP);
            zone.repaint();
    }


    public void aide(Fenetre fen){
        Aide a = new Aide(fen,"Aide utilisateur");
    }


    public void apropos(ZoneDessin zone){

            JOptionPane.showMessageDialog(
                            zone.getParent(),
                            "éDmonstrateur de éèthorme par la émthode des tableaux ésmantiques.\n Auteurs : Mathieu Alamichel & éGrgory
                            "A propos de TabSemantique.",
                            JOptionPane.INFORMATION_MESSAGE);

    }


}//fin classe Actions
```

Fichier Aide.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;


class Aide extends JDialog implements ActionListener{


        public Aide(Fenetre parent, String titre){

                super(parent,titre);
                //S'il y a une êfentre parent, fixer la position de la îbote d edialogue
à                // l'éintrieur
                if (parent != null){
                        Dimension parentSize = parent.getSize(); //Taille de la êfentre parent
                        Point p = parent.getLocation();          //Position de la êfentre parent
                        setLocation(p.x + parentSize.width/4,p.y + parentSize.height/4);
                }

                //Volet de message
                JPanel messagePane = new JPanel();
                JLabel message1 = new JLabel("1.␣Ouvrir␣un␣fichier␣de␣preuve");
                JLabel message2 = new JLabel("2.␣éêSlctionner␣l'une␣des␣deux␣éstratgies␣pour␣en␣faire␣la␣preuve.");

                messagePane.add(message1);
                messagePane.add(message2);
                getContentPane().add(messagePane);

                //Bouton OK
                JPanel boutonPanel = new JPanel();
                JButton bOk = new JButton("OK");
                boutonPanel.add(bOk);
                bOk.addActionListener(this);
                getContentPane().add(boutonPanel,BorderLayout.SOUTH);
                setBounds(200,200,500,150);
                setDefaultCloseOperation(DISPOSE_ON_CLOSE);
                setVisible(true);
        }

        public void actionPerformed(ActionEvent e){
                setVisible(false);
                dispose();
        }


}
```

Fichier makefile

```
.SUFFIXES:  .class .java

USER = mathieu

FAC = oui

ifdef FAC
        HOMEDIR = /home/etud/info/maitinfo/$(USER)/
        JAVA = java
        JAVAC = javac
else
        HOMEDIR = /home/$(USER)/
        #JAVA = $(JAVADIR)java
        #JAVAC =  $(JAVADIR)javac
endif

DOCDIR = ./documentation/
JFLEXDIR = $(HOMEDIR)JAVA/JFlex/bin/
BYACCDIR = $(HOMEDIR)JAVA/byacc/
JAVADIR = $(HOMEDIR)JAVA/sdk/bin/

JFLEX  = $(JFLEXDIR)jflex
BYACCJ = $(BYACCDIR)yacc.linux -J

JAVADOC_FLAGS= -author -d $(DOCDIR) -private
JAVADOC= javadoc

JAVAFLAGS =
JAVACFLAGS =

main_java =  Actions.java TabSemantique.java
formules_java = Symbole.java TableSymboles.java Formule.java  ListeFormules.java Noeud.java ArbrePreuve.java
interface_java = BarreEtat.java BarreMenus.java BarreOutils.java ZoneDessin.java ZoneFormules.java Fenetre.java Aide.java
parseur_java = Parser.java ParserVal.java Yylex.java

SRC = $(formules_java)

class_JAVA = $(parseur_class) $(formules_class) $(interface_class) $(main_class)

main_class =  Actions.class TabSemantique.class
formules_class = Symbole.class TableSymboles.class Formule.class ListeFormules.class Noeud.class ArbrePreuve.class
interface_class = BarreEtat.class BarreMenus.class BarreOutils.class ZoneDessin.class ZoneFormules.class Fenetre.class Aide.class
parseur_class = Parser.class ParserVal.class Yylex.class


all: $(class_JAVA)

%.class: %.java
                $(JAVAC) $(JAVACFLAGS) $*.java

Parser.class: Yylex.java Parser.java
        $(JAVAC) Parser.java

Yylex.java: Parseur.flex
        $(JFLEX) Parseur.flex

Parser.java: Parseur.y
        $(BYACCJ) Parseur.y

# efface tous les fichiers generes
clean:
        rm $(parseur_java) *.class
        rm *~

# genere la documentation
doc:    $(SRC)
        $(JAVADOC) $(JAVADOC_FLAGS) $(SRC)

# lance le programme TabSemantique
run: TabSemantique.class
        $(JAVA) TabSemantique
```