Les démonstrateurs automatiques de théorèmes[1]

Claude Bolduc

24 août 2001

[1]Cette recherche a pu être menée à terme grâce à une bourse CRSNG du 1er cycle.

Table des matières

Table des figures

Résumé

Les logiciels se développent de plus en plus rapidement. Pour pouvoir suivre le courant, il faut que les compagnies produisent leurs logiciels en un temps toujours plus restreint. On peut réduire le temps de production en démontrant que les spécifications du logiciel sont étanches. Ces démonstrations sont souvent fastidieuses. En utilisant un démonstrateur automatique de théorèmes, les preuves deviennent plus aisées et cela permet donc de réduire le temps de vérification des spécifications.

Ce document vise à montrer les différents démonstrateurs automatiques de théorèmes existants présentement. Aussi, il est présenté ici, un manuel d'utilisation pour le démonstrateur automatique de théorèmes Z/EVES version 2.1. Ce document présente la logique équationnelle de Gries & Schneider [5] comme pilier pour la démonstration de théorèmes de la logique propositionnelle. Cette logique a été implantée dans Z/EVES et son utilisation est aussi expliquée dans le manuel d'utilisation de Z/EVES. Tout ceci dans le but de favoriser un apprentissage humain de la démonstration de théorèmes à l'aide d'un démonstrateur automatique de théorèmes.

Chapitre 1

Introduction

1.1 Motivation

L'informatique prend de plus en plus de place dans nos vies. Un ordinateur fonctionne grâce aux différentes logiques implantées en son sein. Une des logiques primordiales pour le bon fonctionnement d'un ordinateur est la logique dite *classique*. Cette logique a pour but de « recréer le principe de raisonnement des mathématiques où l'ambiguïté et l'imprécision ne sont pas une bonne option » [7]. La logique classique en subdivisée en plusieurs parties et celle qui est la plus élémentaire est la logique propositionelle.

Pour concevoir de bons logiciels, ceux-ci doivent pouvoir se décrire presque toujours par la logique classique. Donc, pour améliorer la recherche de bogues dans les logiciels, de même qu'améliorer le temps de production des logiciels, nous devons développer des outils pour vérifier ceux-ci. Un des meilleurs outils pour cette vérification est un démonstrateur automatique de théorèmes. Un démonstrateur automatique de théorèmes est un logiciel qui peut traiter une logique quelconque (dans le cadre de cette recherche, nous ne traiterons que de la logique classique de premier ordre).

Les démonstrateurs automatiques de théorèmes qui traitent de la logique classique de premier ordre ont d'autres utilités. Parmi celles-ci on retrouve : l'aide à la recherche en mathématiques et en informatique et l'aide à l'apprentissage de la logique pour des humains. Cette recherche vise surtout

l'apprentissage de la démonstration de théorèmes à l'aide d'un démonstrateur automatique de théorèmes.

1.2 Présentation du projet

Ce rapport vise à étudier et à comprendre les démonstrateurs automatiques de théorèmes en général avec un grand accent sur les démonstrateurs qui traitent de la logique classique de premier ordre. Les démonstrateurs fonctionnant avec une autre logique ne seront que mentionnés. Quelques-uns de ceux présentés seront approfondis et un d'entre eux sera présenté en profondeur : Z/EVES Version 2.1. C'est le démonstrateur automatique de théorèmes que nous préconiserons dans ce rapport conformément aux critères de comparaison. Celui-ci sera présenté par un guide d'utilisation qui permettra de mieux le comprendre et de l'utiliser dans le cadre d'un apprentissage pour la démonstration rigoureuse de théorèmes.

Cette recherche se base sur la logique équationnelle de Gries & Schneider [5]. Notre but est de réussir à implanter cette logique dans le démonstrateur automatique de théorèmes Z/EVES Version 2.1[1] pour permettre de faire des démonstrations de théorèmes pour cette logique.

[1] Z/EVES a été créé par ORA Canada lors du projet TR-97-5505-04g débuté en janvier 1996 [15].

Première partie

Quatre-vingt différents démonstrateurs automatiques de théorèmes

Chapitre 2

La logique propositionnelle

2.1 Introduction à la logique propositionnelle

Il y a plusieurs logiques différentes. Une de celles-ci est la logique classique. « Elle a été créée pour rédiger les principes de raisonnement des mathématiques où l'ambiguïté et l'imprécision sont une mauvaise chose » [7]. Le but de cette logique est de créer un monde idéal : un modèle mathématique. Cette logique est composée d'autres logiques dont l'une est la logique propositionnelle. La logique présentée ici est basée sur la logique équationnelle de Gries & Schneider [5].

La logique propositionnelle est une logique partisane de la doctrine disant que tout est blanc ou tout est noir[1].

La logique propositionnelle permet de représenter d'une façon simplifiée et non ambigüe des réalités (surtout des affirmations). Prenons comme exemple la phrase suivante : « je suis étudiant ». Cette phrase peut être représentée par une variable booléenne qui peut être soit vraie, soit fausse. Appelons cette variable p. La variable p ne pourra être vraie que si, en ce moment, je suis réellement un étudiant. Autrement, p est fausse.

Cette logique permet aussi de faire des arrangements complexes en mettant en relation des variables booléennes. Les opérateurs de coordination per-

[1] « Une proposition est un énoncé qui peut être vrai ou faux » [4].

mettant de réaliser des arrangements complexes sont définis dans la section 2.2.

Cette logique est basée sur des axiomes de base (pour la liste exhaustive de ceux utilisés dans cette recherche, voir [5]). « Un axiome est une expression dont on assume qu'elle est un théorème sans en donner la démonstration » [4].

Ces axiomes permettent de dériver des théorèmes. « Un théorème [est] soit un axiome, soit la conclusion d'une règle dont les prémisses sont des théorèmes ou soit une expression dont on démontre [...] qu'elle est égale à un axiome ou à un théorème précédemment démontré. » [4].

La logique équationnelle de Gries & Schneider est basée seulement sur des axiomes qui sont des tautologies[2].

Chaque théorème de la logique propositionnelle peut être démontré à l'aide des axiomes de cette logique.

Il y a plusieurs façons de réaliser une preuve d'un théorème. Il y a donc plusieurs manières d'arriver au résultat. Une façon de procéder peut faire en sorte que l'on arrive très rapidement et de façon efficace au résultat alors qu'une autre méthode pourrait rallonger de beaucoup la preuve et même ne pas y parvenir. Les humains sont capables de produire des preuves très courtes en usant d'intelligence et d'expérience (souvent même d'astuce).

Les démonstrateurs automatiques de théorèmes essaient d'imiter (d'une façon limitée) le comportement humain lors de la démonstration de théorèmes. Ces outils ont commencé par utiliser des heuristiques[3] simples de démonstration de théorèmes. Ensuite, ils se sont raffinés pour devenir plus performants et conviviaux. La presque totalité des démonstrateurs automatiques de théorèmes performants ne sont pas totalement *automatiques*. En fait, pour augmenter leur puissance, on les rend *semi-automatiques* pour qu'un humain puisse diriger le démonstrateur à sa guise. Cela occasionne aussi le fait que l'humain doit souvent aider le démonstrateur.

En fait, un démonstrateur automatique de théorèmes est un peu comme

[2] « Proposition complexe qui reste vraie en vertu de sa forme seule, quelle que soit la valeur de vérité des propositions qui la composent » [1].

[3] « Méthode de recherche empirique ayant recours aux essais et erreurs pour la résolution de problèmes » [3].

un apprenti cuisinier. Vous, vous êtes les maîtres cuisiniers. Vous donnez à votre élève des ingrédients dont il doit se servir pour cuisiner des plats spécifiques que vous désirez. Ces ingrédients sont les axiomes de base que votre démonstrateur connaît. Ensuite, vous lui donnez un gâteau déjà préparé et décoré. Il est déjà tout prêt, mais votre élève n'a rien vu de sa préparation. Vous dites à votre apprenti de faire un gâteau totalement identique à celui que vous lui présentez. Ce gâteau est la même chose qu'un théorème. Votre apprenti utilisera ses connaissances et ses ingrédients pour essayer de faire ce magnifique gâteau seul, mais il se peut qu'il en soit incapable. À ce moment-là, vous devez lui donner des indices comme utiliser tel ou tel ingrédient avec telle quantité ou simplement lui dire de ne pas se servir de tel ingrédient. De la même manière, vous pouvez dire à votre démonstrateur de tenter de démontrer un théorème et, s'il n'y arrive pas seul, vous pouvez lui donner des commandes spécifiques pour le diriger afin qu'il réussisse cette tâche.

Dans cette recherche, nous donnerons, pour débuter, à notre « apprenti cuisinier » seulement des axiomes de la logique équationnelle de Gries & Schneider pour vérifier comment il se débrouille avec ces ingrédients.

En bref, la logique propositionnelle est une représentation simplifiée de la réalité qui nous permet de mieux isoler un problème pour déterminer sa véracité.

2.2 La syntaxe de la logique propositionnelle

La logique propositionnelle possède sa propre notation. D'ailleurs celle-ci varie quelque peu d'un ouvrage à un autre. La syntaxe utilisée dans cette recherche est celle préconisée par Gries & Schneider [5]. Cette notation se compose des conventions de base données dans la figure 2.1.

Cette notation permet de représenter des phrases françaises. Reprenons l'exemple de la section 2.1. La variable p représente le fait « je suis étudiant ». Faisons de même pour les phrases « je travaille » et « je gagne de l'argent » et appelons les respectivement q et r. À ce moment-là, on peut relier ces phrases pour faire des raisonnements. En utilisant la syntaxe présentée dans la figure 2.1, on peut donner les axiomes suivants :

Opérateurs de coordination	Équivalent en français	Utilisation	Nom de l'opérateur
$\neg$	Non	$\neg p$	Négation
$\wedge$	Et	$p \wedge q$	Conjonction
$\vee$	Ou	$p \vee q$	Disjonction
$\Rightarrow$	Implique	$p \Rightarrow q$	Implication
$\Leftarrow$	Conséquence	$p \Leftarrow q$	Conséquence
$\not\equiv$	Inéquivalent	$p \not\equiv q$	Inéquivalence
$\equiv$	Équivalent	$p \equiv q$	Équivalence
$=$	Égal	$p = q$	Égalité

FIG. 2.1 – Tableau de la syntaxe de la logique propositionnelle

- $q \equiv r$
- $p \wedge \neg q \Rightarrow \neg r$

Le premier axiome dit que je travaille si et seulement si je gagne de l'argent et le deuxième affirme que si je suis étudiant et que je ne travaille pas alors je ne gagne pas d'argent.

Grâce à cela, on peut maintenant représenter de façon simplifiée une réalité. Ceci permet de représenter les propositions plus facilement en donnant un certain degré d'abstraction et permet aussi de mathématiser la logique propositionnelle.

Ce rapport décrit comment cette logique a été implantée dans le logiciel Z/EVES créé par ORA Canada.

Chapitre 3

Les différents démonstrateurs automatiques de théorèmes

3.1 Les critères de comparaison

Quatre-vingt démonstrateurs automatiques de théorèmes ont été étudiés dans cette recherche. Pour chacun de ces démonstrateurs, on a vérifié des critères permettant d'orienter une décision pour obtenir le démonstrateur automatique de théorèmes le plus adapté aux besoins que nous avions. Ces démonstrateurs ont été comparés suivant ces critères[1] :

- Les *plateformes* sur lesquelles le démonstrateur automatique de théorèmes fonctionne. Nous recherchons un démonstrateur fonctionnant sur la plateforme adoptée par la majorité des étudiants en informatique à l'université Laval : Windows.
- La *qualité de la bibliographie* offerte pour supporter l'apprentissage et la connaissance du logiciel. Nous recherchons simplement une bibliographie ayant une bonne quantité d'information pertinente pour mieux comprendre et apprivoiser le démonstrateur automatique de théorèmes.
- Le *coût de l'implantation* du démonstrateur automatique de théorèmes. De préférence, nous utiliserons un démonstrateur ayant un faible coût

[1] Les critères sont classés en ordre décroissant de priorité.

d'implantation pour des étudiants.

- La *puissance maximale* que le démonstrateur peut atteindre (les différentes logiques qu'il supporte). Nous nous limitons à la logique classique de premier ordre dans le cadre de cette recherche.
- Le *langage de programmation* sur lequel est construit le démonstrateur automatique de théorèmes. Nous préconisons un démonstrateur écrit dans un langage d'actualité et simple d'apprentissage.
- La *convivialité* du démonstrateur automatique de théorèmes. Nous désirons un démonstrateur ayant une convivialité acceptable pour être capable de réaliser des démonstrations devant public.
- Le *type d'interaction* du logiciel. Une interaction flexible est la bienvenue.
- La *rapidité* du démonstrateur lors de la réalisation d'un problème complexe et d'un problème simple. Il est difficile d'obtenir ce renseignement et c'est le critère le moins important pour cette présente recherche. Tant que le logiciel peut réaliser des preuves dans un temps raisonnable, il est convenable à ce moment-là.

Ces critères ont été déterminés avant d'entreprendre l'analyse des différents démonstrateurs automatiques de théorèmes.

3.2 Les démonstrateurs vérifiés

La présente recherche est basée sur la liste intitulée *overview of systems implementing "mathematics in the computer"* de Freek Wiedijk[2]. Cette liste contient deux cent dix-neuf outils différents. Parmis ces outils, quatre-vingt ont été vérifiés dans cette recherche. Nous nous limitions dans la liste aux catégories d'outils nommées : *theorem prover*, *first-order prover* et *tactic prover*[3].

[2]Cette liste est disponible sur internet à l'adresse `http://www.cs.kun.nl/~freek/digimath/index.html` [21].

[3]Les catégories qui n'ont pas été prises en compte dans cette recherche sont : *logic education*, *computer algebra*, *proof checker*, *specification environment*, *information*, *representation* et *authoring*.

Pour mieux visualiser les différences entre les différents démonstrateurs automatiques de théorèmes, les données ont été regroupées sous forme de tableaux : les tableaux A.1 à A.25 dans l'annexe A.

L'annexe A.1 fournit le tableau contenant les informations sur les différences entre les démonstrateurs sur les critères des *plateformes* que supportent le démonstrateur automatique de théorèmes, du *langage* dans lequel est écrit le démonstrateur et de la *puissance maximale* de cet outil.

L'annexe A.2 montre les différences entre les démonstrateurs pour la *qualité de la bibliographie* et le *type d'interaction*.

L'annexe A.3 présente le *coût de l'implantation* pour chacun des démonstrateurs automatiques de théorèmes.

L'annexe A.4 donne les différences entre les démonstrateurs automatiques de théorèmes pour les critères suivants : la *rapidité pour les démonstrations simples et complexes*, la *convivialité* et le *niveau d'intérêt* soulevé par chacun d'eux pour cette recherche.

Finalement, l'annexe A.5 fournit *l'adresse Internet* utilisée pour retrouver les informations concernant chacun de ces démonstrateurs[4].

[4]Seule l'adresse URL est donnée.

Chapitre 4

Les démonstrateurs automatiques de théorèmes qui ont été approfondis

Grâce aux critères de comparaison énoncés à la section 3.1, nous avons pu procéder à l'évaluation des démonstrateurs automatiques de théorèmes. Cette évaluation a permis d'éliminer plusieurs démonstrateurs et de cibler rapidement les plus intéressants parmi la liste. Ceci a permis de sélectionner cinq de ces démonstrateurs pour les approfondir. Ces démonstrateurs automatiques de théorèmes sont[1] :

- ACL2
- HOL98
- HOL Light
- Otter
- Z/EVES

[1]Pour le restant de ce chapitre, le terme Unix désigne autant cette plateforme que celles fonctionnant sur le même principe que ce système d'exploitation. Par exemple, ce terme réfère aussi à Linux, Solaris, DEC Unix, IRIX, SCO Unix, ...

4.1 ACL2

Le démonstrateur automatique de théorèmes nommé ACL2 [6] est un logiciel très intéressant programmé en Common Lisp. Ses principaux avantages et désavantages sont réunis ici :

Avantages :

1. Il est assez polyvalent (il tourne sur des machines ayant comme système d'exploitation : Unix, Windows 98 et Mac OS).
2. La documentation concernant ce logiciel est très complète (environ 900 pages).
3. Il ne nécessite qu'un compilateur de Lisp pour fonctionner.
4. Le logiciel est gratuit.
5. Ce logiciel traite de la théorie des ensembles finis.

Désavantages :

1. Il est difficile à télécharger pour une personne désirant une copie du logiciel adaptée à une machine ne fonctionnant pas sur la plateforme Unix car il n'y a pas d'instructions sur l'installation sur une plateforme autre que Unix.
2. Il prend 19.5 Mo de mémoire.
3. Il est simplement un démonstrateur de théorèmes semi-automatique (il fonctionne seulement par dialogue).
4. Il demande beaucoup de temps d'apprentissage pour bien savoir s'en servir (quelques mois pour un bon informaticien), d'après son créateur.
5. La documentation concernant la théorie des ensembles finis est dispersée à l'intérieur du code.

4.2 HOL98

Le démonstrateur automatique de théorèmes HOL98 [20] est un logiciel écrit en Moscow ML démontrant des théorèmes de la logique classique.

Avantages de HOL98 :

1. Il est adapté pour les plateformes Unix et Windows NT.
2. Sa bibliographie est très complète et intéressante (il y a trois manuels sur son utilisation qui se complètent : le tutoriel [10], les références [9] et la description [8] de HOL).
3. La notation de la logique dans ce logiciel est proche de la notation utilisée par Gries & Schneider.
4. Le style de preuve est relativement facile à suivre.
5. Ce logiciel est gratuit.
6. Ce logiciel traite de la théorie des ensembles.

Désavantages de HOL98 :

1. L'installation de Moscow ML est assez complexe (celle de HOL98 aussi si on utilise une autre plateforme que Windows NT).
2. L'utilisation de HOL98 n'est pas très intinctive pour un nouvel utilisateur.
3. Son mode d'interaction est le dialogue.
4. C'est un démonstrateur de théorèmes semi-automatique (il faut aider HOL à trouver les preuves).
5. La puissance maximale de ce démonstrateur automatique de théorèmes est trop élevée pour les besoins de la cause.
6. Il n'a pas encore été testé sur les plateformes Windows 95/98 .

4.3 HOL Light

HOL Light [11] est dans la même lignée que HOL98 (ils font partie de la même famille de démonstrateurs automatiques de théorèmes. De même que son prédécesseur, HOL Light est capable de démontrer des théorèmes de la logique classique. Il est programmé en CAML Light.

Avantages de HOL Light :

1. Il donne plusieurs outils pour la résolution automatique et il offre des théorèmes mathématiques déjà prouvés contrairement à ses prédécesseurs.

2. Son manuel d'instruction est concis et clair (il contient une centaine de pages [12]).
3. Ce démonstrateur automatique de théorèmes est facilement adaptable.
4. Il est gratuit.
5. Il peut manipuler la théorie des ensembles.
6. Il est surtout utilisé en mathématique (c'est son point fort).

Désavantages de HOL Light :

1. Il fonctionne avec un mode d'interaction de type dialogue.
2. Ce logiciel ne fonctionne que sur une plateforme Unix.

4.4 Otter

Le démonstrateur automatique de théorèmes Otter [13] est un logiciel écrit en C et ayant un système de démonstration basé sur la résolution[2].

Avantages de Otter :

1. Il est gratuit.
2. Il est facilement modifiable.
3. Il possède un mode autonome.
4. Il peut recevoir des instructions en notation infixée, suffixée ou postfixée.
5. Il fonctionne sur presque toutes les plateformes.
6. Sa documentation est intéressante et complète [14].

Désavantages de Otter :

1. Il donne souvent des preuves par contradiction.
2. Il a de la difficulté à trouver des preuves par induction.
3. Sa notation laisse à désirer.
4. Sa documentation n'est pas simple.

[2]La résolution est une méthode inventée par J.A. Robinson vers 1960 [7].

4.5 Z/EVES

Le démonstrateur automatique de théorèmes nommé Z/EVES est un logiciel basé sur un système automatique de déduction appelé NEVER. Il conjoint la force du démonstrateur nommé EVES [2] avec le langage de spécification Z.

Avantages de Z/EVES :

1. Il tourne sur les plateformes Unix et Windows.
2. Il traite seulement de la logique classique (notre objectif est de traiter seulement de la logique classique).
3. Sa documentation est très bien organisée et elle est volumineuse (voir [15], [16], [17] et [18]).
4. Son interface est très intéressante.
5. Ce logiciel est gratuit.
6. Il traite de la théorie des ensembles.
7. Ses preuves avec la théorie des ensembles ont un bon degré d'automatisme.

Désavantages de Z/EVES :

1. Il faut avoir une licence pour pouvoir obtenir une version du logiciel.
2. Il fonctionne avec une interaction de type dialogue.
3. Le style de preuve est très peu explicite sur les transformations effectuées.

Deuxième partie

Z/EVES

Chapitre 5

Démonstrateur automatique de théorèmes choisi : Z/EVES Version 2.1

5.1 Raisons qui motivent le choix de ce démonstrateur

Après une analyse approfondie des cinq démonstrateurs automatiques de théorèmes, deux choix ont été retenus : HOL98 et Z/EVES. Ces deux logiciels ont été testés d'une façon rigoureuse pour déterminer le meilleur choix entre ces deux démonstrateurs tout aussi intéressants l'un que l'autre.

L'apprentissage de ces deux logiciels a été réalisé et on a tenté d'implanter, dans chacun de ces démonstrateurs, la logique de Gries & Schneider. Cette expérience a permis de faire ressortir les points suivants.

5.1.1 HOL

Avantages :

- Il possède une librairie booléenne qu'on peut utiliser pour réaliser une logique qui se rapproche de celle de Gries & Schneider.

- On peut modifier assez facilement les opérateurs déjà présents et leur priorité (et même en définir d'autres.)

Désavantages :

- Il est difficile d'apprentissage.
- Il ne possède pas l'équivalence comme opérateur.
- L'interface est purement textuelle.

5.1.2 Z/EVES

Avantages :

- L'interface est graphique (GUI[1]) ou textuelle.
- Z/EVES est relativement facile d'apprentissage (la grammaire est facilement accessible).
- Z/EVES possède l'équivalence comme opérateur (sur des prédicats ou des schémas).
- On peut créer d'autres opérateurs.

Désavantages :

- Il ne possède pas de type booléen (car le Z standard n'a pas la notion de booléen).
- Il est très difficile (impossible dans certain cas) de redéfinir les opérateurs prédéfinis.

En considérant ces points, notre choix s'est porté sur Z/EVES, car il était plus naturel, flexible et convivial que HOL98. Z/EVES permet de réaliser et de visualiser les preuves plus facilement que HOL98. De plus, on ne peut retracer le cheminement suivi par HOL98 pour démontrer un théorème ce qui rendait son utilisation très peu intéressante pour l'apprentissage de la démonstration de théorèmes. Le logiciel créé par ORA Canada permet de suivre les étapes de la démonstration des théorèmes d'une façon plus naturelle sans qu'on ait l'impression que Z/EVES sort un lapin d'un chapeau. Il était beaucoup plus intéressant pour permettre un bon apprentissage de la démonstration de théorèmes à de nouveaux utilisateurs.

Finalement, Z/EVES était le choix logique pour nos besoins.

[1] *Graphic User Interface*

5.2 Guide d'utilisation de Z/EVES Version 2.1

5.2.1 Introduction

Z/EVES est un logiciel créé par ORA Canada. C'est un logiciel puissant qui conjoint la force du démonstrateur automatique de théorèmes EVES avec la syntaxe du langage de spécification Z. La syntaxe utilisée du Z est celle du Z standard tel que défini par J.M. Spivey (voir [19] pour plus de détails concernant cette notation). EVES est un démonstrateur automatique de théorèmes développé aussi par ORA Canada.

Avant de débuter La syntaxe des quantificateurs universels et existentiels utilisée dans cette recherche diffère quelque peu de la syntaxe préconisée par Gries & Schneider [5]. Cette syntaxe est celle utilisée par Z/EVES. Voilà la syntaxe pour ces deux quantificateurs[2] :

$$\star \text{ liste\_variables\_liées} : \text{type} \bullet \text{prédicat}$$

Cette syntaxe diffère de celle de Gries & Schneider surtout pour l'utilisation des parenthèses. En fait, Z/EVES n'utilise pas de parenthèses pour définir la limite de la portée des quantificateurs ; sa portée est définie par le plus grand prédicat possible après le symbole « • ». Le symbole « • » est équivalent au symbole « | : » dans la notation de Gries & Schneider.

5.2.2 Obtenir et installer Z/EVES

Le logiciel Z/EVES Version 2.1 est disponible sur le site d'ORA à l'adresse URL : `http://www.ora.on.ca/z-eves/welcome.html`. Ce logiciel est disponible au public pour une utilisation soit pédagogique, soit de recherche. Z/EVES n'a pas encore de licence pour une utilisation commerciale. Il faut remplir une demande pour obtenir une licence du logiciel. Cette demande doit être postée à ORA Canada et, ensuite, ORA Canada donnera (ou non) son autorisation. À ce moment-là, vous recevrez des instructions disant comment aller télécharger le logiciel sur leur serveur.

[2]Ici le symbole $\star$ désigne soit le quantificateur universel ($\forall$), soit le quantificateur existentiel ($\exists$).

Une fois que vous avez obtenu la licence et que vous êtes allés chercher votre copie de Z/EVES, il faut l'installer. Sur la plateforme Windows, l'installation n'est pas très complexe. Vous décompressez le fichier spécifié du logiciel pour Windows et ensuite vous n'aurez qu'à lancer l'application concernant l'installation (le fichier *setup.exe*). Ensuite, vous suivez les instructions d'installation.

Ceci permet de faire fonctionner Z/EVES d'une façon partielle. En effet, avec cette installation, il se peut que vous n'avez pas accès à toutes les fonctionnalités du logiciel. Z/EVES fonctionne sur deux modes : le mode textuel et le mode graphique. Le mode textuel fonctionne très bien après la procédure d'installation donnée plus haut. Le mode graphique utilise une interface graphique écrite en Python 1.5.2. Pour faire fonctionner cette interface, vous devez avoir deux logiciels supplémentaires disponibles gratuitement sur le web : Python 1.5.2 et Tcl/Tk 8.0.

Python 1.5.2 est disponible à l'adresse URL : `http://www.pythonlabs.com/download/`. C'est la page officielle de la société qui a créé Python.

Tcl/Tk 8.0 est disponible à l'adresse URL : `http://www.scriptics.com/software/tcltk/8.0.tml`. C'est la page des développeurs de Tcl.

L'installation de ces deux logiciels se fait facilement en suivant les instructions du fichier *readme*.

Maintenant, l'interface graphique de Z/EVES devrait fonctionner.

5.2.3 Débuter le démonstrateur

Comme expliqué précédemment, Z/EVES peut fonctionner en deux modes indépendants, mais ces deux modes ne peuvent coexister en même temps. Chacune des sections qui suivent sera donc séparée en deux sous-sections, décrivant respectivement le mode textuel et le mode graphique.

Mode textuel

Pour débuter Z/EVES en mode textuel, il suffit, dans la plateforme Windows, de cliquer sur « Démarrer → Programmes → Z-EVES 2.1 → Z-EVES ».

Normalement, l'interface du mode textuel devrait apparaître.

Mode graphique

Pour lancer Z/EVES en mode graphique, il faut, dans le système d'exploitation Windows, cliquer sur « Démarrer → Programmes → Z-EVES 2.1 → Z-EVES GUI ».

À ce moment, si l'installation s'est bien déroulée, l'interface graphique devrait être visible.

5.2.4 Explication des interfaces

Il y a deux interfaces dans Z/EVES. Chacune de ces interfaces est différente et les outils pour faire le lien entre ces deux interfaces ne sont pas totalement implantés dans ce démonstrateur. ORA Canada continue d'ailleurs d'améliorer le lien entre ces interfaces. Bientôt les deux communiqueront sans problèmes, mais la version 2.1 fait ressortir que la construction du pont n'est pas terminée.

Mode textuel

Ce mode possède une interface qui permet d'entrer dans le mode Z/LaTeX. Cette interface est simple d'utilisation. La syntaxe pour entrer des spécifications en Z est donnée à la section 5.2.5. Cette section se propose de donner un aperçu des options possibles de cette interface[3].

Une fois que vous avez débuté le démonstrateur automatique de théorèmes dans le mode textuel (comme présenté à la section 5.2.3), vous vous retrouvez devant une fenêtre comme celle présentée à la figure 5.1.

Cette interface est simple. Il y a peu de commandes accessibles par le menu. Il faut entrer au clavier le reste des spécifications que vous désirez.

[3]Cette section du mode textuel est basée en grande partie sur le document [15].

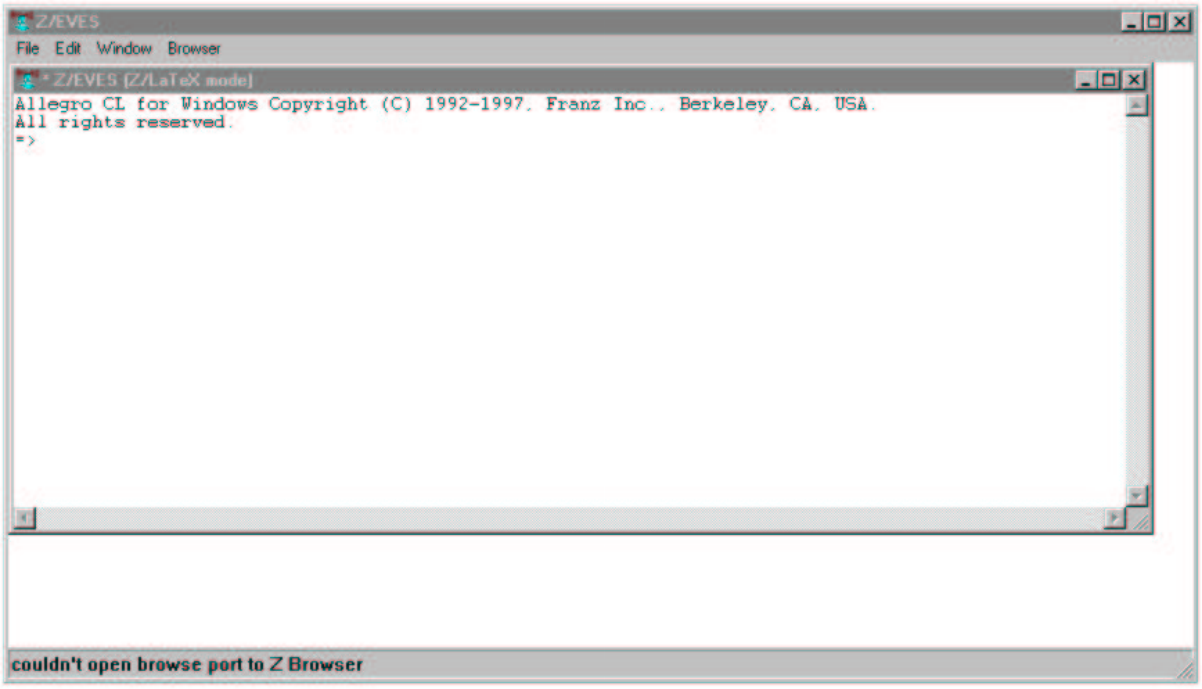

FIG. 5.1 – Interface Z/LaTeX de Z/EVES sous la plateforme Windows.

La barre de menu se compose de quatre différents menus : `File`, `Edit`, `Window` et `Browser`. Ces quatre menus permettent l'accès à des commandes de base. Pour bien comprendre et assimiler ces commandes de base, lisez la section 5.1 du manuel d'instruction [15].

La commande importante dans le menu est celle pour lire des fichiers en mode *batch*. Elle est importante et intéressante car elle permet de lire un fichier déjà écrit en Z/LaTeX pour visualiser ses résultats. Donc, on peut s'en servir pour faire tourner des exemples déjà programmés. Cette commande sera utile pour le chapitre 6.

Une dernière chose est importante lorsque vous utilisez l'interface textuelle. C'est l'utilisation de la touche `Break` sur le clavier. Elle est importante car elle permet de gagner du temps lorsque vous savez que Z/EVES travaille sur une preuve et qu'il semble être perdu dans un trop grand nombre de possibilités. Appuyez sur cette touche, celle-ci vous permettra d'arrêter la tentative en cours. Entrez des commandes plus précises et ensuite relancez Z/EVES. Ceci vous fera économiser du temps.

Mode graphique

L'utilisation de l'interface en mode graphique est naturelle. Cette interface est conviviale. Pour bien comprendre les différentes fenêtres et leurs commandes respectives, lisez la section 2.2 du manuel [18].

La commande pertinente pour cette recherche est une de celles qui se retrouvent dans la fenêtre *Specification* dans le menu *File* : la commande *Import*. Cette commande puissante permet d'importer des fichiers Z/LaTeX dans l'interface graphique. Dans cette recherche, elle est utile pour charger la banque de théorèmes et d'axiomes de la logique de Gries & Schneider pour pouvoir travailler avec. Le seul défaut de cette commande pour l'instant est de ne pas importer les preuves dans l'interface graphique. Donc, on a la banque de théorèmes, mais on n'a pas les démonstrations de ces théorèmes (il faut les refaire soi-même).

5.2.5 Syntaxe du démonstrateur

Z/EVES fonctionne sous deux formes de syntaxe : celle du mode textuel et celle du mode graphique. Ces formes sont presque identiques.

Mode textuel

La syntaxe du logiciel en mode textuel est très proche de celle de LaTeX.

Syntaxe de base du Z standard Z/EVES connaît la syntaxe de base du Z standard, mais cette syntaxe est écrite en LaTeX modifié. Le Z standard a une grammaire de base. Plusieurs *lexèmes* de cette grammaire ne sont pas utilisés dans cette recherche. La grammaire que l'on fournit ci-dessous ne comprend pas les commandes destinées à diriger le démonstrateur automatique de théorèmes. Les *lexèmes* qui ont été omis dans cette recherche et qui font partie du Z standard sont :

- Les *schema boxes* : C'est à l'intérieur de ces boîtes que résident les définitions des schémas en Z.
- Les *z sections* : Ce sont les sections où on peut placer les paragraphes du Z pour en faire des blocs qu'on peut référencer.
- Les *axiomatic boxes* : C'est à l'intérieur de ces boîtes qu'on peut donner des axiomes, conformément au Z standard, qui seront pris en compte durant toute la spécification.
- Les *generic boxes* : Ce sont presque des *axiomatic boxes* sauf que ces boîtes possèdent une liste d'arguments formels génériques.
- Les *schema definitions* : C'est une autre manière que celle des *schema boxes* pour définir le contenu d'un schéma.
- Les *abbreviation definitions* : C'est une façon rapide d'assigner à une variable, une expression.
- Les *labelled predicates* : Ce sont des prédicats qu'on peut référencer grâce à leur étiquette. De plus, on peut donner l'état d'un prédicat (dire si ce prédicat est actif ou non durant la vérification de la spécification).

La grammaire suivante est celle utilisée dans cette présente recherche[4] :

spécification	::=	objet_de_spéc. . . objet_de_spéc
objet_de_spéc	::=	décl_syntaxe
	|	paragraphe_Z
décl_syntaxe	::=	`\syndef`{phrase_ou_mot}{classe_op}{phrase_ou_mot}
phrase_ou_mot	::=	phrase | mot
classe_op	::=	`infun1` | `infun2` | `infun3` | `infun4` | `infun5` | `infun6`
	|	`ingen` | `pregen`
	|	`inrel` | `prerel`
	|	`postfun`
	|	`word`
	|	`ignore`
paragraphe_Z	::=	boîte_zed
	|	théorème
boîte_zed	::=	`\begin`[para_opt]{`zed`}
		objet_boîte_zed sep . . . sep objet_boîte_zed
		`\end`{`zed`}
	|	`\begin`[para_opt]{`syntax`}
		objet_boîte_zed sep . . . sep objet_boîte_zed
		`\end`{`syntax`}
para_opt	::=	[état]
état	::=	`enabled` | `disabled`
objet_boîte_zed	::=	définition_ensemble_donné
	|	définition_type_libre
définition_ensemble_donné	::=	[liste_identifiant]
définition_type_libre	::=	identifiant : := branche | . . . | branche
sep	::=	\\ | `;` | `\also`

[4]Pour en savoir plus sur la grammaire complète du logiciel d'ORA Canada, veillez consulter [16].

branche	::=	identifiant
	|	nom_variable \ldata expression \rdata
liste_identifiant	::=	identifiant , ... , identifiant
identifiant	::=	mot décoration
nom_variable	::=	identifiant | (nom_op)
nom_op	::=	_ sym_in décoration _
	|	sym_pre décoration _
	|	_ sym_post décoration
	|	_ \limg _ \rimg
	|	- décoration
décoration	::=	[marque... marque]
marque	::=	' | ! | ?
	|	_0 | _1 | _2 | _3 | _4 | _5 | _6 | _7 | _8 | _9
sym_in	::=	fct_in | gén_in | rel_in
sym_pre	::=	gén_pre | rel_pre
sym_post	::=	fct_post
théorème	::=	\begin[para_opt]{theorem}{[usage] nom_théorème}[par_formel_gén]
		prédicat
		[\proof
		commande sep ... sep commande]
		\end{theorem}
usage	::=	axiom | rule | grule | frule
nom_théorème	::=	mot
par_formel_gén	::=	[liste_identifiant]
prédicat	::=	prédicat_1
prédicat_1	::=	expression rel ... rel expression
	|	prédicat_1 \implies prédicat_1
	|	prédicat_1 \iff prédicat_1
	|	true
	|	false
	|	(prédicat)

expression	::=	expression_1
expression_1	::=	expression_2
expression_2	::=	expression_3
expression_3	::=	expression_4
expression_4	::=	[Locale_ou_Globale] nom_var [par_eff_gén]
Locale_ou_globale	::=	\Local | \Global
par_eff_gén	::=	[liste_expression]
liste_expression	::=	expression, ..., expression
rel	::=	= | \in

Syntaxe ajoutée à celle de base par la présente recherche Notre but lors de cette recherche était d'implanter la logique de Gries & Schneider dans un démonstrateur automatique de théorèmes avec son système d'axiomes et ses opérateurs.

Nous avons donc créé une algèbre booléenne conçue sur les axiomes et les opérateurs fournis par Gries & Schneider qui permettait de réaliser des démonstrations de théorèmes de cette logique équationnelle. De plus, elle favorisait un apprentissage plus naturel de la démonstration de théorèmes.

Pour réussir ceci, nous avons implanté à l'intérieur de Z/EVES un nouveau type : le type `BoolGries`. Ce nouveau type est en fait un ensemble que nous avons donné à Z/EVES en ne lui disant pas comment il était défini. Nous lui avons donné deux constantes (les constantes `vrai` et `faux`) et nous lui avons spécifié qu'elles étaient élément de cet ensemble. Ceci avait comme rôle de créer un type booléen.

Ensuite, nous avons implanté les opérateurs de base de Gries & Schneider. Ces opérateurs sont tous des fonctions. Leur définition de type est définie ci-dessous :

Syntaxe de l'opérateur	Symbole de l'opérateur	Type de l'opérateur
\equiv	$\equiv$	BoolGries $\times$ BoolGries $\rightarrow$ BoolGries
\nonequiv	$\not\equiv$	BoolGries $\times$ BoolGries $\rightarrow$ BoolGries
\implique	$\Rightarrow$	BoolGries $\times$ BoolGries $\rightarrow$ BoolGries
\consequence	$\Leftarrow$	BoolGries $\times$ BoolGries $\rightarrow$ BoolGries
\et	$\wedge$	BoolGries $\times$ BoolGries $\rightarrow$ BoolGries
\ou	$\vee$	BoolGries $\times$ BoolGries $\rightarrow$ BoolGries
\non	$\neg$	BoolGries $\rightarrow$ BoolGries

Nous avons donné à Z/EVES seulement la classe des opérateurs, leur priorité et leur nombre d'argument. Ensuite, cela nous permettait d'utiliser ces opérateurs de la façon définie par le tableau ci-dessous[5] :

Syntaxe de l'opérateur	Symbole de l'opérateur	Exemple d'utilisation	Priorité
\equiv	$\equiv$	p \equiv q	3
\nonequiv	$\not\equiv$	p \nonequiv q	3
\implique	$\Rightarrow$	p \implique q	4
\consequence	$\Leftarrow$	p \consequence q	4
\et	$\wedge$	p \et q	5
\ou	$\vee$	p \ou q	5
\non	$\neg$	\non p	9

Maintenant, nous avons les opérateurs de la logique propositionnelle de Gries & Schneider que nous pouvons utiliser, mais nous n'avons ni leur définition, ni leur comportement. L'approche que nous avons utilisée consistait à décrire le comportement des opérateurs grâce aux axiomes de base. Cela nous permettait de ne pas être obligé de définir chaque opérateur comme étant des combinaisons des opérateurs de base de Z/EVES. Nous avons donc étendu les opérateurs de Z/EVES à la logique équationnelle de Gries & Schneider.

Ces opérateurs permettent, en conjonction avec des variables de type BoolGries, de créer des expressions dans le langage de Z/EVES[6]. À ce

[5]La syntaxe est donnée en ordre croissant de priorité.

[6]Remarquez bien qu'ici le mot *expression* est employé et non le mot *prédicat*.

moment-là, on peut créer des prédicats avec des expressions et, donc, donner des théorèmes de la logique équationnelle de Gries & Schneider grâce à cette notation.

Les axiomes que nous avons implantés sont donnés à l'annexe B.

Rendu à ce stade-ci de la recherche, nous avons donc un nouveau type que nous avons appelé `BoolGries`, nous avons de nouveaux opérateurs, leur priorité, leur nombre d'arguments et nous avons même spécifié leur comportement. Nous sommes donc prêts à soumettre des théorèmes de cette algèbre booléenne à Z/EVES et à voir sa réaction. Nous avons réalisé ceci et nous nous sommes rendus compte que Z/EVES se prêtait très bien à la démonstration des théorèmes de la logique équationnelle de Gries & Schneider. Les preuves sont naturelles et intéressantes. Parfois Z/EVES utilisait même des chemins auxquels nous ne pensions pas à prime abord. Sauf qu'il faut savoir bien diriger ce démonstrateur pour obtenir de bons résultats rapidement. Les théorèmes démontrés par Z/EVES sont donnés dans l'annexe C ainsi que leur démonstration[7]. Avant de vous y référer, nous vous invitons d'abord à vous familiariser avec la syntaxe de Z/EVES en mode graphique (donnée à la section 5.2.5) et les différentes commandes utilisées dans les démonstrations (voir section 5.2.6).

Mode graphique

Le mode graphique est beaucoup plus simple d'utilisation que le mode textuel. En mode graphique, pour entrer un texte, si la fenêtre *specification* est affichée, il faut choisir dans le menu :

$$\textit{Edit} \rightarrow \textit{New paragraph.}$$

Ensuite, la fenêtre du *Mini-editor* apparaît. À ce moment-là, vous pouvez entrer votre spécification Z (dans notre cas, soit un théorème ou un axiome) dans la fenêtre de texte.

Pour réaliser les caractères spéciaux du Z, il y a déjà des boutons de commandes prévus à cet effet ainsi que leur macro respective. Pour utiliser

[7]Les théorèmes et les axiomes des annexes B et C sont donnés de la même manière qu'ils sont représentés dans le mode graphique de Z/EVES.

les opérateurs ajoutés dans Z/EVES par cette recherche[8], il suffit d'utiliser la syntaxe définie ci-dessous :

Syntaxe de l'opérateur	Symbole de l'opérateur	Exemple d'utilisation	Priorité
equiv	$\equiv$	p equiv q	3
nonequiv	$\not\equiv$	p nonequiv q	3
implique	$\Rightarrow$	p implique q	4
consequence	$\Leftarrow$	p consequence q	4
et	$\wedge$	p et q	5
ou	$\vee$	p ou q	5
non	$\neg$	non p	9

Les caractères spéciaux (les symboles) sont accessibles dans le mode graphique grâce aux boutons de commandes et leur macro respective, mais les commandes LaTeX, entrées au clavier comme si vous étiez dans le mode textuel, ne fonctionnent pas. Il faut absolument utiliser leur homologue graphique.

Les mêmes axiomes et théorèmes que ceux définis dans le mode textuel fonctionnent et peuvent être utilisés dans le mode graphique.

5.2.6 Commandes du démonstrateur

Les commandes permettant de diriger le démonstrateur fonctionnent tout aussi bien dans le mode textuel que graphique[9], sauf qu'il y a des commandes qui ne fonctionnent que dans le mode textuel[10]. Pour entrer des commandes de façon manuelle dans le mode graphique, il faut sélectionner dans le menu de la fenêtre *Proof* :

Edit $\rightarrow$ *New Command.*

Ceci vous amènera dans le *mini-editor* qui vous permettra d'entrer vos commandes.

[8] Ces opérateurs sont les mêmes que dans le mode textuel, mais la syntaxe est différente.

[9] La description des commandes s'inspire grandement de celle réalisée dans [16].

[10] Ces commandes seront marquées de ce signe : $\star\star\star$.

Z/EVES fonctionne selon la méthode du but fixé. Cette méthode utilise le théorème qu'on demande de résoudre par le démonstrateur et celui-ci devient le but principal. En entrant des commandes, on dirige Z/EVES pour qu'il puisse, à l'aide de ses axiomes de base et de ses règles de réécriture, tenter de réduire le plus possible ce but à un but *équivalent* qui est appelé le but courant. Une démonstration est considérée terminée lorsque le but courant est le prédicat *true*. Donc, si le but courant n'est pas *true*, on essaie de le réduire pour l'obtenir.

5.2.6.1 Commandes pour séparer un but en sous-buts

Quelquefois, les preuves sont ardues et on a de la difficulté à cerner le problème dans son entier. À ce moment-là, des commandes peuvent servir pour séparer un but courant en plusieurs sous-buts dont leur ensemble est équivalent. Ces commandes sont :

Cases La commande `cases` possède cette syntaxe :

commande ::= `cases`

Cette commande sépare le but en plusieurs sous-buts si le but est :

- une conjonction
- une disjonction
- un prédicat conditionnel
- une implication dont le conséquent a une des formes mentionnées plus haut

Le but est donc séparé en sous-buts et Z/EVES donne visuellement un des sous-buts pour qu'on puisse continuer de le réduire. Les autres sous-buts sont accessibles seulement avec la commande `next`.

Next La commande `next` est définie par la syntaxe suivante :

commande ::= `next`

Cette commande ne fonctionne qu'en conjonction avec la commande `cases`. Elle permet de changer de sous-but après avoir réalisé un `cases`. Il n'est pas nécessaire d'avoir terminé la preuve du sous-but courant pour changer de sous-but.

Split La syntaxe de la commande `split` est :

commande ::= `split` prédicat

La commande `split` P (où P est un prédicat) permet de transformer un but G en un nouveau but. Le but G devient `if` P `then` G `else` G. *Attention :* le prédicat P doit respecter le type du but G. Autrement dit, on ne peut faire apparaître de nouvelles variables dans P.

Conjunctive Cette commande est définie par la syntaxe suivante :

commande ::= `conjunctive`

Cette commande ne fait qu'effectuer les transformations suivantes sur le but courant :

1. Toutes les implications, les équivalences et les prédicats conditionnels sont remplacés (d'une façon équivalente) par les opérateurs de conjonction et de disjonction. *Attention :* ces transformations ne sont effectuées que sur les opérateurs cités plus haut faisant partie de la grammaire initiale de Z/EVES et non sur les opérateurs que l'on a implantés par la suite.
2. La portée de l'opérateur de négation de Z/EVES est minimisée. Par exemple, $\neg(\forall X \bullet P)$ devient, après transformation, $(\exists X \bullet \neg P)$.
3. Z/EVES effectue la distributivité de la disjonction sur la conjonction.

Toutes ces transformations permettent de modifier le but pour obtenir un prédicat composé seulement de conjonctions, de disjonctions et de négations et ayant, de plus, la conjonction comme opérateur ayant la plus basse préséance parmi ceux qui se retrouvent dans l'expression. Autrement dit, cette commande transforme l'expression dans la forme normale conjonctive[11].

Disjunctive La syntaxe de cette commande est :

commande ::= `disjunctive`

Le but de cette commande est presque identique à celui de la commande `conjunctive`, sauf que l'opérateur ayant la plus basse préséance parmi ceux qui se retrouvent dans l'expression doit être la disjonction et non la conjonction. Autrement dit, cette commande transforme l'expression dans la forme

[11] « La forme normale conjonctive est une forme du calcul des propositions où la formule est une conjonction de littéraux disjonctifs (ou de « clauses ») » [3].

normale disjonctive. Les transformations que la commande effectue sont les mêmes que celle de la commande `conjunctive`, sauf pour la dernière transformation : Z/EVES applique la distributivité de la conjonction sur la disjonction à la place de la distributivité de la disjonction sur la conjonction.

5.2.6.2 Utilisation des théorèmes et des définitions

Il y a deux manières d'utiliser les divers théorèmes et définitions dans Z/EVES : la manière manuelle et celle automatique.

5.2.6.2.1 La manière manuelle

Il y a trois commandes qu'on peut utiliser avec la façon manuelle pour jouer avec les théorèmes et les définitions. Ces commandes sont définies ci-dessous :

Apply Cette commande possède une syntaxe propre :

commande ::= `apply` nom_théorème
| `apply` nom_théorème `to expression` expression
| `apply` nom_théorème `to predicate` prédicat

Le théorème *nom_théorème* doit être celui que l'utilisateur souhaite appliquer au but. Ce théorème doit être une règle de réécriture (définie par `rule` dans la grammaire de Z/EVES). Une règle de réécriture est un prédicat contenant deux prédicats ou deux expressions et ceux-ci sont séparés par un signe d'égalité (=). Lorsque Z/EVES rencontre l'expression ou le prédicat de la partie à gauche de l'égalité, alors le démonstrateur sait que le côté droit de l'égalité est équivalent au côté gauche. Donc, Z/EVES peut réécrire un prédicat à chaque fois qu'il rencontre l'expression ou le prédicat de la partie gauche de l'égalité, en substituant cette partie par la partie droite de l'égalité. Par exemple, si le but courant est ce prédicat :

$$\forall p, q : \texttt{BoolGries} \bullet p \equiv q = \ (p \land q) \lor (\neg p \land \neg q)$$

et qu'on possède la règle de réécriture *etCommutativite* définie comme suit :

$$\forall p, q : \texttt{BoolGries} \bullet p \land q = q \land p,$$

alors notre but devient, après l'exécution de la commande `apply` *etCommutativite* `to expression` $p \land q$:

$$\forall p, q : \texttt{BoolGries} \bullet p \equiv q = (q \wedge p) \vee (\neg p \wedge \neg q).$$

En utilisant la commande `apply` *nom_théorème* seulement, alors Z/EVES tentera d'appliquer la règle de réécriture partout où cela est possible dans le but. Si une expression ou un prédicat particulier est spécifié (une expression est spécifiée par `to expression` et un prédicat par `to predicate`), alors le démonstrateur appliquera la règle, si possible, seulement aux occurrences de l'expression ou du prédicat désiré.

Invoke Cette commande a la syntaxe suivante :

commande	::=	`invoke` [nom_événement]
	|	`invoke predicate` prédicat
nom_événement	::=	nom_schéma | nom_définition

La commande `invoke` permet de remplacer, dans le but courant, toutes les occurrences du nom d'un schéma ou du nom d'une définition spécifiée par leur définition respective. Si aucun nom n'a été spécifié dans la commande, alors tous les noms de schémas ou de définitions seront remplacés par leur définition.

La commande `invoke predicate` est semblable à la commande `invoke`, sauf que l'on spécifie un nom de prédicat dans la commande `invoke predicate`. La grande différence se situe dans la façon dont Z/EVES interprète les commandes. Par exemple, si on dirige le démonstrateur en donnant la commande : `invoke` S (où S est un schéma) et qu'au même moment le but courant contient des références à S et S' (S' est le schéma S décoré[12]), on a comme résultat que chacune des occurrences de S sont remplacées par la définition de S et que les occurrences de S' sont remplacées par la définition de S, mais où chacune des variables de cette définition sont maintenant décorées. Pour éviter ce problème, on utilise la commande `invoke predicate` S qui permet de transformer seulement les occurrences de S dans le but courant.

Use La syntaxe de la commande est donnée comme suit :

[12]D'après le Z standard [19], un schéma sans décoration représente l'état initial du schéma tandis qu'un schéma décoré du signe « $'$ » (prime) représente l'état final du schéma.

commande	::=	`use` référence_théorème
référence_théorème	::=	nom_théo décoration [par_eff_gén] [remplacements]
nom_théo	::=	mot
remplacements	::=	[renommer_ou_subst, ..., renommer_ou_subst]
renommer_ou_subst	::=	nom_décl `/` nom_décl
	|	nom_décl `:=` expression
nom_décl	::=	identifiant | nom_op

La commande `use` sert à prendre un théorème déjà existant et à le rajouter dans les hypothèses du but courant. Cela sert surtout si le théorème que l'on souhaite ajouter est un théorème qui n'a pas été défini avec un usage (les usages sont définis dans Z/EVES comme étant : `axiom`, `rule`, `frule` ou `grule`), mais cette commande s'applique aussi très bien aux théorèmes définis avec un usage. Le nom du théorème doit être écrit correctement (*attention :* Z/EVES est sensible à la casse pour les noms de théorèmes) avec sa décoration (s'il y a lieu). Si le théorème comporte des paramètres formels génériques, alors on doit indiquer les paramètres effectifs génériques que l'on désire utiliser dans le théorème.

Prenons comme exemple le théorème *inCup*[13], un théorème déjà présent dans Z/EVES à son démarrage. Celui-ci possède un paramètre formel générique appelé X. Si on désire ajouter ce théorème aux hypothèses de notre but courant, alors il faut spécifier l'ensemble X. Supposons que nous restons dans la logique implantée dans Z/EVES par cette présente recherche. À ce moment-là, nous pouvons utiliser comme paramètre effectif générique l'ensemble `BoolGries` défini plus haut. On appellerait donc ce théorème grâce à la commande suivante :

```
use inCup[BoolGries] ;
```

Pour chaque variable d'un théorème, on peut renommer ou substituer. On renomme en utilisant la syntaxe suivante :

expression_voulue/variable_libre_du_théorème.

La substitution se réalise suivant cette syntaxe :

variable_libre_du_théorème `:=` expression_voulue.

On ne peut utiliser un théorème avec la commande `use` que si *toutes* ses variables (après avoir renommé et effectué les substitutions exigées dans le

[13]Ce théorème est la définition de l'union dans Z/EVES.

théorème) sont des noms de variables que l'on retrouve dans le but courant. Par exemple, si on a comme but courant :

$$\forall p, q : \texttt{BoolGries} \bullet p \vee (p \vee q) = p \vee q$$

et qu'on désire utiliser le théorème de l'associativité de la disjonction alors il faut utiliser la commande :

$$\texttt{use ouAssociativite2[}q \texttt{ := } p\texttt{, } r \texttt{ := } q\texttt{] ;.}$$

Cette commande ajoute dans les hypothèses :

$$p \in \texttt{BoolGries} \wedge p \in \texttt{BoolGries} \wedge q \in \texttt{BoolGries} \Rightarrow p \vee (p \vee q) = p \vee p \vee q.$$

On doit absolument utiliser la commande `use` pour se servir du théorème *ouAssociativite2* car il a été défini sans qu'on lui donne d'usage.

Autre chose intéressante que la commande `use` réalise : avant de faire la substitution ou de renommer une variable, Z/EVES prend chaque quantificateur universel et l'enlève pour rendre chacune des variables du théorème libre. Pour combler la lacune causée par l'enlèvement de ces quantificateurs, il ajoute aux hypothèses du théorème que chacune des variables qui étaient liées au départ est maintenant élément de l'ensemble (le type) qu'on lui avait spécifié.

5.2.6.2.2 La manière automatique

Les commandes automatiques pour la démonstration de théorèmes dans Z/EVES sont au nombre de trois au même titre que celles de la manière manuelle. Ces trois opérations sont des commandes de réduction que le démonstrateur effectue pour réduire un but à une forme équivalente qu'il considère plus simple pour lui. *Attention :* les commandes automatiques n'utilisent que les théorèmes qui ont un état actif (`enabled`). Un théorème est considéré actif par Z/EVES dès que celui-ci est vérifié et accepté syntaxiquement. Les seuls théorèmes qui peuvent être inactifs (`disabled`) sont ceux qui sont donnés par des *labelled predicates* qui ne sont pas traités dans cette présente recherche[14]. Les trois commandes automatiques sont définies ci-dessous :

[14]Voir 5.2.5 pour une explication brève des *labelled predicates*.

Simplify La syntaxe de cette première commande automatique est :

```
commande ::= simplify
           | trivial simplify
```

Lors d'une demande de simplification (avec la commande `simplify`), Z/EVES réalise les opérations suivantes : il fait du raisonnement au sujet de l'égalité, au sujet des entiers et au sujet des propositions, il vérifie si le but est une tautologie et il applique des règles de transfert (l'usage `frule`) et des règles d'hypothèse (les théorèmes d'usage `grule` et les hypothèses du but courant) où cela est possible dans le but. Cette commande est très utile en conjonction avec la commande `use` car elle permet d'utiliser les règles d'hypothèse pour les appliquer au but sans faire appel aux règles de réécriture (les théorèmes d'usage `rule`), règles qui souvent changent le but courant ce qui fait en sorte que l'hypothèse n'est plus utile.

La commande `trivial simplify` est moins puissante, mais plus rapide que la commande `simplify`. Cette commande réalise seulement une partie d'une simplification. Elle vérifie seulement les tautologies et elle réalise un raisonnement sur les propositions.

Rewrite La commande `rewrite` possède cette syntaxe :

```
commande ::= rewrite
           | trivial rewrite
           | prove by rewrite
```

La commande `rewrite` fait de la réécriture sur le but courant. En réécriture, Z/EVES réalise la simplification (la commande `simplify` mentionnée plus haut) et applique aussi les règles de réécriture dans l'espoir de réduire le but courant. Cette commande est très utile lorsqu'on désire utiliser plusieurs règles de réécriture rapidement. Par exemple, on peut souhaiter utiliser plusieurs fois le théorème de commutativité ou d'associativité de l'équivalence. À ce moment-là, on peut utiliser cette commande et voir le résultat rapidement.

La commande `trivial rewrite` est une forme très limitée de réécriture. En fait, celle-ci ne réalise pas de simplification et applique simplement les règles de réécriture qui ne sont pas conditionnelles.

La commande `prove by rewrite` répète la commande de réduction `rewrite` jusqu'à ce qu'une réécriture n'ait pas d'effet. À chaque itération, les opérations suivantes sont réalisées :

1. Le but courant subit la commande `prenex` (cette commande est définie dans la section 5.2.6.3).
2. La commande `rearrange` est utilisée (cette commande est définie plus loin dans cette section).
3. La commande `equality substitute` est utilisée (voir section 5.2.6.4).
4. Le but courant est modifié par la commande `rewrite`.

Reduce Cette commande est appelée suivant cette syntaxe :

commande ::= `reduce`
| `prove by reduce`

En utilisant `reduce`, Z/EVES réalise les opérations suivantes : il effectue une simplification (`simplify`), une réécriture (`rewrite`) et il remplace toutes les références à des schémas ou des instances génériques (des opérateurs) par leur définition respective et, ensuite, le résultat est réduit une autre fois.

La commande `prove by reduce` répète la commande de réduction `reduce` jusqu'à ce qu'une réduction n'ait pas d'effet. À chaque itération, les opérations suivantes sont réalisées :

1. Le but courant subit la commande `prenex` (cette commande est définie dans la section 5.2.6.3).
2. La commande `rearrange` est utilisée (cette commande est définie plus loin dans cette section).
3. La commande `equality substitute` est utilisée (voir section 5.2.6.4).
4. Le but courant est modifié par la commande `reduce`.

Rearrange Une quatrième commande pourrait se rajouter à la manière automatique d'utiliser les théorèmes et les définitions, mais celle-ci ne tente pas de réduire le but courant. Sa seule fonction est de réarranger les hypothèses pour que les hypothèses apparaissent dans un ordre plus simple pour Z/EVES, ce qui permet d'augmenter l'efficacité et la rapidité des commandes de réduction. Sa syntaxe est celle-ci :

commande ::= `rearrange`

5.2.6.3 Les quantificateurs

Les quantificateurs internes de Z/EVES ont deux commandes spéciales qui sont utiles pour leur manipulation dans le démonstrateur.

Instantiate La syntaxe suivante est définie pour cette commande :

commande	::=	`instantiate` instanciations
instanciations	::=	let_déf, ..., let_déf
let_déf	::=	nom_variable `==` expression

Cette commande est utile pour instancier une ou plusieurs variables quantifiées. Toutes les variables que l'on souhaite instancier doivent apparaître dans le même prédicat quantifié. Par exemple, on ne pourrait pas instancier p et q en même temps dans ce but : $\forall p : \mathbb{Z} \mid p > 1 \bullet \forall q : \mathbb{Z} \bullet \ldots$, mais on pourrait réaliser cela si le prédicat était plutôt : $\forall p, q : \mathbb{Z} \bullet \ldots$.

Bien entendu, l'expression donnée dans la commande est vérifiée par rapport au type pour que l'expression respecte le but courant. Cette commande est utile lorsqu'on souhaite vérifier une égalité possible entre des variables différentes dans deux quantifications différentes.

Prenex La commande `prenex` possède la syntaxe suivante :

commande ::= `prenex`

Le fonctionnement de cette commande est simple. Tout d'abord, le démonstrateur tente de transformer tous les quantificateurs en quantificateurs universels. Ensuite, Z/EVES enlève les quantificateurs universels qui sont dominants (les premiers que l'on traite) et transforme le but courant en ajoutant aux hypothèses (pour le prédicat traité) que chaque variable liée devient une variable élément du type (de l'ensemble) spécifié par la quantification. Par exemple, si le but courant est :

$$\forall p : \texttt{BoolGries} \bullet p \vee \neg p = \texttt{vrai}$$

alors la commande `prenex` donne le but équivalent suivant :

$$p \in \texttt{BoolGries} \Rightarrow p \vee \neg p = \texttt{vrai}.$$

À vrai dire, cette commande crée un effet spécial dans Z/EVES. L'application de `prenex` fait en sorte que chaque variable liée devient libre par la suite. Les substitutions sont faciles à réaliser par la suite.

5.2.6.4 Utilisation des égalités

Il n'existe qu'une commande qui permet d'utiliser les égalités.

Equality substitute La commande a la syntaxe suivante :

commande ::= `equality substitute` expression

Si une expression e est spécifiée dans la commande et qu'il y a une hypothèse dans le but courant qui donne une égalité entre e et une expression e' alors toutes les occurrences de e dans le but sont remplacées par e' (l'expression voulue). Si aucune expression e n'est spécifiée alors les hypothèses contenant des égalités entre expressions (s'il y en a) sont choisies grâce à une méthode heuristique et les remplacements sont réalisés dans le but courant.

5.2.6.5 Modificateurs

Il y a quelques commandes qui permettent d'affecter le comportement d'une commande de réduction de façon temporaire. Celles-ci sont définies ci-dessous :

With disabled Ce modificateur a la syntaxe suivante :

commande	::=	`with disabled` (liste_nom_événement) commande
liste_nom_événement	::=	nom_événement, ..., nom_événement
nom_événement	::=	déclaration
	|	nom_règle_de_transfert
	|	nom_règle_de_réécriture
	|	nom_règle_d'hypothèse
	|	nom_définition

La commande `with disabled` permet de désactiver temporairement (le temps d'une commande) des théorèmes, des définitions ou des déclarations. Ceci permet d'exécuter une commande de réduction automatique (présentée à la section 5.2.6.2.2) en enlevant des événements dont on ne souhaite pas se servir. Tenter de désactiver un événement déjà désactivé n'a aucun effet.

Cette commande a été *extrêmement* utile lors de la présente recherche car Z/EVES fonctionne d'une façon séquentielle pour l'utilisation de ses règles de réécriture. Lorsqu'on désirait se servir d'un théorème qui résidait vers la fin de la liste de théorèmes ou que l'on désirait enlever une définition qui risquait d'être gênante pour le restant de la démonstration, alors on désactivait les théorèmes qu'on ne voulait pas utiliser. Ceci permet de gagner un temps considérable dans les preuves car le nombre de cas possibles diminue lorsqu'on restreint le nombre de théorèmes dont on peut se servir. Par exemple, on souhaite prouver le théorème suivant[15] :

$$\forall p, q, r : \texttt{BoolGries} \bullet p \wedge (q \wedge r) = p \wedge q \wedge (p \wedge r).$$

On peut le démontrer rapidement grâce aux théorèmes de la conjonction seulement. Donc, si on désire utiliser ces théorèmes d'une façon automatique sans utiliser la règle d'or (qui sera appliquée en premier car elle est positionnée plus avant dans la liste de théorèmes que les théorèmes de la conjonction), il faut désactiver la règle d'or pour rendre celle-ci inactive lors de l'exécution de la commande `rewrite`. On entrerait donc comme commande :

```
with disabled ( RegleDOr ) rewrite ;.
```

With enabled Cette commande a la syntaxe suivante :

commande ::= `with enabled` (liste_nom_événement) commande

Cette commande permet d'activer temporairement (le temps d'une commande du démonstrateur) des événements qui sont désactivés, pour ensuite réaliser la commande spécifiée. Tenter d'activer un événement déjà activé n'a pas d'effet.

With expression La syntaxe pour la commande `with expression` est :

commande ::= `with expression` (expression) commande

Elle permet de réaliser la commande spécifiée sur seulement toutes les occurrences de *expression* dans le but courant plutôt que partout dans le but.

With predicate La syntaxe pour la commande `with predicate` est :

commande ::= `with predicate` (prédicat) commande

[15] C'est le théorème *DistDeEtSurEt1* dans notre banque de théorème.

Elle permet de réaliser la commande spécifiée sur seulement toutes les occurrences de *prédicat* dans le but courant plutôt que partout dans le but.

With normalization La syntaxe pour cette commande est la suivante :

commande ::= `with normalization` commande

Avant d'utiliser cette commande, il faut savoir comment Z/EVES représente les propositions dans son langage. À vrai dire, Z/EVES connaît tous les opérateurs logiques sous une forme conditionnelle. Par exemple, pour Z/EVES, la proposition : $p \wedge q$ est représentée à l'interne par l'expression `if` p `then` q `else false`.

Par défaut, la forme conditionnelle est désactivée dans Z/EVES car cette forme a tendance à répéter inutilement des données (ce qui rend les expressions longues). Donc, nous ne voyons que les symboles des opérateurs et nous n'utilisons pas leur définition interne.

En utilisant cette commande, Z/EVES normalise les expressions pour les mettre dans son langage interne, réalise une réduction de la forme conditionnelle sous une forme plus intéressante et, ensuite, il exécute la commande désirée. Ceci a pour effet d'augmenter la puissance de Z/EVES, mais rend les expressions beaucoup plus longues et, voire même, incompréhensibles. Cette commande est surtout utile et intéressante lorsque le but courant est une disjonction.

5.2.6.6 Retour en arrière

Il y a deux commandes qui permettent de revenir en arrière lors d'une démonstration.

Back $\star\star\star$ La syntaxe pour cette commande est :

commande ::= `back` [nombre]

Cette commande permet de revenir en arrière de *nombre* étapes dans la preuve. Si aucun nombre n'est indiqué, alors Z/EVES revient en arrière d'une étape. Z/EVES perd le cheminement réalisé pour ces *nombre* étapes lors de l'exécution de cette commande. On doit donc les réaliser de nouveau si on désire revenir à la situation où on a exécuté cette commande.

Retry $\star\star\star$ Cette commande est définie comme suit :

commande ::= `retry`

La commande `retry` enlève et perd toutes les étapes de la preuve réalisées jusqu'à présent et met le but courant de la démonstration dans sa forme initiale.

5.2.6.7 Autre commande utile

Il existe une autre commande qui est *très* utile pour la démonstration de théorèmes dans Z/EVES et qui est définie très sommairement dans [16].

prove Cette commande possède la syntaxe suivante :

commande ::= `prove`

À vrai dire, la commande `prove` a déjà été définie dans la section 5.2.6.2.2. La commande `prove` est le diminutif de la commande `prove by rewrite`. Donc, dans l'interface graphique de Z/EVES vous ne trouverez que la commande `prove` dans les options du bouton *reduction*. Gardez toujours à l'esprit que cette commande est la même que `prove by rewrite`.

5.2.7 Formalisme des données d'entrée

Maintenant que vous connaissez les différentes fenêtres de Z/EVES dans les deux modes ainsi que la syntaxe de base de Z/EVES, et que vous avez pris connaissance de la syntaxe que la présente recherche a implantée dans Z/EVES, vous êtes prêt à comprendre comment nous avons réussi à donner à ce démonstrateur cette panoplie d'axiomes et de théorèmes.

Mode textuel

Comment entrer un axiome ?

Tout d'abord, Z/EVES hait ne pas connaître le type d'une variable utilisée dans un théorème. Il faut donc toujours déclarer le type de chaque variable

soit à l'aide d'un type prédéfini, soit à l'aide d'un type que vous lui donnez vous-même comme notre type `BoolGries`. Dans cette recherche, vous pouvez remarquer que chaque variable est déclarée de type `BoolGries`. *Attention :* n'oubliez pas que Z/EVES désire avoir comme théorème un prédicat ! Donc, chacun de nos axiomes a été mis sous forme d'un prédicat pour que le démonstrateur les accepte. Ces prédicats sont tous de la même forme : ils possèdent un quantificateur universel qui permet de déclarer chacune des variables de l'axiome.

Ensuite, il faut donner le corps de l'axiome : le prédicat. N'oubliez pas que les opérateurs que nous avons implantés ne permettent que de créer des expressions dans le langage de Z/EVES. Il y a plusieurs façons de réaliser un prédicat (voir [16] pour une liste exhaustive des diverses méthodes de création de prédicats). À la figure 5.2, vous retrouverez une partie de la grammaire de Z/EVES que nous avons modifiée pour créer nos prédicats[16].

En regardant ceci nous nous rendons compte que nous devons relier au moins deux expressions par un opérateur de relation pour créer un prédicat (celui qui a été choisi dans la présente recherche est l'opérateur $=$). Nous avons donc dû créer notre algèbre booléenne en prenant ce facteur en compte. Remarquez que chaque axiome possède un opérateur de relation.

Ce facteur a compliqué quelque peu notre recherche car il fallait, quelquefois, donner plusieurs fois un axiome, mais avec une forme différente pour permettre à Z/EVES de couvrir tous les cas possibles. Par exemple, si nous avions défini les axiomes 3.2 et 3.3[17] sans nous soucier de ce détail, nous aurions obtenu ces deux axiomes :

$$\forall p, q, r : \texttt{BoolGries} \bullet p \equiv q \equiv r = p \equiv (q \equiv r)$$
$$\forall p, q : \texttt{BoolGries} \bullet p \equiv q = q \equiv p$$

et ceux-ci auraient été insuffisants pour démontrer certains théorèmes. Voici un exemple d'un tel théorème :

[16] En tant que tel, la grammaire de Z/EVES sur les prédicats est moins restreinte que celle présentée ici, mais nous vous présentons seulement la partie pertinente pour cette recherche. Les mots en *italiques* dans la grammaire représente les modifications de la grammaire réalisées pour une meilleure compréhension de la formation des prédicats dans cette recherche.

[17] Cette numérotation des théorèmes est la même que celle utilisée dans les notes de cours du cours « Logique et Structures Discrêtes » [4].

prédicat	::=	\forall *déclaration* @ prédicat_1
	|	prédicat_1
prédicat_1	::=	expression rel ... rel expression
	|	(prédicat)
rel	::=	= | \in
expression	::=	expression_1
expression_1	::=	expression_2
expression_2	::=	expression_2 *opérateur_infixe* expression_2
	|	*opérateur_préfixe* expression_2
	|	expression_3
expression_3	::=	expression_4
expression_4	::=	[Locale_ou_Globale] nom_var [par_eff_gén]
opérateur_infixe	::=	*equiv* | *nonequiv* | *et*
	|	*ou* | *implique* | *consequence*
opérateur_préfixe	::=	*non*
déclaration	::=	décl_basique ; ... ; décl_basique
décl_basique	::=	liste_noms_déclarés : *type*
liste_noms_déclarés	::=	nom_décl, ..., nom_décl
nom_décl	::=	identifiant | nom_op
type	::=	*BoolGries* | $\mathbb{N}$ | $\mathbb{Z}$
	|	*autre_ensemble_donné*

FIG. 5.2 – Grammaire utilisée pour la formation des prédicats.

$$\forall p,\, q : \texttt{BoolGries} \bullet p \equiv p = q \equiv q.$$

Pourtant, ce simple théorème se démontre à ce stade-ci dans la progression suivie par Gries & Schneider. Donc, nous avons dû être vigilant pour ne pas oublier tous les cas possibles.

Dans le mode textuel, il était très facile de transcrire ces axiomes en suivant la syntaxe donnée à la section 5.2.5. Nous avons utilisé l'environnement fourni pour la conception de théorèmes et nous avons donné comme usage, à chacun des axiomes, `rule`. Ce qui désignait cet axiome comme étant une règle de réécriture que Z/EVES allait utiliser automatiquement pour réaliser des preuves. De plus, nous ne donnions pas de preuve des axiomes (un axiome est un théorème que l'on accepte comme étant vrai d'après notre définition de ce terme).

Voilà un exemple du code utilisé dans le mode textuel pour implanter l'axiome *nonequivDef* (l'axiome 3.13 d'après la numérotation des théorèmes du cours « Logique et Structures Discrêtes » [4]) :

```
\begin{theorem}{rule nonequivDef}
\forall p, q: BoolGries @ p \nonequiv q = \non p \equiv q
\end{theorem}
```

Comment entrer un théorème ?

Les théorèmes sont donnés de la même façon que les axiomes présentés auparavant, à la différence que l'usage d'un théorème peut varier : si un théorème est utile pour rendre plus automatique les démonstrations alors on lui décerne l'usage `rule` (règle de réécriture) sinon on ne lui donne aucun usage. Pour savoir si un théorème est utile pour rendre le démonstrateur automatique de théorèmes plus performant, on s'est fié à son nombre d'occurrences dans les preuves des théorèmes. Les théorèmes dont le nombre d'occurrences était élevé, ont obtenu un statut de règle de réécriture.

Autre différence entre les théorèmes et les axiomes : les preuves. Pour chaque théorème, nous avons trouvé une preuve possible (pas nécessairement la meilleure) dans Z/EVES. Pour trouver les preuves, nous réalisions les démonstrations dans le mode graphique et transcrivions ensuite cette preuve dans un fichier Z/LaTeX[18]. Une telle technique était utilisée parce que les

[18]Pour l'instant, il n'existe aucun outil dans ce logiciel qui permet de passer du mode graphique au mode textuel.

démonstrations sont beaucoup plus naturelles et visuellement plus faciles à comprendre dans le mode graphique. Le désavantage est que l'on ne sait pas quels théorèmes Z/EVES avait utilisés pour se rendre au but courant. Donc, on le transformait dans le mode textuel pour pouvoir connaître les théorèmes utilisés.

Les théorèmes sont entrés de la même manière que définie dans la section 5.2.5. Voilà un exemple d'un théorème (le théorème *Absorption1*[19]) avec sa preuve :

```
\begin{theorem}{Absorption1}
\forall p, q: BoolGries @ p \et (p \ou q) = p
\proof
rewrite\\
apply DistOuSurEquiv2 to expression p \ou (q \equiv p \ou q)\\
simplify\\
use ouAssociativite2[q:= p, r:= q]\\
rewrite\\
\end{theorem}
```

Vous pouvez aussi entrer un théorème (sans nom) dans l'interface Z/LaTeX en utilisant simplement la commande `try`. Cette commande possède la syntaxe suivante :

commande ::= `try` prédicat

Mode graphique

Le mode graphique est identique au mode textuel pour la théorie de l'implantation de l'algèbre booléenne. Ce qui diffère est la manière d'entrer les axiomes et les théorèmes.

Comment entrer un axiome ?

En suivant les règles énoncées dans la section du mode textuel, on a entré les axiomes en suivant la syntaxe suivante[20] :

[19] Le numéro 3.55 dans la numérotation des théorèmes utilisée dans le cours « Logique et Structures Discrêtes » [4].

[20] Les axiomes doivent être entrés sans se soucier de la police de caractères (Z/EVES ne traite qu'une écriture italique).

theorem rule *nom_théorème*
prédicat

Ce qui donne exactement le même résultat que dans le mode textuel.

Voici un exemple d'un tel axiome dans le mode graphique, l'axiome *nonequivDef* (le même axiome que donné dans le mode textuel pour que vous puissiez voir la différence) :

theorem rule *nonequivDef*
$\forall p,\ q : BoolGries \bullet p\ nonequiv\ q = non\ p\ equiv\ q$

Pour entrer cet axiome dans le mode graphique, il faut tout d'abord sélectionner la fenêtre *Specification* et il faut atteindre la fenêtre du *mini-editor*. Ensuite, vous écrivez l'axiome dans cet éditeur de texte et vous allez sélectionner dans le menu `File|Done`. Finalement, vous cliquez sur le bouton de droite de la souris sur l'axiome en question pour aller sélectionner dans le menu flottant l'option `Check`. Ceci vous dira si Z/EVES accepte l'axiome.

Comment entrer un théorème ?

Pour donner un théorème à Z/EVES dans le mode graphique, il faut tout d'abord entrer le théorème comme si on entrait un axiome[21]. Ensuite, on fait vérifier syntaxiquement le théorème par le démonstrateur qui accepte le théorème s'il correspond à la grammaire de Z/EVES.

La dernière étape consiste, pour différencier un théorème d'un axiome, à réaliser la preuve de ce théorème pour qu'il devienne effectivement un théorème pour Z/EVES. Pour donner une preuve dans le mode graphique, il faut cliquer sur le bouton droit de la souris sur votre théorème et vous sélectionnez l'option `Show proof`. Ceci vous fera accéder à la fenêtre *Proof*. Dans cet environnement, il faut entrer les commandes, une à une, pour diriger le démonstrateur automatique de théorèmes de la même manière que celle présentée dans la section 5.2.6.

Une fois que la preuve est complétée, vous revenez à la fenêtre *Specification* et cela vous permet de voir que votre théorème est véritablement devenu un théorème pour Z/EVES.

[21] Si vous ne désirez pas que le théorème soit une règle de réécriture, omettez simplement le mot *rule* dans l'écriture du théorème.

5.2.8 Formalisme des données de sortie

Cette section se propose de montrer comment lire une preuve d'un théorème réalisée par notre démonstrateur. De plus, cette section ne traite que du mode textuel car le mode graphique ne permet pas de voir le déroulement d'une preuve avec Z/EVES, mais permet seulement de connaître le but courant.

Pour mieux comprendre, prenons un exemple simple tiré de notre banque de théorèmes et faisons-le exécuter par le mode textuel de Z/EVES. Tout d'abord, chargeons les axiomes et les théorèmes dans Z/EVES jusqu'au théorème *DistNegSurEquiv4* (si vous désirez réaliser l'exemple suivant en même temps que vous faites la lecture de ce rapport, on vous invite à lire préalablement la section 5.2.9).

Ensuite, on écrit le théorème *DistNegSurEquiv4* de la façon suivante dans l'interface Z/LaTeX :

```
\begin{theorem}{DistNegSurEquiv4}
\forall p, q: BoolGries @ \non (p \equiv q) \equiv \non p = q
\proof
reduce;
apply equivCommutativite1 to expression \non p \equiv q;
simplify;
rewrite;
\end{theorem}
```

Et voilà le résultat donné par Z/EVES :

```
theorem DistNegSurEquiv4
... theorem DistNegSurEquiv4
Beginning proof of DistNegSurEquiv4 ...

               p \in BoolGries \\
         \land q \in BoolGries \\
\implies \non (p \equiv q) \equiv \non p = q
Which simplifies
when rewriting with DistNegSurEquiv1, DistNegSurEquiv2
forward chaining using KnownMember\$declarationPart,
knownMember, `[internal items]`
with the assumptions select\_2\_1, select\_2\_2,
```

```
‘\non$declaration‘, equiv\$declaration, ‘[internal items]‘
to ...

              p \in BoolGries \\
         \land q \in BoolGries \\
\implies \non p \equiv q \equiv \non p = q
Applying equivCommutativite1 to
\non p \equiv q gives ...

              p \in BoolGries \\
         \land q \in BoolGries \\
\implies (\IF       \non p \in BoolGries \\
             \land q \in BoolGries
          \THEN q \equiv \non p
          \ELSE \non p \equiv q) \equiv \non p
         = q
Which simplifies
forward chaining using KnownMember\$declarationPart,
knownMember, ‘[internal items]‘
with the assumptions select\_2\_1, select\_2\_2,
‘\non$declaration‘, equiv\$declaration, ‘[internal items]‘
to ...

              p \in BoolGries \\
         \land q \in BoolGries \\
\implies (\IF \non p \in BoolGries
          \THEN q \equiv \non p \equiv \non p = q
          \ELSE \non p \equiv q \equiv \non p = q)
Which simplifies
when rewriting with equivCommutativite3,
applicationInDeclaredRangeFun, inPowerSelf
forward chaining using KnownMember\$declarationPart,
knownMember, ‘[internal items]‘
with the assumptions select\_2\_1, select\_2\_2,
equiv\$declaration, fun\_type, ‘\non$declaration‘,
‘[internal items]‘ to ...
true
```

Avant de prendre peur, prenez quelques instants pour bien décortiquer cette réponse. Vous verrez qu'elle est plus compréhensible qu'elle paraît au premier abord. Nous la décortiquerons ensemble.

Tout d'abord, Z/EVES nous annonce qu'il a bien reçu et enregistré le théorème en marquant :

```
theorem DistNegSurEquiv4
... theorem DistNegSurEquiv4
```

Ensuite, le démonstrateur se met en position pour être prêt à démontrer ce théorème :

```
Beginning proof of DistNegSurEquiv4 ...

                p \in BoolGries \\
          \land q \in BoolGries \\
\implies \non (p \equiv q) \equiv \non p = q
```

Ceci est notre but initial, notre théorème. N'oubliez pas que lorsque Z/EVES se met en position pour démontrer un théorème, il exécute préalablement la commande `prenex` pour enlever les quantificateurs universels. Ensuite, il affiche le théorème suivant la priorité des opérateurs (il les affiche de façon à ce que l'on reconnaisse rapidement les différents prédicats et comment ils sont reliés entre eux).

Pour continuer, Z/EVES accomplit la première commande donnée dans les directives pour réaliser la preuve (la commande `reduce`). Il ne mentionne pas la commande qu'il effectue, mais on voit tout ce qu'elle fait réaliser au démonstrateur. Voilà la partie associée à la commande `reduce` :

```
Which simplifies
when rewriting with DistNegSurEquiv1, DistNegSurEquiv2
forward chaining using KnownMember\$declarationPart,
knownMember, `[internal items]`
with the assumptions select\_2\_1, select\_2\_2,
`\non$declaration`, equiv\$declaration, `[internal items]`
to ...

                p \in BoolGries \\
          \land q \in BoolGries \\
\implies \non p \equiv q \equiv \non p = q
```

Dans cette commande, on voit que Z/EVES simplifie le but courant en utilisant les règles de réécriture. Il utilise autant les règles que nous lui avons données que les règles qu'il possède déjà. Premièrement, on voit que Z/EVES réécrit le but en utilisant l'axiome *DistNegSurEquiv1* et, ensuite, l'axiome *DistNegSurEquiv2*. Pour continuer la réduction, le démonstrateur automatique de théorèmes utilise des règles de transfert (des théorèmes d'usage `frule`) qui sont déjà dans les outils de Z/EVES lorsqu'il se charge. Ces règles sont : *KnownMember*\$*declarationPart*, *knowMember* et *'[internal items]'*. La première règle est en fait une règle que Z/EVES s'est créée à partir des définitions que les concepteurs lui ont données. Les règles créées par Z/EVES sont de la forme *nom_objet_Z*\$*type_théorème_créé*. Dans ce cas-ci, ce théorème vient du schéma *knownMember* (le théorème créé par le démonstrateur ne contient que la partie de déclaration du schéma *KnownMember*). Pour en savoir plus sur les théorèmes créés par Z/EVES (ils ne sont pas pertinents pour la présente recherche), lisez [16] (surtout la section 3.5). Les *'[internal items]'* sont des objets que nous ne pouvons pas toucher dans Z/EVES et qui lui permettent de faire des liens entre les divers opérateurs et prédicats. Ensuite, le démonstrateur nous indique qu'il a utilisé ces règles de transfert avec les hypothèses données (autant les hypothèses du but courant que celles des différents théorèmes utilisés pour la démonstration). Finalement, la commande se termine en nous donnant le but courant à prouver[22] :

```
                p \in BoolGries \\
          \land q \in BoolGries \\
\implies \non p \equiv q \equiv \non p = q
```

Ensuite, Z/EVES passe à l'autre commande invoquée (de façon séquentielle) et applique la règle de réécriture *equivCommutativite1* seulement aux occurrences de l'expression

```
\non p \equiv q.
```

Cette règle de réécriture a été appliquée de la façon que Z/EVES se représente les opérateurs à l'interne. Donc, le bout de théorème modifié par cette commande est mis sous la forme conditionnelle. Cette commande correspond à la section de la preuve suivante dans le résultat :

```
Applying equivCommutativite1 to
\non p \equiv q gives ...
```

[22]Une commande se termine toujours en nous donnant le but courant à prouver.

```
              p \in BoolGries \\
         \land q \in BoolGries \\
\implies (\IF       \non p \in BoolGries \\
              \land q \in BoolGries
          \THEN q \equiv \non p
          \ELSE \non p \equiv q) \equiv \non p
         = q
```

Pour continuer, Z/EVES exécute la troisième commande : la commande `simplify`. Cette commande se lit de la même manière que la commande `reduce` dont l'explication a été donnée plus haut. Voilà la section qui correspond à cette commande :

```
Which simplifies
forward chaining using KnownMember\$declarationPart,
knownMember, `[internal items]`
with the assumptions select\_2\_1, select\_2\_2,
`\non$declaration`, equiv\$declaration, `[internal items]`
to ...

              p \in BoolGries \\
         \land q \in BoolGries \\
\implies (\IF \non p \in BoolGries
          \THEN q \equiv \non p \equiv \non p = q
          \ELSE \non p \equiv q \equiv \non p = q)
```

Si vous désirez connaître exactement en quoi consiste chacun des théorèmes qui sont utilisés dans la démonstration, mais qui ne proviennent pas de la logique implantée dans cette recherche, vous trouverez cette information dans le *Mathematical Toolkit* de Z/EVES [17].

Il reste encore la dernière commande de la preuve à exécuter : la commande `rewrite`. Les effets de cette commande sont définis dans la section 5.2.6. Voilà la partie qui correspond à cette commande dans la preuve :

```
Which simplifies
when rewriting with equivCommutativite3,
applicationInDeclaredRangeFun, inPowerSelf
forward chaining using KnownMember\$declarationPart,
knownMember, `[internal items]`
with the assumptions select\_2\_1, select\_2\_2,
```

```
equiv\$declaration, fun\_type, `\non$declaration`,
`[internal items]` to ...
true
```

Il n'y a rien de bien nouveau à traiter dans le résultat de cette commande. Nous en avons déjà discuté dans les autres commandes. La seule différence est le prédicat final (le but courant). Ce prédicat est *true*. Lorsqu'on arrive avec le but courant *true* dans une preuve, Z/EVES considère que la preuve est terminée et la démonstration s'arrête là pour ce théorème.

Donc, on a démontré le théorème. Toutes les démonstrations peuvent se lire comme nous venons de le faire. Quelquefois, il se peut qu'il y ait une étape que nous n'avons pas expliquée ici. Pour combler cette lacune, consultez la définition de la commande que vous avez utilisée pour réaliser cette étape. Normalement, la signification devrait vous sauter aux yeux.

Vous pouvez donc maintenant lire une preuve donnée par l'interface Z/LaTeX.

5.2.9 Utilisation de la banque d'axiomes et de théorèmes

À cette étape-ci, vous devriez comprendre d'une façon générale le fonctionnement de Z/EVES. Maintenant, voici la méthode rapide pour être capable de charger notre banque d'axiomes et de théorèmes dans le démonstrateur pour que vous soyez capable de les utiliser. De plus, cette section montre comment modifier cette banque de données.

Comment accéder à la banque d'axiomes et de théorèmes ?

Dépendant de l'interface que vous choisissez pour charger la banque, la méthode différera.

Mode textuel

En mode textuel, le chargement de la banque d'axiomes et de théorèmes est très simple si vous choisissez de charger la banque en entier et cela se complique un peu si vous désirez ne charger qu'une partie de la banque.

Pour charger la banque en entier Le chargement de la banque de données dans son entier est très simple, mais elle est aussi *très* coûteuse en temps. Elle est *très* coûteuse en temps car il faut que Z/EVES réalise *toutes* les preuves des théorèmes de la banque de données avant que la banque soit opérationnelle. Le chargement de cette banque en son entier nécessite cinquante et une minutes pour un *Pentium II* 400MHz avec 64 Megs de mémoire vive. *Attention :* si vous désirez réduire le délai d'attente pour le chargement de la banque de théorèmes dans le but d'une utilisation rapide, alors remplacez toutes les occurrences du nom de fichier : *AlgBoolZLaTeX.zed* par le nom de fichier : *AlgBoolZLaTeX2.zed*. Ce fichier contient les mêmes théorèmes et axiomes que le fichier *AlgBoolZLaTeX.zed*, mais il ne contient aucune démonstration des divers théorèmes. Donc, le gain de temps est très visible. En fait, ce fichier ne nécessite qu'environ dix secondes pour se charger en entier.

Voilà les étapes qu'il faut suivre pour charger la banque :

1. Ouvrez Z/EVES en mode textuel (dans la plateforme Windows, cliquez sur `Démarrer` → `Programmes` → `Z-EVES 2.1` → `Z-EVES`).
2. Sélectionnez dans le menu *File*, la commande *Read*.
3. Donnez le chemin d'accès correct pour le fichier contenant la banque d'axiomes et de théorèmes et appuyez sur *Ouvrir*. Dans la présente recherche, le fichier se nomme *AlgBoolZLaTeX.zed*. Il faudrait donc spécifier le chemin d'accès : (chemin jusqu'au répertoire contenant le fichier)\AlgBoolZLaTeX.zed.
4. Attendre que la banque se charge (cela risque de prendre quelque temps et vous allez voir tout au long de cette opération des preuves défiler sous vos yeux.)

Pour charger une partie de la banque de données On peut aussi raccourcir le temps d'attente pour accéder aux théorèmes et axiomes si on ne désire pas avoir toute la banque pour travailler. Quelquefois, il est plus utile (et surtout moins long) de ne charger qu'une partie des divers théorèmes dans Z/EVES. Par exemple, on pourrait vouloir ne charger que les trentes premiers théorèmes et axiomes pour vérifier ce qu'on peut prouver grâce à eux.

Si tel est votre désir, voilà les étapes à suivre pour charger seulement la

partie de la banque qui vous intéresse :

1. Ouvrez votre éditeur de texte préféré (qui doit être capable de lire des fichiers LaTeX).
2. Visionnez la banque d'axiomes et de théorèmes et décidez jusqu'à quel théorème vous désirez charger les informations (si vous choisissez de prendre des théorèmes de façon éparse, n'oubliez pas de vérifier si vous possédez tous les théorèmes et les axiomes qui sont nécessaires pour démontrer chacun des théorèmes).
3. Copiez les informations de ces théorèmes dans le presse-papier de votre système d'exploitation.
4. Ouvrez Z/EVES en mode textuel (dans la plateforme Windows, cliquez sur `Démarrer` → `Programmes` → `Z-EVES 2.1` → `Z-EVES`).
5. Collez dans Z/EVES les informations que vous venez de copier (vous pouvez réaliser ceci grâce à la commande *Paste* dans le menu *Edit* de l'interface textuel).
6. Appuyez sur la touche *entrée* sur votre clavier juste devant la dernière instruction que vous venez de coller. Ceci enclanchera Z/EVES qui se mettra à lire les informations et à exécuter les démonstrations.
7. Attendez que le démonstrateur ait terminé toutes ses démonstrations et, ensuite, la banque de données que vous désiriez sera chargée.

Mode graphique

Charger la banque d'axiomes et de théorèmes dans le mode graphique est *beaucoup* plus rapide et simple que dans le mode textuel. La méthode est presque identique pour charger toute la banque de données ou seulement une partie de celle-ci. En fait, elle ne diffère que par la dernière étape. À vrai dire, la raison pour laquelle le chargement des théorèmes est beaucoup moins long dans le mode graphique que dans celui textuel est le fait que le mode graphique ne donne que les théorèmes sans vérifier ni prendre en considération les preuves de ceux-ci. Donc, cela revient presqu'à dire que tous les théorèmes sont rendus des axiomes car ils n'ont pas de preuves dans le mode graphique, mais ce ne sont pas des axiomes car on *peut* prouver chacun des théorèmes avec les axiomes qu'on a.

La méthode commune pour accéder en partie ou en totalité à la banque d'axiomes et de théorèmes dans le mode graphique est la suivante :

1. Ouvrez Z/EVES dans le mode graphique (dans la plateforme Windows, cliquez sur `Démarrer` → `Programmes` → `Z-EVES 2.1` → `Z-EVES GUI`).
2. Sélectionnez dans le menu *File*, la commande *Import* ....
3. Donnez le chemin d'accès correct pour le fichier contenant la banque d'axiomes et de théorèmes et appuyez sur *Ouvrir*. Dans la présente recherche, le fichier se nomme *AlgBoolZLaTeX.zed*. Il faudrait donc spécifier le chemin d'accès : (chemin jusqu'au répertoire contenant le fichier)\AlgBoolZLaTeX.zed.

À ce stade-ci, vous devriez voir dans la fenêtre *Specification* de votre démonstrateur tous les axiomes et théorèmes avec, dans les deux colonnes de droite, des points d'interrogation. Le point d'interrogation le plus à gauche veut simplement signifier que cette ligne n'a pas été vérifiée syntaxiquement par Z/EVES. Donc, le démonstrateur n'a pas encore accepté cette ligne-là. Le point d'interrogation à droite du premier signifie simplement que l'on n'a pas encore prouvé cette ligne-là.

Si vous désirez travailler avec toute la banque de théorèmes, il faut aller à la dernière ligne d'instruction (le dernier théorème) et appuyer sur le bouton de droite de la souris sur cette ligne. À ce moment-là, un menu flottant apparaît et vous sélectionnez la commande *Check up to here*. Ensuite, vous attendez que tous les théorèmes aient été vérifiés syntaxiquement. Avec un *Pentium II* 400 MHz avec 64 Megs de mémoire vive, cette opération a pris une minute et huit secondes.

Si vous désirez travailler avec seulement une partie de la banque de théorèmes, vous devez connaître les théorèmes et les axiomes que vous souhaitez utiliser. Ensuite, pour chacun de ces axiomes et théorèmes, il faut, s'ils ne sont pas dans un ordre séquentiel, cliquer sur le bouton droit de la souris sur chacun d'eux et il faut sélectionner la commande *Check* dans le menu déroulant. Il faut préalablement que vous ayez opéré cette commande *Check* sur le type `BoolGries`, ainsi que sur chacun des opérateurs et des constantes de ce type. Si vous voulez charger seulement une partie de la banque de données, mais que celle-ci se présente d'une façon séquentielle, alors vous pouvez simplement suivre les intructions données pour le chargement des théorèmes dans le mode graphique pour toute la banque de théorèmes, mais vous exécutez la dernière commande sur le dernier théorème que vous souhaitez traiter.

C'était les méthodes rapides pour charger la banque de données utilisée lors de cette présente recherche.

Comment modifier cette banque ?

On peut modifier la banque d'axiomes et de théorèmes soit en ajoutant un axiome ou un théorème, soit en enlevant un théorème (si on enlève un axiome, alors ceci change totalement le système d'axiomes et on risque de ne plus pouvoir prouver certains théorèmes).

Pour ajouter un axiome ou un théorème

Il y a deux manières d'ajouter un axiome ou un théorème : soit d'une manière *temporaire*, soit d'une manière *permanente*.

Pour ajouter un axiome ou un théorème d'une manière permanente, cela est facile. Il s'agit simplement d'éditer le fichier contenant l'algèbre booléenne fournie (dans cette présente recherche ce fichier s'appelle *AlgBoolZ-LaTeX.zed*) dans votre éditeur de texte favori qui supporte LaTeX. Ensuite, vous vous déplacez à l'endroit où vous souhaitez insérer le nouveau théorème ou axiome et vous écrivez tout simplement à cet endroit, le nouveau théorème ou axiome désiré. *Attention :* il faut respecter la syntaxe définie dans la section 5.2.5 pour que Z/EVES accepte votre entrée. De plus, il ne faut pas oublier que si vous entrez un théorème, alors vous devez connaître les commandes nécessaires à la réalisation de sa preuve (vous ne pourrez la vérifier dans votre éditeur de texte, vous devrez charger le fichier dans Z/EVES pour savoir si ces commandes sont exactes). Ensuite, vous enregistrez le tout et le tour est joué. Votre banque est maintenant enrichie d'un axiome ou d'un théorème de plus de façon permanente. Pour vérifier si tout a bien fonctionné, vous pouvez charger votre nouvelle banque dans Z/EVES.

Si vous souhaitez ajouter un théorème d'une manière temporaire, ceci devra se réaliser dans Z/EVES. La façon d'ajouter un axiome ou un théorème est différente selon le mode utilisé.

Mode textuel Pour le mode textuel, après avoir chargé votre banque de théorèmes initiale, vous inscrivez tout simplement, à la ligne de commande, le nouveau théorème ou axiome que vous désirez implanter. Celui-ci devra

respecter la syntaxe du Z/LaTeX telle que définie dans la section 5.2.5. Il faut aussi réaliser la preuve ensuite si vous avez entré un nouveau théorème.

Par exemple, vous auriez pu vouloir ajouter ce nouveau théorème :

$$\forall p, q, r : \texttt{BoolGries} \bullet p \Rightarrow q \vee r \equiv (p \Rightarrow q) \vee (p \Rightarrow r).$$

À ce moment-là, vous auriez dû écrire :

```
\begin{theorem}{NouveauTheoreme}
\forall p, q, r: BoolGries @ p \implique q \ou r = (p \implique
q) \ou (p \implique r)
\proof
use DefAltDeImplication1[q:= q \ou r];
use DefAltDeImplication1;
use DefAltDeImplication1[q:= r];
use DistOuSurOu1[p:= \non p];
with disabled (implicationDef, DeMorgan1, DeMorgan2,
ouAssociativite, ouCommutativite) prove;
\end{theorem}
```

Mode graphique Pour ajouter un axiome ou un théorème dans le mode graphique, commencez tout d'abord par charger votre banque de données dans l'interface graphique de Z/EVES. Ensuite, sélectionnez la fenêtre du *mini-editor* et entrez votre axiome ou votre théorème en suivant la syntaxe montrée à la section 5.2.7. Ne vous préoccupez pas maintenant de la preuve du théorème.

Tout d'abord, vérifiez où vous souhaitez implanter ce nouveau théorème ou axiome dans votre liste. Ensuite, copiez votre théorème ou axiome dans le presse-papier de Z/EVES (sélectionnez votre nouvel arrivant et, dans le menu *Edit*, choisissez la commande *Copy*). Placez le curseur juste à l'endroit où vous voulez insérer ce bijou et copiez-le à cet endroit (dans le menu *Edit*, choisissez la commande *Paste*). Ensuite, il vous faut effacer le théorème ou l'axiome de l'endroit initial où il se terrait pour la première fois (la fin de la liste). Pour ceci, cliquez sur le bouton droit de la souris sur ce théorème et appuyez sur *delete* dans le menu flottant.

Ensuite, il faut vérifier syntaxiquement votre axiome ou votre théorème pour qu'il s'inscrive dans la banque interne de Z/EVES. *Attention :* lorsque

vous utilisez la commande *Check* sur un objet quelconque (ici, un axiome ou un théorème) qui n'a pas encore été vérifié, à ce moment-là Z/EVES enlève la vérification de chacun des objets en dessous de cet objet. Donc, après avoir vérifié syntaxiquement votre nouvel arrivant, il faut revérifier les théorèmes ou axiomes qui se sont désactivés durant cette opération.

Si vous ajoutiez un axiome, votre travail s'arrête ici. Votre démonstrateur contient désormais un axiome de plus ! Si vous ajoutiez un théorème, alors votre travail continue ; il faut réaliser la preuve de ce théorème. Pour parvenir à ceci, suivez la méthode de la section 5.2.7 pour entrer votre preuve.

Dans le mode graphique, pour ajouter le même nouveau théorème que celui ajouté dans le mode textuel (l'exemple précédent), vous auriez dû inscrire :

> **theorem** *NouveauTheoreme*
> $\forall p, q, r : BoolGries \bullet p$ *implique* q *ou* $r =$ *(*p *implique* q*) ou (*p *implique* r*)*

Ensuite, vous auriez dû réaliser la même preuve que dans l'exemple précédent pour que ce théorème soit considéré comme un théorème par Z/EVES.

Pour enlever un théorème

De la même façon que pour ajouter un théorème, il y a deux manières d'enlever un théorème : la manière *temporaire* et la manière *permanente*.

Encore une fois, la manière permanente est très simple : vous éditez le fichier contenant l'algèbre booléenne dans un éditeur de texte supportant LaTeX et vous effacez tout simplement ce qui a trait à ce théorème dans la banque de données.

Ce coup-ci, la manière temporaire ne fonctionne que dans le mode graphique. Il faut simplement, une fois que la banque est chargée, cliquer avec le bouton droit de la souris sur le théorème dont vous désirez la mort. Ensuite, vous sélectionnez la commande *Delete* dans le menu flottant. Finalement, s'il y avait des théorèmes ou des axiomes après le théorème que vous venez d'enlever, ces théorèmes et axiomes sont désactivés. Pour les réactiver, vous devez les revérifier syntaxiquement.

5.2.10 Exemple complet d'une démonstration de théorème

Dans cette section, nous nous proposons de montrer une démonstration complète d'un théorème, étape par étape. Ce rapport est déjà parsemé d'exemples, mais ces exemples sont plutôt concentrés sur la matière expliquée et n'incluent jamais toutes les étapes d'une démonstration. Nous entendons donc réaliser un exemple complet pour permettre une bonne compréhension générale du démonstrateur automatique de théorèmes.

Prenons un théorème simple, mais intéressant dans sa démonstration et réalisons cet exemple dans les deux modes pour montrer les différences. Le théorème choisi est le théorème *DistOuSurOu1* :

$$\forall p, q, r : \texttt{BoolGries} \bullet p \vee (q \vee r) \equiv (p \vee q) \vee (p \vee r)$$

Avant même de nous lancer dans l'aventure périlleuse de démontrer ce théorème, il est bon d'avoir une idée de la preuve. Z/EVES a un bon degré d'automatisme, mais il nécessite tout de même notre aide de temps à autre. Tentons donc de démontrer ce théorème par nous-même :

$\forall p, q, r : \texttt{BoolGries} \bullet p \vee (q \vee r) \equiv (p \vee q) \vee (p \vee r)$
$=$ $\langle$ Idempotence de la disjonction (3.34) $\rangle$
$\forall p, q, r : \texttt{BoolGries} \bullet (p \vee p) \vee (q \vee r) \equiv (p \vee q) \vee (p \vee r)$
$=$ $\langle$ Associativité de la disjonction (3.33), avec $q, r := p, q \vee r$ $\rangle$
$\forall p, q, r : \texttt{BoolGries} \bullet p \vee (p \vee (q \vee r)) \equiv (p \vee q) \vee (p \vee r)$
$=$ $\langle$ Associativité de la disjonction (3.33) $\rangle$
$\forall p, q, r : \texttt{BoolGries} \bullet p \vee ((p \vee q) \vee r) \equiv (p \vee q) \vee (p \vee r)$
$=$ $\langle$ Commutativité de la disjonction (3.32) $\rangle$
$\forall p, q, r : \texttt{BoolGries} \bullet p \vee ((q \vee p) \vee r) \equiv (p \vee q) \vee (p \vee r)$
$=$ $\langle$ Associativité de la disjonction (3.33), avec $p, q := q, p$ $\rangle$
$\forall p, q, r : \texttt{BoolGries} \bullet p \vee (q \vee (p \vee r)) \equiv (p \vee q) \vee (p \vee r)$
$=$ $\langle$ Associativité de la disjonction (3.33), avec $r := p \vee r$ $\rangle$
$\forall p, q, r : \texttt{BoolGries} \bullet (p \vee q) \vee (p \vee r) \equiv (p \vee q) \vee (p \vee r)$
- Réflexivité de l'équivalence (3.7), avec $p := (p \vee q) \vee (p \vee r)$ $\square$

Nous avons réussi à démontrer ce théorème. Donc, c'est un théorème dans notre système d'axiomes. Voyons comment Z/EVES se débrouille pour réaliser la démonstration de ce théorème.

Mode textuel

Pour débuter, il faut d'abord donner à Z/EVES les différents théorèmes et axiomes précédant celui que nous désirons démontrer. Donc, éditons le fichier *AlgBoolZLaTeX.zed* et copions dans le presse-papier du système d'exploitation les informations précédant le *lemme6* (nous ne copions pas les théorèmes *lemme6* et *lemme7* car nous devrons les créer dans la démonstration). Ensuite, démarrons Z/EVES en mode Z/LaTeX. Les étapes à suivre pour réaliser ceci sont décrites dans la section 5.2.9. Plaçons les informations dans la fenêtre de Z/EVES et appuyons sur *entrée* devant la dernière ligne d'instruction apparaissant à l'écran. Attendons quelques instants.

Maintenant que nous avons tous nos outils, nous pouvons travailler. Tout d'abord, il faut entrer le théorème que nous désirons prouver (dans notre cas, le théorème *DistOuSurOu1*). Nous le définissons comme ceci :

```
\begin{theorem}{DistOuSurOu1}
\forall p, q, r: BoolGries @ p \ou (q \ou r) = p \ou q \ou
(p \ou r)
\end{theorem}
```

Ensuite, Z/EVES se met en place pour la démonstration du théorème. Vérifions tout d'abord si, à l'aide des axiomes, Z/EVES serait capable de démontrer le théorème sans notre aide. Donnons lui la commande `prove` et voyons comment il se débrouille.

```
prove;
```

Nous nous rendons compte que Z/EVES n'a pas vraiment progressé. Il reste en fait au but :

```
              p \in BoolGries \\
        \land q \in BoolGries \\
        \land r \in BoolGries \\
\implies p \ou (q \ou r) = p \ou (q \ou (p \ou r))
```

À vrai dire, le démonstrateur n'a utilisé que l'associativité de la disjonction avec cette commande. Donc, ce n'est pas un bon début. Recommençons

donc en décidant de le diriger. Tapons tout d'abord :

```
retry;
```

pour revenir au début de la démonstration (au but initial).

Ensuite, servons-nous de la preuve réalisée plus haut pour nous aider dans notre direction de Z/EVES. À première vue, nous voyons que la démonstration est un peu complexe et qu'elle nécessite beaucoup d'étapes. Ce qui pourrait nous aider à ce moment-là est un lemme[23]. Pour tenter de démontrer le théorème, nous allons utiliser l'heuristique de la progression « par les deux bouts » (voir [4] pour la description de cette heuristique). Tout d'abord, tentons de voir ce qui serait simple à démontrer dans notre preuve. Nous pourrions séparer notre preuve en deux parties[24] : la première partie consisterait à démontrer que :

$$\forall p,\, q,\, r : \texttt{BoolGries} \bullet p \lor (q \lor r) \equiv (p \lor p) \lor (q \lor r)$$

et la deuxième partie consisterait en la démonstration de :

$$\forall p,\, q,\, r : \texttt{BoolGries} \bullet (p \lor p) \lor (q \lor r) \equiv (p \lor q) \lor (p \lor r).$$

Ces deux parties représenteront donc deux lemmes que nous pourrons utiliser dans la démonstration du théorème principal. Commençons tout d'abord par prouver le premier lemme (que nous appelerons *lemme6* pour suivre la notation utilisée dans cette recherche). Donnons à Z/EVES ce premier lemme :

```
\begin{theorem}{lemme6}
\forall p, q, r: BoolGries @ p \ou (q \ou r) = p \ou p \ou
(q \ou r)
\end{theorem}
```

Ensuite, Z/EVES est en position pour prouver ce lemme. Tentons de le prouver par une commande simple : `rewrite`, car nous savons que ce théorème ne nécessite pas beaucoup d'étapes et ces étapes devraient être automatique dans Z/EVES. Entrons donc la commande :

```
rewrite;
```

Cette commande nous donne le but courant *true*. Z/EVES a donc démontré le théorème *lemme6* avec nos règles de réécriture.

[23] « Un lemme est un théorème de moindre importance utilisé dans la preuve d'un théorème auquel on s'intéresse davantage » [4].

[24] Ceci n'est qu'une suggestion de preuve. Il en existe surement plusieurs, mais c'est celle-ci que nous privilégierons.

Démontrons alors l'autre partie et appelons-la *lemme7*. Nous donnons tout d'abord à Z/EVES le lemme pour qu'il puisse se mettre en position pour la démonstration de celui-ci :

```
\begin{theorem}{lemme7}
\forall p, q, r: BoolGries @ p \ou p \ou (q \ou r) = p \ou q
\ou (p \ou r)
\end{theorem}
```

Maintenant que Z/EVES est prêt pour la démonstration de ce lemme, donnons-lui la première instruction. Tentons tout d'abord de voir ce que Z/EVES peut faire grâce à nos règles de réécriture sans que nous l'aidions :

```
rewrite;
```

Nous nous rendons compte que nous n'avons pas beaucoup progressé car Z/EVES a décidé d'utiliser l'axiome de l'idempotence de la disjonction pour réduire le théorème. Il faut donc désactiver ce théorème pour permettre de progresser quelque peu. Commençons par revenir au but initial par la commande :

```
retry;
```

et donnons-lui une commande plus spécifique :

```
with disabled (ouIdempotence) rewrite;
```

Voilà le but courant que Z/EVES nous retourne :

```
              p \in BoolGries \\
        \land q \in BoolGries \\
        \land r \in BoolGries \\
\implies p \ou (p \ou (q \ou r)) = p \ou (q \ou (p \ou r))
```

Nous avons progressé quelque peu. Il s'agit maintenant de commuter p et q dans cette expression et le tour est joué ! Commençons par permuter tout d'abord q et $p \lor r$ dans l'expression $q \lor (p \lor r)$. Ce qui se traduit en Z/EVES par la commande :

```
apply ouCommutativite to expression q \ou (p \ou r);
```

Z/EVES donne la nouvelle expression sous une forme conditionnelle, mais elle est tout de même compréhensible. Maintenant, vérifions si le démonstrateur est capable de terminer la preuve seul en lui donnant la commande :

```
rewrite;
```

Finalement, Z/EVES nous apprend qu'il a été capable de terminer la preuve seul en nous retournant le but courant *true*. Nous avons donc notre *lemme7* qui est opérationnel. Nous avons maintenant tous nos outils pour notre démonstration du théorème initial.

Commençons tout d'abord par reprendre la preuve du théorème *DistOuSurOu1*. Il faut préalablement comprendre que Z/EVES fonctionne de façon séquentielle. Ce qui fait en sorte que nous ne pouvons utiliser dans une démonstration un théorème qui a été défini après le moment où nous avons implanté le théorème initial. Donc, nous ne pourrons utiliser l'énoncé de notre théorème initial pour réaliser la preuve finale car les lemmes *lemme6* et *lemme7* ont été définis après ce théorème. Nous devons donc réentrer notre théorème avec un nom différent pour pouvoir recommencer notre preuve de ce théorème[25] :

```
\begin{theorem}{DistOuSurOu}
\forall p, q, r: BoolGries @ p \ou (q \ou r) = p \ou q \ou
(p \ou r)
\end{theorem}
```

Nous sommes revenus au début de la preuve du théorème initial. Donnons à notre démonstrateur des hypothèses supplémentaires pour ce théorème : les deux nouveaux lemmes que nous venons de prouver. Nous pouvons réaliser ceci grâce aux commandes :

```
use lemme6;
```

et :

```
use lemme7;
```

Finalement, il ne reste qu'à dire à Z/EVES de démystifier le tout pour réaliser la preuve. Essayons la commande `rewrite` pour effectuer cela (c'est la commande la mieux adaptée pour nos besoins ici) :

```
rewrite;
```

Z/EVES nous retourne le but courant *true*. Z/EVES a donc réussi à

[25]Il y aura donc deux théorèmes qui auront le même prédicat, mais avec des noms différents. Ceci n'est que pour une fin de démonstration seulement. Après, vous n'avez qu'à transcrire les théorèmes dans le bon ordre et vous n'aurez pas à définir deux fois le même théorème avec des noms différents. Z/EVES ne tentera pas de démontrer le second théorème à l'aide de l'énoncé du premier car le premier théorème a été implanté sans usage (donc Z/EVES ne peut l'utiliser automatiquement).

démontrer le théorème. Maintenant, ce théorème est accessible pour une utilisation future. Cette démonstration s'est effectuée dans le mode textuel.

Mode graphique

Dans le mode graphique, la preuve est théoriquement la même que celle dans le mode textuel. Donc, vous n'aurez pas à refaire un cheminement complexe pour arriver à la démonstration. Commençez tout d'abord par ouvrir Z/EVES en mode graphique et chargez la banque de théorèmes jusqu'au théorème *DistOuSurOu1* (pour la réalisation de cette étape, voir la section 5.2.9).

Pour cette démonstration, vous vous êtes épargnés l'écriture de ces théorèmes dans le mode graphique, mais cela est uniquement car vous avez déjà réalisé la même chose dans le mode textuel. L'écriture d'un théorème dans le mode graphique ne devrait pas vous poser de difficultés.

Cliquez sur le bouton droit de la souris sur le théorème *lemme6* et sélectionnez la commande *Show proof* dans le menu flottant qui apparaît. Ceci vous a permis d'accéder à la fenêtre *Proof* pour le théorème *lemme6*.

Ce lemme se démontre de la même manière que dans le mode textuel. Ce qui veut dire que vous n'avez qu'à appliquer la commande `rewrite` qui se situe dans le menu du bouton *Reduction*. Vérifiez si, dans la fenêtre du bas, le prédicat est *true*. Ceci signifie que la démonstration de ce lemme est terminée. Revenez à la fenêtre principale donnant accès à tous vos théorèmes et axiomes. Pour cela, cliquez sur le bouton *Window* et sélectionnez l'option *Specification*.

Il faut maintenant réaliser la preuve du lemme *lemme7*. Accédez à la fenêtre *Proof* de la même manière que vous avez utilisée pour accéder à celle du lemme *lemme6*. Il faut prouver ce théorème. Vous pouvez le prouver exactement comme dans le mode textuel. Pour cela, il vous faut entrer les commandes manuellement.

Tout d'abord, sélectionnez la commande *New Command* dans le menu *Edit*. Dans la fenêtre blanche de l'éditeur, écrivez l'instruction[26] :

[26]Le mode graphique n'utilise pas le « ; » comme fin d'instruction.

with disabled (ouIdempotence) rewrite

Ensuite, sélectionnez la commande *Done* dans le menu *File*. Ceci vous ramène à la fenêtre *Proof* avec, si tout s'est bien déroulé, le texte de la commande que vous avez entré au clavier. Cette commande devrait être affectée d'un point d'interrogation à sa gauche pour spécifier qu'elle n'a pas encore été activée. Activez-la en cliquant avec le bouton droit de la souris sur le texte de la commande et sélectionnez la commande *Run*. Normalement, vous devriez voir le point d'interrogation changer pour un « Y » pour dire que la commande a eu un effet sur le but courant.

Ensuite, il faut entrer la deuxième commande de la preuve manuellement comme vous venez de le faire pour la précédente sauf que vous écrivez, dans le *mini-editor*, la commande :

apply ouCommutativite to expression q ou (p ou r)

Ensuite, vous effectuez cette commande à l'aide de la commande *Run*. Ceci vous donne le même but courant qu'à la même étape dans la preuve de ce lemme dans le mode textuel. Finalement, vous complétez la preuve à l'aide de la commande `rewrite` accessible dans les commandes du bouton *Reduction*. Le lemme *lemme7* est prouvé. Revenez à la fenêtre *Specification*.

Normalement, à cette étape, devant les théorèmes *lemme6* et *lemme7*, il devrait y avoir deux « Y ». Si cela n'est pas le cas, revisez la séquence réalisée jusqu'à présent[27].

Vous êtes maintenant prêts à prouver le théorème *DistOuSurOu1*. Accédez à la fenêtre *Proof* de ce théorème. Ensuite, entrez manuellement et *une à la fois* les commandes `use lemme6` et `use lemme7` de la même manière que celle utilisée jusqu'à présent dans le mode graphique. Terminez la preuve en choisissant la commande `rewrite` dans le bouton *Reduction*. Retournez ensuite dans la fenêtre *Specification* et votre théorème est démontré !

Cet exemple devrait vous avoir permis de mieux comprendre le fonctionnement de Z/EVES et comment travailler avec celui-ci.

[27]Le premier « Y » signifie que le théorème est enregistré dans Z/EVES et l'autre « Y » que ce théorème a été prouvé.

5.2.11 Spécialisation du démonstrateur

L'algèbre booléenne que nous présentons dans cette recherche est intéressante, mais elle n'est peut-être pas *totalement* adaptée aux besoins d'un utilisateur particulier. À ce moment-là, la solution est d'étendre la logique. L'augmentation de cette algèbre se réalise très facilement grâce à Z/EVES et à la définition du type `BoolGries` que nous avons implanté.

Tout d'abord, nous pouvons étendre cette logique en lui donnant de nouveaux opérateurs. Cela permet d'avoir de nouveaux axiomes pour définir le comportement de cet opérateur et même de prouver ensuite de nouveaux théorèmes que nous ne pouvions pas prouver précédemment. Pour réaliser ceci, nous devons tout d'abord connaître le nouvel opérateur et les axiomes le régissant.

Par exemple, prenons un nouvel opérateur que nous appellerons *coeur* et donnons-lui ce symbole : ♡. Disons que *coeur* a une priorité de 5 (voir la figure de la section 5.2.5) et que cet opérateur est associatif, commutatif et qu'il peut se distribuer sur l'équivalence. Cet opérateur est une fonction qui prend deux arguments de type `BoolGries` et qui retourne un type `BoolGries`. Nous devons commencer par déclarer ce nouvel opérateur (en suivant la syntaxe présentée dans la section 5.2.5) :

```
\syndef{\coeur}{infun5}{\coeur}
```

Ensuite, nous devons déclarer son type dans une définition d'axiome générique :

```
\begin{axdef}
\_ \coeur \_: BoolGries \cross BoolGries \fun BoolGries
\end{axdef}
```

Maintenant, il ne reste qu'à entrer les axiomes de cet opérateur :

L'associativité de ♡ :

```
\begin{theorem}{rule coeurAssociativite}
\forall p, q, r: BoolGries @ p \coeur q \coeur r = p \coeur (q
\coeur r)
\end{theorem}
```

La commutativité de ♡ :

```
\begin{theorem}{rule coeurCommutativite}
```

```
\forall p, q: BoolGries @ p \coeur q = q \coeur p
\end{theorem}
```

La distributivité de ♡ sur ≡ :

```
\begin{theorem}{rule DistCoeurSurEquiv}
\forall p, q, r: BoolGries @ p \coeur (q \equiv r) = \non p
\coeur q \equiv p \coeur r
\end{theorem}
```

Notre nouvel opérateur est donc fonctionnel ! De plus, notre logique vient d'être étendue !

Une autre façon d'étendre la logique implantée dans cette recherche est de donner de nouvelles constantes au type (l'ensemble) `BoolGries`. Bien entendu, il faut aussi donner à Z/EVES les axiomes qui décrivent le comportement de ces nouvelles constantes. De cette façon, nous pouvons obtenir une logique qui n'est plus simplement à deux valeurs (`vrai` et `faux`), mais elle peut aussi avoir d'autres possibilités.

Par exemple, si nous souhaitons introduire deux nouvelles constantes appellées `peut-être1` et `peut-être2`, alors il faut d'abord déclarer ces constantes globales pour tout le type. Ceci se réalise de la manière suivante :

```
\begin{axdef}
\peutEtre1: BoolGries\\
\peutEtre2: BoolGries
\end{axdef}
```

Ensuite, nous devons décrire le comportement de ces constantes soit avec les constantes déjà présentes (comme la constante `faux`), soit avec les opérateurs déjà présents (comme la constante `vrai`). Nous pourrions décrire une partie du comportement de ces constantes en donnant l'axiome suivant :

```
\begin{theorem}{rule peutEtre1Negation}
\forall p: BoolGries @ \non \peutEtre1 = \peutEtre2
\end{theorem}
```

Attention : il faut être *extrêmement* vigilant lorsque nous définissons de nouvelles constantes ! Gardons toujours à l'esprit de vérifier, après l'implantation des nouvelles constantes, si tous les axiomes de base (qui concernent de près ou de loin ces constantes) sont toujours valides, car, en implantant `peut-être1` et `peut-être2`, nous venons peut-être de rendre invalide des axiomes et des théorèmes de notre banque de théorèmes ! Heureusement, tel

n'est pas le cas avec ces deux nouvelles constantes (on peut démontrer que toutes les constantes du type `BoolGries` réunies dans ce cas-ci forment un treillis complet), mais cela aurait pu se produire.

En effet, si nous aurions implanté plutôt la constante `peut-être` au lieu de `peut-être1` et `peut-être2` et si nous aurions défini cette constante à l'aide de cet axiome :

```
\begin{theorem}{rule peutEtreNegation}
\forall p: BoolGries @ \non \peutEtre = \peutEtre
\end{theorem}
```

alors ceci aurait créer un effet étrange à notre logique. Cet axiome aurait rendu notre logique *univaluée*. Donc, nous n'aurions pas étendu, mais plutôt restreint notre algèbre.

Il est facile de s'en convaincre. Tout d'abord, pensons que p (dans l'axiome du tiers exclu) prend la valeur `peut-être`. Dès cet instant, nous obtenons le cas particulier suivant :

$$\texttt{peut-être} \vee \neg\texttt{peut-être} = \texttt{vrai}.$$

En utilisant l'axiome *peutEtreNegation*, on obtient le résultat suivant :

$$\texttt{peut-être} \vee \texttt{peut-être} = \texttt{vrai}.$$

Ce qui se réduit avec l'idempotence de la disjonction en :

$$\texttt{peut-être} = \texttt{vrai}.$$

Ensuite, imaginons que p (dans le théorème de la contradiction) prend la valeur `peut-être`. À ce moment-là, nous obtenons le cas particulier suivant :

$$\texttt{peut-être} \wedge \neg\texttt{peut-être} = \texttt{faux}.$$

En utilisant l'axiome *peutEtreNegation*, on obtient ceci :

$$\texttt{peut-être} \wedge \texttt{peut-être} = \texttt{faux}.$$

Ce qui se réduit avec l'idempotence de la conjonction en :

$$\texttt{peut-être} = \texttt{faux}.$$

Donc, on obtient :

$$\texttt{vrai} = \texttt{faux} = \texttt{peut-être}.$$

Ce qui permet de dire que notre logique est *univaluée*, elle est contradictoire. Ce qui n'était pas le résultat voulu au départ car nous voulions étendre notre logique. Nous n'avons réussi qu'à la restreindre par l'ajout de cette nouvelle constante.

C'est pour cela qu'il faut faire *très* attention lorsque nous tentons d'étendre cette algèbre à l'aide de nouvelles constantes.

Une dernière possibilité pour étendre la logique proposée ici est de donner de nouveaux axiomes valides pour les opérateurs déjà présents. Si nous faisons cela, nous devons faire attention de ne pas entrer en conflit avec les axiomes déjà définis pour ces opérateurs, car le but est de conserver tout de même le système d'axiome initial pour le rendre plus puissant en l'enrichissant de nouvelles connaissances.

Grâce à ceci, nous venons de spécialiser le type `BoolGries` aux besoins d'un utilisateur particulier. Donc, nous venons de créer une nouvelle théorie qui est utile pour une utilisation différente de celle de cette recherche du type `BoolGries`.

Chapitre 6

Exemples déjà testés

Ce chapitre invite l'utilisateur à réaliser des démonstrations faciles et pratiques avec Z/EVES. À vrai dire, ce chapitre montre comment utiliser la banque d'exemples créée dans cette recherche qui peut être récupérée rapidement par l'utilisateur. Chaque fichier donné ci-dessous contient un exemple d'une démonstration d'un théorème[1]. La résolution de ces exemples est donnée seulement pour le mode graphique de Z/EVES car il permet d'apprendre rapidement la démonstration de théorèmes. Aussi, le temps de réalisation de la démonstration est tenu en compte ici car si nous devons charger plusieurs théorèmes dans Z/EVES avant de pouvoir débuter la démonstration, alors le mode textuel prendrait *beaucoup* plus de temps (le mode textuel réalise les preuves de chaque théorème[2]).

Voici les fichiers qui seront présentés dans ce chapitre[3] :

- *ExTheoCourtLogProp.tex*
- *ExTheoMoyenLogProp.tex*
- *ExTheoLongLogProp.tex*
- *SetTheoryZEVES.zed*

[1]Exception faite du fichier *SetTheoryZEVES.zed* qui contient plusieurs exemples.

[2]Nous pouvons tout de même réaliser ces exemples dans le mode textuel, il s'agit simplement d'adapter les commandes qui seront présentées dans ce chapitre pour les rendre compatibles dans le mode textuel.

[3]Les fichiers ne contiennent que les axiomes et les théorèmes pertinents pour la démonstration.

6.1 Exemples sur la logique propositionnelle

Premier exemple : *ExTheoCourtLogProp.tex*

Le premier exemple est celui du fichier *ExTheoCourtLogProp.tex*. C'est un exemple simple permettant de se « faire les dents » sur la démonstration de théorèmes avec Z/EVES.

Le théorème que nous souhaitons démontrer dans cet exemple est le théorème :

$$\forall p, q : \texttt{BoolGries} \bullet p \vee (q \vee p \equiv p) \equiv p \vee q \equiv p.$$

Tout d'abord, nous allons charger le fichier *ExTheoCourtLogProp.tex* dans le mode graphique de Z/EVES. Pour réaliser le chargement de cet exemple dans le mode graphique, nous utilisons la méthode expliquée à la section 5.2.9.

Après avoir chargé (sans oublier d'avoir vérifié syntaxiquement tous les théorèmes), rendons-nous au dernier théorème : le théorème *TheoremeADemontrer*. Ensuite, cliquons sur le bouton droit de la souris sur le théorème et sélectionnons dans le menu flottant : *Show proof*.

Nous avons accédé à la fenêtre *Proof* de ce théorème. Maintenant, nous devons le prouver. Il existe plusieurs façons de prouver ce théorème. Nous en exposons deux ici.

Première preuve du théorème

Nous proposons ici une preuve presque automatique. Nous allons utiliser nos règles de réécriture et nous ne poserons pas de contraintes à Z/EVES pour la démonstration.

Exécutons tout d'abord la commande *prove*. Ceci permet à Z/EVES d'appliquer, de façon répétitive, la commande *rewrite* jusqu'à ce qu'elle n'effectue plus de transformations (voir la description de la commande, section 5.2.6.7). Nous obtenons le but courant suivant :

$p \in \texttt{BoolGries} \wedge q \in \texttt{BoolGries} \Rightarrow p$ *equiv* p *ou* $(p$ *ou* $q) = p$ *equiv* p *ou* q.

En examinant ce but, nous nous rendons compte que Z/EVES n'est pas capable de conclure que p *ou* $(p$ *ou* $q)$ donne p *ou* q. Il faut donc l'aider. Pour

cela, nous avons dans notre banque de théorèmes un théorème sur l'associativité de la disjonction. Utilisons-le en entrant manuellement la commande dans le *mini-editor* :

use ouAssociativite2[q := p, r := q].

Si tout s'est bien déroulé, nous devrions avoir ajouté, dans les hypothèses du but courant, le théorème *ouAssociativite2* avec les substitutions demandées.

Finalement, nous demandons à Z/EVES de compléter la preuve avec la commande : *prove*. Z/EVES retourne le but courant *true*. Voilà, le théorème est démontré !

Deuxième preuve du théorème

Cette preuve est plus longue que la première, mais elle permet de réaliser toutes les étapes de la démonstration manuellement.

Avant de débuter *Il faut connaître une subtilité de la fenêtre* Proof *de Z/EVES pour épargner du temps dans les démonstrations. La petite fenêtre* Formula *(la partie basse de la fenêtre* Proof*) permet de sélectionner une partie d'un théorème seulement et, si nous cliquons sur le bouton de droite de la souris, nous pouvons appliquer des commandes déjà programmées dans Z/EVES (ces commandes sont spécifiques à chaque expression ou prédicat sélectionné). En tant que tel, ceci n'est qu'un raccourci pour réaliser les preuves, nous pouvons réaliser n'importe quelle preuve sans jamais nous en soucier, mais, quelque fois, il est préférable de miser sur la rapidité de la démonstration. Cette subtilité renferme beaucoup de commandes utiles pour les démonstrations.*

Démonstration Il faut appliquer, dans l'ordre, les commandes suivantes pour réussir à démontrer le théorème :

1. *apply DistOuSurEquiv2 to expression p ou (q ou p equiv p)*
2. *simplify*
3. *apply ouIdempotence to expression p ou p*
4. *simplify*

5. *apply ouCommutativite to expression p ou (q ou p)*
6. *simplify*
7. *apply ouAssociativite to expression q ou p ou p*
8. *simplify*
9. *apply ouIdempotence to expression p ou p*
10. *simplify*
11. *apply ouCommutativite to expression q ou p*
12. *rewrite*

Nous utilisons la commande *rewrite* comme dernière commande simplement parce que cette commande réalise un raisonnement sur les prédicats conditionnels plus puissant que celui de la commande *simplify*. En effet, si nous utilisons la commande *simplify* alors cela ne réduit pas le but à *true*, mais, au contraire, le transforme en un prédicat désagréable à démontrer.

Deuxième exemple : *ExTheoMoyenLogProp.tex*

Essayons de démontrer un théorème plus difficile que le précédent : le théorème à démontrer du fichier *ExTheoMoyenLogProp.tex*. C'est le théorème suivant que nous voulons donc démontrer :

$$\forall p, q, r : \texttt{BoolGries} \bullet p \Rightarrow q \wedge r \equiv (p \Rightarrow q) \wedge (p \Rightarrow r).$$

Ce théorème se démontre facilement à l'aide de notre banque de théorèmes. Tout d'abord, chargeons le fichier *ExTheoMoyenLogProp.tex* dans le mode graphique. Ensuite, accédons à la fenêtre *Proof* du théorème *TheoremeADemontrer*. Pour réaliser la preuve de ce théorème, il suffit que nous entrions les commandes suivantes :

1. *use DefAltDeImplication1[q := q et r]*
2. *use DefAltDeImplication1*
3. *use DefAltDeImplication1[q := r]*
4. *use DistOuSurEt1[p := non p]*
5. *with disabled (implicationDef, ouAssociativite, ouCommutativite, etCommutativite, etAssociativite, DeMorgan1, DeMorgan2, DistOuSurEt1, DistEtSurOu1, RegleDOr) prove*

Ce qui revient à effectuer la preuve suivante :

$\forall p, q, r : \texttt{BoolGries} \bullet p \Rightarrow q \wedge r \equiv (p \Rightarrow q) \wedge (p \Rightarrow r)$
$=$ $\langle$ DefAltDeImplication1 (3.72), avec $q := q \wedge r$ $\rangle$
$\forall p, q, r : \texttt{BoolGries} \bullet \neg p \vee (q \wedge r) \equiv (p \Rightarrow q) \wedge (p \Rightarrow r)$
$=$ $\langle$ DefAltDeImplication1 (3.72) $\rangle$
$\forall p, q, r : \texttt{BoolGries} \bullet \neg p \vee (q \wedge r) \equiv (\neg p \vee q) \wedge (p \Rightarrow r)$
$=$ $\langle$ DefAltDeImplication1 (3.72), avec $q := r$ $\rangle$
$\forall p, q, r : \texttt{BoolGries} \bullet \neg p \vee (q \wedge r) \equiv (\neg p \vee q) \wedge (\neg p \vee r)$
- DistOuSurEt1, avec $p := \neg p_{\square}$

Donc, nous avons démontré le théorème.

Troisième exemple : *ExTheoLongLogProp.tex*

Le troisième théorème qui sera démontré est un théorème tiré de la banque de théorèmes que nous avons réalisé dans cette recherche. C'est le théorème :

$$\forall p, q : \texttt{BoolGries} \bullet p \vee q \Rightarrow p \wedge q \equiv p \equiv q$$

Il a été choisi car c'est un théorème qui possède une preuve plutôt complexe dans Z/EVES et ce théorème a une signification intéressante. En effet, ce théorème donne un lien possible pour créer une équivalence grâce aux opérateurs $\Rightarrow$, $\vee$ et $\wedge$.

Ce théorème et sa démonstration sont présentés dans le fichier *ExTheoLongLogProp.tex*. Nous donnons, dans l'ordre, les commandes à appliquer dans le mode graphique pour permettre de démontrer ce théorème :

1. *apply implicationDef to expression p ou q implique p et q*
2. *simplify*
3. *use DistOuSurEt1[p := p ou q, q := p, r := q]*
4. *apply ouAssociativite to expression p ou q ou p*
5. *simplify*
6. *apply ouCommutativite to expression q ou p*
7. *simplify*
8. *use lemme14*
9. *apply ouAssociativite to expression p ou q ou q*
10. *simplify*

11. *apply ouIdempotence to expression q ou q*
12. *simplify*
13. *apply etIdempotence to expression p ou q et (p ou q)*
14. *simplify*
15. *apply RegleDOr to expression p et q*
16. *simplify*
17. *apply equivCommutativite1 to expression p ou q equiv (p equiv q equiv p ou q)*
18. *simplify*
19. *use equivAssociativite1[p := p equiv q, q := p ou q, r := p ou q]*
20. *apply equivIdentite1 to expression p ou q equiv p ou q*
21. *simplify*
22. *with disabled (ouAssociativite, ouCommutativite) prove*

La démonstration est plutôt complexe, mais elle est donnée pour montrer que Z/EVES réussit bien à gérer les démonstrations longues.

6.2 Exemples sur la théorie des ensembles

Tout au long de cette recherche, nous nous sommes attardés à la démonstration de théorèmes selon la logique de Gries & Schneider pour la logique propositionnelle. Il existe aussi d'autres théories intéressantes et nous avons tenté d'en explorer quelques facettes avec Z/EVES.

Nous nous sommes intéressés à la théorie des ensembles. Cette théorie est déjà très bien implantée dans Z/EVES alors nous avons réutilisé les opérateurs déjà présents dans le démonstrateur. Notre but était de réaliser des démonstrations de théorèmes de la théorie des ensembles de Gries & Schneider, mais avec les axiomes de base de Z/EVES. Trois opérateurs étaient inconnus du démonstrateur, il a fallu les implanter. Ces opérateurs sont : le surensemble ($\supseteq$), le suremsemble propre ($\supset$) et le complément ($\sim$). Pour les implanter, nous n'avons donné que leur définition selon les opérateurs que Z/EVES connaissait déjà.

Voilà la syntaxe ajoutée pour notre théorie des ensembles :

```
\syndef{\surensemble}{inrel}{surensemble}
\syndef{\surensembleP}{inrel}{surensembleP}

\begin{gendef}[X]
\_ \surensemble \_, \_ \surensembleP \_: \power X \rel
\power X \\
\complement: \power X \fun \power X
\where
\Label{rule surensembleDef} \forall S, T: \power X @ T
\surensemble S \iff S \subseteq T \\
\Label{rule surensembleProprеDef} \forall S, T: \power X @ T
\surensembleP S \iff S \subset T \\
\Label{rule complementDef} \forall S: \power X; v: X @ v \in
\complement S \iff v \in X \land v \notin S
\end{gendef}
```

Ceci statue simplement que le *surensemble* est une relation infixée entre deux arguments de type $\mathbb{P}\ X$[4] et que cette relation possède la définition suivante :

$$\forall S, T : \mathbb{P}\ X \bullet T \supseteq S \equiv S \subseteq T.$$

De plus, cela affirme que l'opérateur *surensemble propre* est une relation infixée entre deux arguments de type $\mathbb{P}\ X$. Voilà sa définition :

$$\forall S, T : \mathbb{P}\ X \bullet T \supset S \equiv S \subset T.$$

Les dernières lignes donnent la définition de l'opérateur *complément* qui est une fonction de $\mathbb{P}\ X$ dans $\mathbb{P}\ X$. Sa définition est celle-ci :

$$\forall S : \mathbb{P}\ X ; v : X \bullet v \in {\sim}S \equiv v \in X \land v \notin S.$$

Les définitions que nous donnons des trois nouveaux opérateurs sont données en fonction des opérateurs de base de Z/EVES.

Tout ceci a permis de créer une banque d'exemples de théorèmes que nous pouvons utiliser dans le démonstrateur. Cette banque d'exemples est disponible dans le fichier *SetTheoryZEVES.zed*. Nous pouvons y accéder de la même manière que celle que nous avons utilisée pour notre algèbre booléenne dans la section 5.2.9. Nous pouvons donc utiliser facilement des démonstrations de cette banque d'exemples et même ajouter des théorèmes de la théorie des ensembles grâce aux opérateurs que nous avons implantés.

[4] L'ensemble X est un paramètre formel générique.

La manipulation de ce fichier est la même que celle pour le fichier *AlgBoolZLaTeX.zed*. Quelques petites choses diffèrent par contre : dans le fichier de la théorie des ensembles les démonstrations sont réalisées grâce aux théorèmes déjà présents dans notre démonstrateur[5]. De plus, le type `BoolGries` n'est plus utilisé pour soumettre les théorèmes à Z/EVES et nous utilisons maintenant un paramètre formel générique (nous l'avons appelé X dans cette recherche) qui permet de rendre les théorèmes portatifs à tous les types que nous désirons. Nous n'avons qu'à préciser, lors de l'utilisation d'un théorème dans une preuve d'un autre théorème, que le paramètre effectif générique sera le type utilisé à l'intérieur de notre théorème[6].

Avec tout ceci, nous avons une bonne banque d'exemples que nous pouvons utiliser et, de plus, nous pouvons la modifier facilement.

[5]Les théorèmes présents dans Z/EVES sont disponibles dans le *Mathematical Toolkit* [17].

[6]Normalement, ceci ne sera pas nécessaire car Z/EVES possède un très bon niveau d'automatisme dans les preuves de la théorie des ensembles avec ses propres axiomes.

Chapitre 7

Conclusion

En résumé, les démonstrateurs automatiques de théorèmes de la logique classique de premier ordre sont extrêmement intéressants pour apprendre la démonstration de théorèmes. Nous avons réalisé dans cette recherche un tour d'horizon des différents démonstrateurs automatiques de théorèmes existants et nous avons approfondi et testé un de ceux-ci : Z/EVES version 2.1. Nous lui avons implanté la logique équationnelle de Gries & Schneider pour vérifier son comportement avec cette logique. Le démonstrateur a très bien répondu à cette implantation et nous avons même créé une ébauche de la théorie des ensembles. Nous avons donc réussi à implanter la logique que nous voulions dans Z/EVES dans le but de favoriser un apprentissage rapide de la démonstration de théorèmes.

Dans ce rapport, on ne s'est intéressé qu'aux démonstrateurs automatiques de théorèmes de la logique classique de premier ordre. Pourtant, il y a d'autres démonstrateurs fonctionnant sur d'autres logiques qui sont aussi intéressants. Prenons le cas d'OSCAR : un démonstrateur automatique de théorèmes qui a été développé à l'université de l'Arizona. C'est un démonstrateur automatique de théorèmes qui raisonne sur des cas généraux avec, en son sein, un agent rationnel de déduction. C'est un démonstrateur expérimental très puissant qui permet d'aider dans les recherches en philosophie.

Bibliographie

[1] *Le Nouveau Petit Robert*. Maury Imprimeur S.A., 1993.

[2] ORA Canada. *ORA Canada : EVES Verification System*. ORA Canada, 1999. `http://www.ora.on.ca/eves.html`.

[3] Office de la langue française. *Le grand dictionnaire terminologique*. Adresse URL : `http://www.granddictionnaire.com/_fs_global_01.htm`, 2001.

[4] Jules Desharnais. *Notes de cours du cours de logique et structures discrètes*. Département d'informatique, Université Laval, 2000.

[5] David Gries et Fred B. Schneider. *A Logical Approch to Discrete Math*. Springer, 1994.

[6] Matt Kaufmann et J. Strother Moore. *ACL2 version 2.5*. Université du Texas à Austin, 2000. `http://www.cs.utexas.edu/users/moore/acl2/acl2-doc.html`.

[7] Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, 1996.

[8] Mike Gordon. *The HOL System Description*. Cambridge Research Center of SRI International, février 2001.

[9] Mike Gordon. *The HOL System Reference*. Cambridge Research Center of SRI International, février 2001.

[10] Mike Gordon. *The HOL System Tutorial*. Cambridge Research Center of SRI International, février 2001.

[11] John Harrison. *HOL Light*. Université de Cambridge, 1998. `http://www.cl.cam.ac.uk/users/jrh/hol-light/`.

[12] John Harrison. *The HOL Light Manual (1.1)*. University of Cambridge Computer Laboratory, avril 2000.

[13] Mathematics and Computer Science Division. *Otter : An Automated Deduction System*. Argonne National Laboratory, 2001. `http://www-unix.mcs.anl.gov/AR/otter/`.

[14] William W. McCune. *Otter 3.0 Reference Manual and Guide*. Argonne National Laboratory, 9700 South Cass Avenue, janvier 1994.

[15] Irwin Meisels. *Software Manual for Windows Z/EVES Version 2.1*. ORA Canada, One Nicholas Street, Suite 1208, Ottawa (Ontario), K1N 7B7, juillet 2000.

[16] Irwin Meisels and Mark Saaltink. *The Z/EVES Reference Manual (for Version 1.5)*. ORA Canada, One Nicholas Street, Suite 1208, Ottawa (Ontario), K1N 7B7, septembre 1997.

[17] Mark Saaltink. *The Z/EVES 2.0 Mathematical Toolkit*. ORA Canada, One Nicholas Street, Suite 1208, Ottawa (Ontario), K1N 7B7, octobre 1999.

[18] Mark Saaltink. *The Z/EVES 2.0 User's Guide*. ORA Canada, One Nicholas Street, Suite 1208, Ottawa (Ontario), K1N 7B7, octobre 1999.

[19] J.M. Spivey. *The Z Notation : A Reference Manual, Second Edition*. Prentice Hall, 1992.

[20] Daryl Stewart. *Automated Reasoning Group HOL page*. Université de Cambridge, 2000. `http://www.cl.cam.ac.uk/Research/HVG/HOL/`.

[21] Freek Wiedijk. *Overview of systems implementing "mathematics in the computer"*. Liste disponible sur internet à l'adresse `http://www.cs.kun.nl/~freek/digimath/index.html`, 2001.

Troisième partie

Annexes

Annexe A

Annexe - Différences entre 80 démonstrateurs automatiques de théorèmes grâce aux critères de comparaison énoncés à la section 3.1

A.1 Plateformes, langage de base et puissance maximale

Nom	plateforme	Langage de base	Puissance maximale
3TaP	Unix	Prolog	Logique de premier ordre classique
ACL2	Unix, Windows 98, Macintosh	Common Lisp	N/A
Analytica	Unix, Windows, Macintosh	Mathematica	Logique classique
Bertrand	Macintosh	N/A	Logique de premier ordre classique
Bliksem	Unix, Windows	C	Logique de premier ordre classique
CLIN: CLIN-S	N/A	N/A	Logique de premier ordre classique
RDL	N/A	SICStus Prolog	Logique de premier ordre
Coq	Unix, Windows 95/98/NT	Objective CAML	Logique constructive
CtCoq	Linux, Sun OS Dec Alpha	N/A	Logique constructive
Pcoq	Unix, Windows 95/98/NT	Java	Logique classique
Discount	N/A	N/A	N/A
Doris	N/A	Prolog	Logique de premier ordre classique
DTP	Unix, Macintosh	Common Lisp	Logique de premier ordre
E	Unix	C	Logique classique
EQP	Unix	C	Logique de premier ordre classique
FDPLL	Unix	Eclipse Prolog	Logique de premier ordre classique

FIG. A.1 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant les critères de comparaison : plateforme, langage de base et puissance maximale.

Nom	plateforme	Langage de base	Puissance maximale
FINDER	Unix	C	Logique de premier ordre classique
Frapps	Unix	Common Lisp	Logique de premier ordre
FT	N/A	C	Logique de premier ordre constructive
GANDALF	Unix, DOS, Windows	Scheme	Logique de premier ordre
Getfol	N/A	Common Lisp	Logique de premier ordre classique
HOL98	Unix, Windows NT	Standard ML	Logique classique
HOL Light	Unix	CAML Light	Logique classique
HR	N/A	Prolog	Logique classique
ILF	Unix	Prolog	Logique classique
IMPS	Linux, Solaris	Common Lisp	Logique classique
INKA	Linux	Lucida Common Lisp	Logique de premier ordre intuitionniste
IPR	Unix	Common Lisp	Logique classique
Isabelle	Linux, Solaris	Standard ML	Logique classique et intuitionniste
Isabelle: Isar	Linux, Solaris	Standard ML	Logique classique et intuitionniste
Keim	Unix	Common Lisp, CLOS	Logique classique
Kimba	Unix, Windows 95/NT	Oz	Logique de premier ordre classique

FIG. A.2 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant les critères de comparaison : plateforme, langage de base et puissance maximale (suite).

Nom	plateforme	Langage de base	Puissance maximale
Kripke	Unix, Windows	Pascal	Logique de premier ordre constructive
Larch: LP	Linux, Solaris	N/A	Logique de premier ordre classique
LeanTaP	Unix	Prolog	Logique de premier ordre classique
Lego	Unix	Standard ML	Logique constructive
llprover	Unix	SICStus Prolog	Logique de premier ordre
MACE	Unix	C	Logique de premier ordre classique
Maude	Unix	N/A	Logique classique
METEOR	Unix	N/A	Logique de premier ordre classique
Minlog	Unix, Windows	Scheme	Logique constructive
MINLOG	Unix	C	Logique intuitionniste et propositionnelle
MVL	Unix	Common Lisp	Logique de premier ordre, ATMS
Nqthm (the Boyer-Moore prover)	Unix, Macintosh	Common Lisp	Logique classique
Nuprl	Unix	Common Lisp, ML	Logique constructive
Oleg	N/A	ML	Logique constructive
Oscar	Windows, Unix	Common Lisp	Logique de premier ordre
OSHL	N/A	Eclipse Prolog	Logique de premier ordre classique

FIG. A.3 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant les critères de comparaison : plateforme, langage de base et puissance maximale (suite).

Nom	plateforme	Langage de base	Puissance maximale
Otter	Unix, Macintosh, DOS, Windows	C	Logique de premier ordre classique
Oyster, Clam	Unix	Prolog	Logique constructive
PhoX	Unix (bientôt d'autres plateformes)	Objective CAML	Logique constructive
Plastic	Unix	Haskell	Logique constructive
ProCom	Unix	Eclipse Prolog	Logique classique
Proof General	Unix	Emacs Lisp	Logique classique et constructive
ProofPower	N/A	Standard ML	Logique classique
PROTEIN	Unix	Eclipse Prolog	Logique de premier ordre classique
Prover, NP-Tools	Unix, Windows NT	C	Logique classique
PTTP	Unix	Common Lisp ou Prolog	Logique de premier ordre
Purr	Unix	N/A	Logique classique
PVS	Unix	N/A	Logique classique
REDLOG	Unix, DOS, Windows, Macintosh	N/A	Logique de premier ordre
RRL	Unix	Zeta Lisp	Logique de premier ordre classique
Satchmo	Unix	Prolog	Logique de premier ordre classique
Saturate	Linux, Solaris	Prolog	Logique de premier ordre classique

FIG. A.4 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant les critères de comparaison : plateforme, langage de base et puissance maximale (suite).

Nom	plateforme	Langage de base	Puissance maximale
SCOTT (combinaison d'Otter et de FINDER)	Unix	C	Logique de premier ordre classique
SEM	Unix	N/A	Logique de premier ordre
SETHEO	Unix	N/A	Logique de premier ordre classique
Simplify	Unix, Windows 95/98/NT	Modula-3	Logique classique
Snark	N/A	Common Lisp	Logique de premier ordre classique
SPASS	Linux, Solaris, Windows 95/98/NT	N/A	Logique de premier ordre classique
SPRFN	Unix	Prolog	Logique de premier ordre classique
SPTHEO	Unix	N/A	Logique de premier ordre classique
Theorema	Unix, Windows, Macintosh	Mathematica	Logique classique
TPS	Unix	Common Lisp	Logique classique
Typelab	N/A	Common Lisp	Logique constructive
Vampire	Linux, Solaris	N/A	Logique de premier ordre classique
Waldmeister	Unix	C	Logique de premier ordre classique
Watson	Unix	Standard ML	Logique classique
Yarrow	Unix	Haskell	Logique constructive
Z/EVES	Unix, Windows	Common Lisp	Logique classique

FIG. A.5 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant les critères de comparaison : plateforme, langage de base et puissance maximale (suite).

A.2 Qualité de la bibliographie et type d'interaction

Nom	qualité de la bibliographie	Type d'interaction
3TaP	Très bonne (livre d'instruction de 190 pages contenant: le livre de l'utilisateur, la description de l'implantation de 3TaP et un document relatant l'historique de 3TaP)	Script
ACL2	Excellente (environ 900 pages)	Dialogue
Analytica	Mauvaise (presque inexistante)	Dialogue
Bertrand	Très basse	Éditeur
Bliksem	Mauvaise (le manuel d'instruction n'est pas encore terminé)	Script
CLIN: CLIN-S	Très basse	Script
RDL	Très basse	Script
Coq	Bonne	Dialogue
CtCoq	Bonne mais incomplète	Dialogue
Pcoq	Bonne	Dialogue
Discount	Inexistante	N/A
Doris	Inexistante	Batch
DTP	Mauvaise (non mise à jour)	N/A
E	Très basse (encore en développement)	N/A
EQP	Mauvaise	Script
FDPLL	Très basse	Script

FIG. A.6 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant les critères de comparaison : qualité de la bibliographie et type d'interaction.

Nom	qualité de la bibliographie	Type d'interaction
FINDER	bonne (manuel d'instruction de 77 pages)	Batch
Frapps	Mauvaise	N/A
FT	Très basse	Script
GANDALF	Très basse	Script
Getfol	Très bonne	Dialogue
HOL98	Excellente (plus de 800 pages de bibliographie officielle)	Dialogue
HOL Light	Très bonne (environ une centaine de pages pour le livre d'instruction)	Dialogue
HR	Bonne	Batch
ILF	Bonne	Dialogue
IMPS	Très bonne (livre d'instruction de 300 pages)	Dialogue
INKA	Bonne, mais écrite seulement en Allemand	Éditeur
IPR	Très basse	N/A
Isabelle	Excellente (très volumineuse)	Dialogue
Isabelle: Isar	Très bonne	Éditeur
Keim	Bonne	Librairie
Kimba	Non trouvée	Script

FIG. A.7 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant les critères de comparaison : qualité de la bibliographie et type d'interaction (suite).

Nom	qualité de la bibliographie	Type d'interaction
Kripke	Très basse	Script
Larch: LP	Très bonne	Dialogue
LeanTaP	Très basse	Script
Lego	Bonne (environ 70 pages)	Dialogue
llprover	Très basse (environ 12 pages)	Script
MACE	Moyenne	Script
Maude	Très bonne	N/A
METEOR	Moyenne (abondante mais non structurée)	Script
Minlog	Bonne (environ 50 pages)	Dialogue
MINLOG	Basse	N/A
MVL	Très basse	N/A
Nqthm (the Boyer-Moore prover)	Bonne	Dialogue
Nuprl	Très bonne (abondante et pertinante)	Dialogue
Oleg	Très basse (presque inexistante)	Dialogue
Oscar	Très bonne (volumineuse, complète et intéressante)	Dialogue
OSHL	Très basse	Script

FIG. A.8 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant les critères de comparaison : qualité de la bibliographie et type d'interaction (suite).

Nom	qualité de la bibliographie	Type d'interaction
Otter	Très bonne (un livre d'instruction complet avec plusieurs livre sur la manipulation de Otter)	Script
Oyster, Clam	Bonne	Dialogue
PhoX	Bonne, mais incomplête	Script
Plastic	Non trouvée	Dialogue
ProCom	Très basse	Dialogue
Proof General	Très bonne (complête et intéressante)	Éditeur
ProofPower	Non trouvée	Dialogue
PROTEIN	Bonne	Script
Prover, NP-Tools	Non trouvée (c'est un produit à acheter)	Librairie
PTTP	Très basse	N/A
Purr	Bonne, mais le logiciel est encore instable	Script
PVS	Très bonne (beaucoup de documentation accessible)	Dialogue
REDLOG	Bonne (bien structurée)	N/A
RRL	Moyenne	Script
Satchmo	Très basse	Script
Saturate	Très bonne (complête et intéressante)	Script

FIG. A.9 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant les critères de comparaison : qualité de la bibliographie et type d'interaction (suite).

Nom	qualité de la bibliographie	Type d'interaction
SCOTT (combinaison d'Otter et de FINDER)	Très basse (sur SCOTT seulement)	N/A
SEM	Très basse	N/A
SETHEO	Bonne	Script
Simplify	Basse	Batch
Snark	Bonne	Script
SPASS	Bonne	Script
SPRFN	Basse (environ une vingtaine de pages de documentation)	Script
SPTHEO	Non trouvée	Script
Theorema	Non trouvée	Dialogue
TPS	Bonne (environ 60 pages)	Dialogue
Typelab	Non donnée	Dialogue
Vampire	Non trouvée	Script
Waldmeister	Non donnée	Script
Watson	Très bonne (complête et intéressante)	Dialogue
Yarrow	Bonne (complete)	Dialogue
Z/EVES	Très bonne (livre d'instruction volumineux)	Dialogue

FIG. A.10 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant les critères de comparaison : qualité de la bibliographie et type d'interaction (suite).

A.3 Coût de l'implantation

Nom	Coût de l'implantation
3TaP	Un compilateur Prolog
ACL2	Un compilateur Common Lisp
Analytica	Un logiciel Mathématica (environ 895$ pour une license de professeur)
Bertrand	Rien (0$)
Bliksem	Un compilateur C
CLIN: CLIN-S	Un compilateur Quintus Prolog
RDL	Un logiciel SICStus Prolog 3.8 au minimum
Coq	Les logiciels Objective Calm (gratuit) et Calmp4 version 3.01 (gratuit)
CtCoq	Il faut un logiciel de Coq
Pcoq	Il faut un logiciel de Coq
Discount	Logiciel non disponible
Doris	le logiciel Mathweb-sb et SISCtus Prolog
DTP	Un compilateur Common Lisp Common Lisp
E	Un compilateur C
EQP	Un compilateur C
FDPLL	Un logiciel Eclipse Prolog (gratuit)

FIG. A.11 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant le critère de comparaison : coût de l'implantation.

Nom	Coût de l'implantation
FINDER	Un compilateur C
Frapps	Un compilateur Common Lisp
FT	Un compilateur C
GANDALF	Soit rien (0$) si seulement l'exécutable ou un compilateur C si vous voulez le code source
Getfol	Rien (0$)
HOL98	Le logiciel Moscow ML 2.00 (gratuit)
HOL Light	Le logiciel Moscow ML 2.00 (gratuit)
HR	Logiciel non disponible
ILF	Un démonstrateur de théorèmes parmis les suivants: Discount, Protein, Setheo et Spass
IMPS	Un compilateur Common Lisp et le logiciel Emacs
INKA	Un compilateur Allegro Lisp (gratuit)
IPR	Un compilateur Common Lisp
Isabelle	Les logiciels bash (gratuit), Perl (gratuit) et Standard ML (gratuit)
Isabelle: Isar	Les logiciels Emacs (gratuit) et Isabelle (gratuit)
Keim	Un compilateur Common Lisp et CLOS
Kimba	Le logiciel Mozart (gratuit)

FIG. A.12 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant le critère de comparaison : coût de l'implantation (suite).

Nom	Coût de l'implantation
Kripke	Logiciel non disponible
Larch: LP	Rien (0$)
LeanTaP	Un compilateur Prolog
Lego	Le logiciel Standard ML (gratuit)
llprover	Soit rien (gratuit) si on utilise la version sur le web ou un compilateur Prolog si vous voulez le code source (gratuit)
MACE	Rien (gratuit) car on ne peut que l'utiliser sur le web
Maude	Rien (gratuit)
METEOR	Logiciel non disponible
Minlog	Le logiciel Scheme (gratuit)
MINLOG	Un compilateur C
MVL	Un compilateur Common Lisp
Nqthm (the Boyer-Moore prover)	Un compilateur Common Lisp
Nuprl	Les logiciels ML et Common Lisp
Oleg	Logiciel non disponible
Oscar	Un compilateur LISP (le logiciel est gratuit seulement dans le cadre d'une utilisation académique ou de recherche)
OSHL	Rien (gratuit) car on ne peut que l'utiliser sur le web

FIG. A.13 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant le critère de comparaison : coût de l'implantation (suite).

Nom	Coût de l'implantation
Otter	Un compilateur C
Oyster, Clam	Un compilateur Prolog
PhoX	Le logiciel Objective Calm (gratuit)
Plastic	Un compilateur Haskell
ProCom	Un compilateur Prolog
Proof General	Le logiciel Emacs (gratuit)
ProofPower	Logiciel non disponible
PROTEIN	Un compilateur Prolog
Prover, NP-Tools	Non mentionné
PTTP	Un compilateur Prolog (gratuit) ou un compilateur Lisp (gratuit)
Purr	Logiciel non disponible
PVS	Les logiciels Emacs (gratuit), Tcl/Tk, LaTeX (gratuit)
REDLOG	Le logiciel REDUCE
RRL	Un compilateur Common Lisp
Satchmo	Un compilateur Prolog
Saturate	Un compilateur Prolog

FIG. A.14 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant le critère de comparaison : coût de l'implantation (suite).

Nom	Coût de l'implantation
SCOTT (combinaison d'Otter et de FINDER)	Un compilateur C
SEM	Non mentionné
SETHEO	Rien (gratuit)
Simplify	Un Java run-time system (Sun's JDK ou JRE) (gratuit)
Snark	Logiciel non disponible
SPASS	Rien (gratuit), mais le code source n'est pas fourni pour Windows
SPRFN	Un compilateur prolog
SPTHEO	Logiciel non disponible
Theorema	Logiciel non disponible, mais on peut utiliser une version incomplète sur le web
TPS	Un compilateur Common Lisp (gratuit)
Typelab	Logiciel non disponible
Vampire	Non mentionné
Waldmeister	Un compilateur C
Watson	Le logiciel Standard ML (gratuit)
Yarrow	Un compilateur Haskell
Z/EVES	Rien (0$)

FIG. A.15 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant le critère de comparaison : coût de l'implantation (suite).

A.4 Rapidité avec démonstration complexe et simple, convivialité et niveau d'intérêt

Nom	rapidité avec démonstration		Convivialité	Niveau d'intérêt
	complexe	simple		
3TaP	N/A	N/A	Bonne	*
ACL2	Lente et difficile	N/A	N/A	****
Analytica	N/A	N/A	N/A	*
Bertrand	N/A	N/A	Bonne	*
Bliksem	N/A	N/A	N/A	*
CLIN: CLIN-S	N/A	N/A	N/A	*
RDL	N/A	N/A	N/A	*
Coq	N/A	N/A	N/A	*
CtCoq	N/A	N/A	Bonne	***
Pcoq	N/A	N/A	Bonne	*
Discount	N/A	N/A	N/A	*
Doris	N/A	N/A	N/A	*
DTP	N/A	N/A	N/A	*
E	N/A	N/A	N/A	*
EQP	N/A	N/A	N/A	***
FDPLL	Lente et difficile	Lente et difficile	N/A	*

FIG. A.16 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant les critères de comparaison : rapidité avec démonstration complexe et simple, convivialité et niveau d'intérêt.

Nom	rapidité avec démonstration		Convivialité	Niveau d'intérêt
	complexe	simple		
FINDER	N/A	N/A	N/A	*
Frapps	N/A	N/A	N/A	*
FT	N/A	N/A	N/A	**
GANDALF	N/A	N/A	Bonne	**
Getfol	N/A	N/A	N/A	**
HOL98	N/A	N/A	N/A	****
HOL Light	N/A	N/A	N/A	**
HR	N/A	N/A	N/A	*
ILF	N/A	N/A	Bonne	*
IMPS	N/A	N/A	N/A	*
INKA	N/A	N/A	N/A	*
IPR	N/A	N/A	N/A	*
Isabelle	N/A	N/A	N/A	**
Isabelle: Isar	N/A	N/A	Très bonne	**
Keim	N/A	N/A	N/A	*
Kimba	N/A	N/A	N/A	**

FIG. A.17 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant les critères de comparaison : rapidité avec démonstration complexe et simple, convivialité et niveau d'intérêt (suite).

Nom	rapidité avec démonstration		Convivialité	Niveau d'intérêt
	complexe	simple		
Kripke	N/A	N/A	N/A	*
Larch: LP	rapide	N/A	N/A	*
LeanTaP	N/A	N/A	N/A	*
Lego	N/A	N/A	N/A	*
llprover	N/A	N/A	Moyenne	*
MACE	N/A	N/A	N/A	*
Maude	N/A	N/A	N/A	*
METEOR	N/A	N/A	N/A	*
Minlog	N/A	N/A	N/A	*
MINLOG	N/A	N/A	N/A	*
MVL	N/A	N/A	N/A	*
Nqthm (the Boyer-Moore prover)	N/A	N/A	N/A	*
Nuprl	N/A	N/A	N/A	*
Oleg	N/A	N/A	N/A	*
Oscar	N/A	N/A	N/A	****
OSHL	N/A	N/A	N/A	*

FIG. A.18 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant les critères de comparaison : rapidité avec démonstration complexe et simple, convivialité et niveau d'intérêt (suite).

Nom	rapidité avec démonstration		Convivialité	Niveau d'intérêt
	complexe	simple		
Otter	N/A	N/A	N/A	****
Oyster, Clam	N/A	N/A	N/A	*
PhoX	N/A	N/A	Très bonne	**
Plastic	N/A	N/A	N/A	*
ProCom	N/A	N/A	N/A	*
Proof General	N/A	N/A	Excellente	*
ProofPower	N/A	N/A	N/A	*
PROTEIN	N/A	N/A	N/A	*
Prover, NP-Tools	N/A	N/A	N/A	*
PTTP	moyen	Très rapide	N/A	*
Purr	N/A	N/A	N/A	*
PVS	N/A	N/A	Très bonne	*
REDLOG	N/A	N/A	N/A	*
RRL	N/A	N/A	N/A	*
Satchmo	N/A	N/A	N/A	*
Saturate	N/A	N/A	N/A	*

FIG. A.19 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant les critères de comparaison : rapidité avec démonstration complexe et simple, convivialité et niveau d'intérêt (suite).

Nom	rapidité avec démonstration		Convivialité	Niveau d'intérêt
	complexe	simple		
SCOTT (combinaison d'Otter et de FINDER)	rapide	rapide	N/A	**
SEM	N/A	N/A	N/A	*
SETHEO	N/A	N/A	N/A	*
Simplify	N/A	N/A	N/A	**
Snark	N/A	N/A	N/A	*
SPASS	N/A	N/A	Moyenne	***
SPRFN	N/A	N/A	N/A	*
SPTHEO	Bonne	Très rapide	N/A	*
Theorema	N/A	N/A	N/A	**
TPS	N/A	N/A	N/A	**
Typelab	N/A	N/A	N/A	*
Vampire	N/A	N/A	N/A	*
Waldmeister	N/A	N/A	N/A	*
Watson	N/A	N/A	N/A	*
Yarrow	N/A	N/A	N/A	*
Z/EVES	N/A	N/A	Très bonne	****

FIG. A.20 – Tableau sur les différences entre les démonstrateurs automatiques de théorèmes suivant les critères de comparaison : rapidité avec démonstration complexe et simple, convivialité et niveau d'intérêt (suite).

A.5 Adresses Internet

Nom	Adresse Internet
3TaP	http://i12www.ira.uka.de/~threetap/
ACL2	http://www.cs.utexas.edu/users/moore/acl2/acl2-doc.html
Analytica	http://www.cs.cmu.edu/~andrej/analytica/
Bertrand	http://www.humnet.ucla.edu/humnet/phil/grads/herzberg/Bertrand.html
Bliksem	http://www.mpi-sb.mpg.de/~bliksem/
CLIN: CLIN-S	http://www.cs.unc.edu/Research/mi/mi-provers.html
RDL	http://www.mrg.dist.unige.it/ccr/software.html
Coq	http://pauillac.inria.fr/coq/
CtCoq	http://www-sop.inria.fr/croap/ctcoq/ctcoq-eng.html
Pcoq	http://www-sop.inria.fr/lemme/pcoq/
Discount	http://www.uni-kl.de/AG-AvenhausMadlener/discount.html
Doris	http://www.coli.uni-sb.de/~bos/atp/doris-info.html
DTP	http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/reasonng/atp/systems/dtp/0.html
E	http://wwwjessen.informatik.tu-muenchen.de/~schulz/WORK/eprover.html
EQP	http://www.mcs.anl.gov/AR/eqp/
FDPLL	http://www.uni-koblenz.de/~peter/FDPLL/

FIG. A.21 – Adresses Internet utilisées pour l'analyse des différents démonstrateurs automatiques de théorèmes .

Nom	Adresse Internet
FINDER	http://arp.anu.edu.au/~jks/finder.html
Frapps	http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/reasonng/atp/systems/frapps/0.html
FT	http://www.sics.se/ps/ft.html
GANDALF	http://www.cs.chalmers.se/~tammet/gandalf/
Getfol	http://www.cs.unitn.it/~getfol/startgwigb.html
HOL98	http://www.cl.cam.ac.uk/Research/HVG/HOL/
HOL Light	http://www.cl.cam.ac.uk/users/jrh/hol-light/
HR	http://www.dai.ed.ac.uk/daidb/people/homes/simonco/research/hr/
ILF	http://www-irm.mathematik.hu-berlin.de/~ilf/
IMPS	http://imps.mcmaster.ca/
INKA	http://www.dfki.de/vse/systems/inka/
IPR	http://www.cs.wcu.edu/~shults/FTP/IPR/
Isabelle	http://www.cl.cam.ac.uk/Research/HVG/Isabelle/
Isabelle: Isar	http://isabelle.in.tum.de/Isar/
Keim	http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/reasonng/atp/systems/keim/0.html
Kimba	http://www.ags.uni-sb.de/~konrad/kimba.html

FIG. A.22 – Adresses Internet utilisées pour l'analyse des différents démonstrateurs automatiques de théorèmes (suite).

Nom	Adresse Internet
Kripke	http://citeseer.nj.nec.com/18535.html
Larch: LP	http://www.sds.lcs.mit.edu/spd/larch/LP/overview.html
LeanTaP	http://i12www.ira.uka.de/~posegga/leantap/
Lego	http://www.dcs.ed.ac.uk/home/lego/
llprover	http://bach.seg.kobe-u.ac.jp/llprover/
MACE	http://www.mcs.anl.gov/AR/mace/
Maude	http://maude.csl.sri.com/
METEOR	http://www.cs.duke.edu/~ola/meteor.html
Minlog	http://www.mathematik.uni-muenchen.de/~logik/minlog_e.html
MINLOG	http://arp.anu.edu.au/~jks/minlog.html
MVL	http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/reasonng/atp/systems/mvl/0.html
Nqthm (the Boyer-Moore prover)	ftp://ftp.cs.utexas.edu/pub/boyer/nqthm/index.html
Nuprl	http://simon.cs.cornell.edu/Info/Projects/NuPrl/nuprl.html
Oleg	http://www.dcs.ed.ac.uk/home/ctm/oleg/quiet/
Oscar	http://www.u.arizona.edu/~pollock/
OSHL	http://www.cs.unc.edu/~zhu/prover.html

FIG. A.23 – Adresses Internet utilisées pour l'analyse des différents démonstrateurs automatiques de théorèmes (suite).

Nom	Adresse Internet
Otter	http://www.mcs.anl.gov/AR/otter/
Oyster, Clam	http://dream.dai.ed.ac.uk/
PhoX	http://www.lama.univ-savoie.fr/~RAFFALLI/af2.html
Plastic	http://www.dur.ac.uk/CARG/plastic.html
ProCom	http://www.koralle.imn.htwk-leipzig.de/pub/ProCom/procom.html
Proof General	http://zermelo.dcs.ed.ac.uk/~proofgen/
ProofPower	http://www.lemma-one.com/ProofPower/index/
PROTEIN	http://www.uni-koblenz.de/ag-ki/Systems/PROTEIN/
Prover, NP-Tools	http://www.prover.com/
PTTP	http://www.ai.sri.com/~stickel/pttp.html
Purr	http://www-cad.eecs.berkeley.edu/~cm/publications/mt/
PVS	http://pvs.csl.sri.com/
REDLOG	http://www.fmi.uni-passau.de/~redlog/
RRL	file://cs.uiowa.edu/pub/hzhang/rrl/
Satchmo	http://www.pms.informatik.uni-muenchen.de/software/
Saturate	http://www.mpi-sb.mpg.de/SATURATE/

FIG. A.24 – Adresses Internet utilisées pour l'analyse des différents démonstrateurs automatiques de théorèmes (suite).

Nom	Adresse Internet
SCOTT (combinaison d'Otter et de FINDER)	http://arp.anu.edu.au/~jks/scott.html
SEM	http://www.cs.uiowa.edu/~hzhang/sem.html
SETHEO	http://wwwjessen.informatik.tu-muenchen.de/~setheo/
Simplify	http://research.compaq.com/SRC/esc/Simplify.html
Snark	http://www.ai.sri.com/~stickel/snark.html
SPASS	http://spass.mpi-sb.mpg.de/
SPRFN	http://www.cs.unc.edu/Research/mi/mi-provers.html
SPTHEO	http://wwwjessen.informatik.tu-muenchen.de/~suttner/sptheo.html
Theorema	http://www.theorema.org/
TPS	http://gtps.math.cmu.edu/tps.html
Typelab	http://www.informatik.uni-ulm.de/ki/typelab.html
Vampire	http://www.cs.man.ac.uk/~riazanoa/Vampire/
Waldmeister	http://www.uni-kl.de/AG-AvenhausMadlener/waldmeister.html
Watson	http://math.boisestate.edu/~holmes/proverpage.html
Yarrow	http://www.cs.kun.nl/~janz/yarrow/
Z/EVES	http://www.ora.on.ca/z-eves/welcome.html

FIG. A.25 – Adresses Internet utilisées pour l'analyse des différents démonstrateurs automatiques de théorèmes (suite).

Annexe B

Annexe - Axiomes implantés dans Z/EVES pour la création de l'algèbre booléenne

Définition du type BoolGries :

$[BoolGries]$

Définition des opérateurs pour la syntaxe :

syntax $\equiv$ *infun*3
syntax $\not\equiv$ *infun*3
syntax $\vee$ *infun*5
syntax $\wedge$ *infun*5
syntax $\Rightarrow$ *infun*4
syntax $\Leftarrow$ *infun*4
syntax $<$ *inrel*

Définitions des constantes et du typage des opérateurs :

$$
\begin{array}{|l}
\text{vrai} : BoolGries \\
\text{faux} : BoolGries \\
\_ \equiv \_ : BoolGries \times BoolGries \rightarrow BoolGries \\
\_ \not\equiv \_ : BoolGries \times BoolGries \rightarrow BoolGries \\
\_ \vee \_ : BoolGries \times BoolGries \rightarrow BoolGries \\
\_ \wedge \_ : BoolGries \times BoolGries \rightarrow BoolGries \\
\_ \Rightarrow \_ : BoolGries \times BoolGries \rightarrow BoolGries \\
\_ \Leftarrow \_ : BoolGries \times BoolGries \rightarrow BoolGries \\
\neg \ : BoolGries \rightarrow BoolGries
\end{array}
$$

$$
\begin{array}{|l}
\_ < \_ : BoolGries \leftrightarrow BoolGries \\
\hline
\langle\!\langle \text{enabled rule pluspetitqueDef} \rangle\!\rangle \\
\forall\, p, q : BoolGries \bullet p < q \Leftrightarrow p \Rightarrow q = \text{vrai}
\end{array}
$$

Axiomes de base

Axiomes de l'équivalence :

theorem rule equivAssociativite1
$\forall\, p, q, r : BoolGries \bullet p \equiv q \equiv r = p \equiv (q \equiv r)$

theorem rule equivAssociativite2
$\forall\, p, q, r : BoolGries \bullet p \equiv (q \equiv r) = p \equiv q \equiv r$

theorem rule equivCommutativite1
$\forall\, p, q : BoolGries \bullet p \equiv q = q \equiv p$

theorem equivCommutativite2
$\forall\, p, q : BoolGries \bullet p = q \equiv q \equiv p$

theorem rule equivCommutativite3
$\forall\, p, q : BoolGries \bullet p \equiv q \equiv q = p$

theorem rule equivIdentite1
$\forall\, p : BoolGries \bullet p \equiv p = \text{vrai}$

theorem rule equivIdentite2
$\forall\, p : BoolGries \bullet \text{vrai} = p \equiv p$

theorem rule equivIdentite3
$\forall\, p : BoolGries \bullet \text{vrai} \equiv p = p$

theorem equivIdentite4
$\forall\, p : BoolGries \bullet p = \text{vrai} \equiv p$

Axiome de faux :

theorem rule fauxDef
$\forall\, p : BoolGries \bullet \text{faux} = \neg\ \text{vrai}$

Axiomes de la négation :

theorem DistNegSurEgalite
$\forall\, p, q : BoolGries \bullet \neg\ p = q \Leftrightarrow p = \neg\ q$

theorem rule DistNegSurEquiv1
$\forall\, p, q : BoolGries \bullet \neg\ p \equiv q = \neg\ (p \equiv q)$

theorem rule DistNegSurEquiv2
$\forall\, p, q : BoolGries \bullet \neg\ (p \equiv q) = \neg\ p \equiv q$

Axiome de l'inéquivalence :

theorem rule nonequivDef
$\forall\, p, q : BoolGries \bullet p \not\equiv\ q = \neg\ p \equiv q$

Axiomes de la disjonction :

theorem rule ouCommutativite
$\forall\, p, q : BoolGries \bullet p \vee\ q = q \vee\ p$

theorem rule ouAssociativite
$\forall\, p, q, r : BoolGries \bullet p \vee\ q \vee\ r = p \vee\ (q \vee\ r)$

theorem rule ouIdempotence
$\forall\, p : BoolGries \bullet p \vee\ p = p$

theorem rule DistOuSurEquiv1
$\forall\, p, q, r : BoolGries \bullet p \vee\ q \equiv p \vee\ r = p \vee\ (q \equiv r)$

theorem rule DistOuSurEquiv2
$\forall\, p, q, r : BoolGries \bullet p \vee\ (q \equiv r) = p \vee\ q \equiv p \vee\ r$

theorem rule TiersExclu
$\forall\, p : BoolGries \bullet p \vee\ \neg\ p = \text{vrai}$

Axiome de la conjonction :

theorem rule RegleDOr
$\forall\, p, q : BoolGries \bullet p \wedge\ q = p \equiv q \equiv p \vee\ q$

Axiome de l'implication :

theorem rule implicationDef
$\forall\, p, q : BoolGries \bullet p \Rightarrow\ q = p \vee\ q \equiv q$

Axiome de la conséquence :

theorem rule consequenceDef
$\forall\, p, q : BoolGries \bullet p \Leftarrow\ q = q \Rightarrow\ p$

Annexe C

Annexe - Théorèmes testés dans Z/EVES ainsi que leur preuve respective

Théorèmes de l'équivalence :

theorem equivIdentite5
$\forall\, p : BoolGries \bullet p \equiv \text{vrai} = p$
proof
*use equivCommutativite*1$[q := \text{vrai}]$
rewrite

Ajout de lemmes :

theorem lemme1
$\forall\, p, q : BoolGries \bullet p = q \Rightarrow p \equiv q = \text{vrai}$
proof
prove

theorem lemme1Point1
$\forall\, p, q : BoolGries \bullet p = q \Rightarrow \text{vrai} = p \equiv q$
proof
prove

theorem lemme2
$\forall\, p, q : BoolGries \bullet p \equiv q = \text{vrai} \Rightarrow p = q$
proof
*use equivIdentite*4
with normalization simplify
rearrange
*use lemme*1*Point*1
*use equivCommutativite*1
*use equivAssociativite*1$[p := q, q := p, r := p]$
*use equivIdentite*1
*use equivIdentite*5$[p := q]$
with disabled (*equivIdentite*2) *prove by reduce*

Fin de l'ajout de lemmes

theorem ModificationTheo
$\forall\, p, q : BoolGries \bullet p = q \Leftrightarrow p \equiv q = \text{vrai}$
proof
*use lemme*1
*use lemme*2
prove
with normalization simplify

theorem EssaiTheo1
$\forall\, p, q : BoolGries \bullet p \equiv p \equiv q \equiv q = \text{vrai}$
proof
rewrite

theorem VraiTheo
$\forall\, p : BoolGries \bullet \text{vrai} = \text{vrai}$
proof
prove

theorem egaliteReflexivite
$\forall\, p : BoolGries \bullet p = p$
proof
prove

Théorèmes de faux :

theorem fauxDef2
$\forall\, p : BoolGries \bullet \text{faux} \equiv \neg\ \text{vrai} = \text{vrai}$
proof
use ModificationTheo$[p := \text{faux}, q := \neg\ \text{vrai}]$
prove

theorem fauxDef3
$\forall\, p : BoolGries \bullet \neg\ \text{vrai} = \text{faux}$
proof
prove

Théorèmes de la négation :

theorem DistNegSurEquiv3
$\forall\, p, q : BoolGries \bullet \neg\ (p \equiv q) \equiv (\neg\ p \equiv q) = \text{vrai}$
proof
use $ModificationTheo[p := \neg\ (p \equiv q), q := (\neg\ p \equiv q)]$
prove

theorem DistNegSurEquiv4
$\forall\, p, q : BoolGries \bullet \neg\ (p \equiv q) \equiv \neg\ p = q$
proof
reduce
apply $equivCommutativite1$ *to expression* $\neg\ p \equiv q$
simplify
rewrite

Théorème de l'inéquivalence :

theorem nonequivDef2
$\forall\, p, q : BoolGries \bullet \neg\ p \equiv q = p \not\equiv\ q$
proof
prove

Théorèmes de l'équivalence, de la négation et de l'inéquivalence :

theorem TroisPointQuatorze
$\forall\, p, q : BoolGries \bullet \neg\ p \equiv q = p \equiv \neg\ q$
proof
rewrite
apply $equivCommutativite1$ *to expression* $p \equiv \neg\ q$
simplify
prove by reduce

theorem rule DoubleNegation
$\forall\, p : BoolGries \bullet \neg\ (\neg\ p) = p$
proof
use $DistNegSurEgalite[p := \neg\ p, q := p]$
prove

theorem DoubleNegation2
$\forall\, p : \mathit{BoolGries} \bullet p = \neg\ (\neg\ p)$
proof
prove

theorem fauxNegation
$\forall\, p : \mathit{BoolGries} \bullet \neg\ \text{faux} = \text{vrai}$
proof
prove

theorem TroisPointDixSept
$\forall\, p, q : \mathit{BoolGries} \bullet p \not\equiv\ q = \neg\ p \equiv q$
proof
prove by reduce

theorem TroisPointDixHuit1
$\forall\, p : \mathit{BoolGries} \bullet \neg\ p \equiv p = \text{faux}$
proof
*use DistNegSurEquiv*1$[q := p]$
rewrite

theorem TroisPointDixHuit2
$\forall\, p : \mathit{BoolGries} \bullet \text{faux} = \neg\ p \equiv p$
proof
*use TroisPointDixHuit*1
rewrite

theorem rule nonequivCommutativite
$\forall\, p, q : \mathit{BoolGries} \bullet p \not\equiv\ q = q \not\equiv\ p$
proof
rewrite

theorem rule nonequivAssociativite
$\forall\, p, q, r : BoolGries \bullet p \not\equiv q \not\equiv r = p \not\equiv (q \not\equiv r)$
proof
use nonequivDef
*use DistNegSurEquiv*1
rewrite
*apply equivCommutativite*1 *to expression* $\neg\, r \equiv \neg\, (p \equiv q)$
simplify
rewrite
*apply equivCommutativite*1 *to expression* $q \equiv r$
simplify
*apply DistNegSurEquiv*2 *to expression* $\neg\, (r \equiv q)$
simplify
*apply equivCommutativite*1 *to expression* $\neg\, r \equiv q$
simplify
rewrite

theorem AssociativiteMutuelle
$\forall\, p, q, r : BoolGries \bullet p \not\equiv q \equiv r = p \not\equiv (q \equiv r)$
proof
rewrite

theorem InterchangeabiliteMutuelle
$\forall\, p, q, r : BoolGries \bullet p \not\equiv q \equiv r = p \equiv q \not\equiv r$
proof
rewrite

Théorèmes de la disjonction :

theorem ouAssociativite2
$\forall\, p, q, r : BoolGries \bullet p \vee (q \vee r) = p \vee q \vee r$
proof
prove

theorem ouIdempotence2
$\forall\, p : BoolGries \bullet p = p \vee p$
proof
rewrite

theorem TiersExclu2
$\forall\, p : \mathit{BoolGries} \bullet \neg\ p \vee\ p = \text{vrai}$
proof
use ouCommutativite$[p := \neg\ p, q := p]$
rewrite

theorem rule ZeroDeOu1
$\forall\, p : \mathit{BoolGries} \bullet p \vee\ \text{vrai} = \text{vrai}$
proof
*use equivIdentite*2
rewrite

theorem rule ZeroDeOu2
$\forall\, p : \mathit{BoolGries} \bullet \text{vrai} \vee\ p = \text{vrai}$
proof
rewrite

Ajout de lemmes :

theorem lemme3
$\forall\, p : \mathit{BoolGries} \bullet p \vee\ \text{faux} = p \vee\ (\neg\ p \equiv p)$
proof
*use TroisPointDixHuit*2
rewrite

theorem lemme4
$\forall\, p : \mathit{BoolGries} \bullet p \vee\ (\neg\ p \equiv p) = p \equiv \text{vrai}$
proof
*use DistOuSurEquiv*2$[q := \neg\ p, r := p]$
rewrite
*use equivIdentite*5
prove

theorem lemme5
$\forall\, p : \mathit{BoolGries} \bullet p \vee\ \text{faux} = p \equiv \text{vrai}$
proof
*use lemme*3
*use lemme*4
rewrite

Fin de l'ajout de lemmes

theorem rule ouIdentite1
$\forall\, p : BoolGries \bullet p \vee \text{faux} = p$
proof
*use lemme*5
rewrite

theorem ouIdentite2
$\forall\, p : BoolGries \bullet \text{faux} \vee\ p = p$
proof
apply ouCommutativite to expression faux $\vee\ p$
with disabled (*equivIdentite*2, *fauxDef*) *rewrite*

Ajout de lemmes :

theorem lemme6
$\forall\, p, q, r : BoolGries \bullet p \vee\ (q \vee\ r) = p \vee\ p \vee\ (q \vee\ r)$
proof
prove

theorem lemme7
$\forall\, p, q, r : BoolGries \bullet p \vee\ p \vee\ (q \vee\ r) = p \vee\ q \vee\ (p \vee\ r)$
proof
with disabled (*ouIdempotence*) *rewrite*
apply ouCommutativite to expression $q \vee\ (p \vee\ r)$
rewrite

Fin de l'ajout de lemmes

theorem DistOuSurOu1
$\forall\, p, q, r : BoolGries \bullet p \vee\ (q \vee\ r) = p \vee\ q \vee\ (p \vee\ r)$
proof
*use lemme*6
*use lemme*7
rewrite

theorem DistOuSurOu2
$\forall\, p, q, r : BoolGries \bullet p \vee\ q \vee\ (p \vee\ r) = p \vee\ (q \vee\ r)$
proof
*use DistOuSurOu*1
prove

theorem TroisPointQuaranteDeux1
$\forall\, p, q : BoolGries \bullet p \vee\ q \equiv p \vee\ \neg\ q = p$
proof
rewrite
*apply equivCommutativite*1 *to expression* $q \equiv \neg\ q$
simplify
*use TroisPointDixHuit*1$[p := q]$
with disabled (*fauxDef*) *rewrite*

theorem TroisPointQuaranteDeux2
$\forall\, p, q : BoolGries \bullet p \vee\ q = p \vee\ \neg\ q \equiv p$
proof
*use TroisPointQuaranteDeux*1
use ModificationTheo$[p := p \vee\ q \equiv p \vee\ \neg\ q, q := p]$
use ModificationTheo$[p := p \vee\ q, q := p \vee\ \neg\ q \equiv p]$
prove by reduce

theorem TroisPointQuaranteDeux3
$\forall\, p, q : BoolGries \bullet p \vee\ q \equiv p = p \vee\ \neg\ q$
proof
*use TroisPointQuaranteDeux*1
use ModificationTheo$[p := p \vee\ q \equiv p \vee\ \neg\ q, q := p]$
use ModificationTheo$[p := p \vee\ q \equiv p, q := p \vee\ \neg\ q]$
prove by reduce

theorem TroisPointQuaranteDeux4
$\forall\, p, q : BoolGries \bullet p \vee\ \neg\ q = p \vee\ q \equiv p$
proof
*use TroisPointQuaranteDeux*3
prove

Théorèmes de la conjonction :

theorem RegleDOr2
$\forall\, p, q : BoolGries \bullet p \wedge q \equiv p = q \equiv p \vee q$
proof
use ModificationTheo$[p := p \wedge q, q := p \equiv q \equiv p \vee q]$
use ModificationTheo$[p := p \wedge q \equiv p, q := q \equiv p \vee q]$
rewrite
*apply equivIdentite*1 *to expression* $p \equiv p$
rewrite

theorem RegleDOr3
$\forall\, p, q : BoolGries \bullet p \wedge q \equiv p \equiv q = p \vee q$
proof
use ModificationTheo$[p := p \wedge q, q := p \equiv q \equiv p \vee q]$
use ModificationTheo$[p := p \wedge q \equiv p \equiv q, q := p \vee q]$
rewrite
*apply equivIdentite*1 *to expression* $p \equiv p$
rewrite

theorem rule etCommutativite
$\forall\, p, q : BoolGries \bullet p \wedge q = q \wedge p$
proof
reduce

Ajout de lemmes :

theorem lemme8
$\forall\, p, q, r : BoolGries \bullet p \wedge\ q \wedge\ r = p \equiv q \equiv p \vee\ q \equiv r \equiv p \vee\ r \equiv q \vee\ r \equiv p \vee\ q \vee\ r$
proof
apply RegleDOr to expression $p \wedge\ q$
simplify
apply RegleDOr to expression $(p \equiv q \equiv p \vee\ q) \wedge\ r$
simplify
apply ouCommutativite to expression $(p \equiv q \equiv p \vee\ q) \vee\ r$
simplify
*apply DistOuSurEquiv*2 *to expression* $r \vee\ (p \equiv q \equiv p \vee\ q)$
simplify
*apply DistOuSurEquiv*2 *to expression* $r \vee\ (p \equiv q)$
simplify
apply ouCommutativite to expression $r \vee\ p$
simplify
apply ouCommutativite to expression $r \vee\ q$
simplify
apply ouCommutativite to expression $r \vee\ (p \vee\ q)$
simplify
*use equivAssociativite*1$[p := p \equiv q \equiv p \vee\ q \equiv r, q := p \vee\ r \equiv q \vee\ r, r := p \vee\ q \vee\ r]$
rewrite

theorem lemme9
$\forall\, p, q, r : BoolGries \bullet p \wedge\ (q \wedge\ r) = p \equiv q \equiv r \equiv q \vee\ r \equiv p \vee\ q \equiv p \vee\ r \equiv p \vee\ q \vee\ r$
proof
rewrite

theorem lemme10
$\forall\, p, q, r : BoolGries \bullet p \equiv q \equiv p \vee\ q \equiv r \equiv p \vee\ r \equiv q \vee\ r \equiv p \vee\ q \vee\ r = p \equiv q \equiv r \equiv q \vee\ r \equiv p \vee\ q \equiv p \vee\ r \equiv p \vee\ q \vee\ r$
proof
rewrite

Fin de l'ajout de lemmes

theorem rule etAssociativite
$\forall\, p, q, r : BoolGries \bullet p \wedge\ q \wedge\ r = p \wedge\ (q \wedge\ r)$
proof
*use lemme*8
*use lemme*9
*use lemme*10
with disabled (*RegleDOr*, *etCommutativite*, *ouCommutativite*, *ouAssociativite*, *ouIdempotence*, *DistOuSurEquiv*1, *equivCommutativite*1, *equivAssociativite*1, *equivAssociativite*2, *equivCommutativite*3) *rewrite*

theorem rule etIdempotence
$\forall\, p : BoolGries \bullet p \wedge\ p = p$
proof
rewrite

theorem etIdempotence2
$\forall\, p : BoolGries \bullet p = p \wedge\ p$
proof
rewrite

theorem rule etIdentite1
$\forall\, p : BoolGries \bullet p \wedge\ \text{vrai} = p$
proof
rewrite

theorem rule etIdentite2
$\forall\, p : BoolGries \bullet \text{vrai} \wedge\ p = p$
proof
rewrite

theorem etIdentite3
$\forall\, p : BoolGries \bullet p = p \wedge\ \text{vrai}$
proof
rewrite

theorem etIdentite4
$\forall\, p : BoolGries \bullet p = \text{vrai} \wedge p$
proof
rewrite

theorem rule ZeroDeEt1
$\forall\, p : BoolGries \bullet p \wedge \text{faux} = \text{faux}$
proof
with disabled (*fauxDef*) *rewrite*

theorem rule ZeroDeEt2
$\forall\, p : BoolGries \bullet \text{faux} \wedge p = \text{faux}$
proof
with disabled (*fauxDef*) *rewrite*

theorem ZeroDeEt3
$\forall\, p : BoolGries \bullet \text{faux} = p \wedge \text{faux}$
proof
with disabled (*fauxDef*) *rewrite*

theorem ZeroDeEt4
$\forall\, p : BoolGries \bullet \text{faux} = \text{faux} \wedge p$
proof
with disabled (*fauxDef*) *rewrite*

Ajout de lemmes :

theorem lemme11
$\forall\, p, q, r : BoolGries \bullet p \wedge (q \wedge r) = p \wedge p \wedge (q \wedge r)$
proof
with disabled (*RegleDOr*) *rewrite*

theorem lemme12
$\forall\, p, q, r : BoolGries \bullet p \wedge p \wedge (q \wedge r) = p \wedge q \wedge (p \wedge r)$
proof
with disabled (*RegleDOr*, *etIdempotence*) *rewrite*
apply etCommutativite to expression $q \wedge (p \wedge r)$
simplify
with disabled (*RegleDOr*) *rewrite*

Fin de l'ajout de lemmes

theorem DistEtSurEt1
$\forall\, p, q, r : BoolGries \bullet p \wedge (q \wedge r) = p \wedge q \wedge (p \wedge r)$
proof
use $lemme11$
use $lemme12$
with disabled $(RegleDOr)$ *rewrite*

theorem DistEtSurEt2
$\forall\, p, q, r : BoolGries \bullet p \wedge q \wedge (p \wedge r) = p \wedge (q \wedge r)$
proof
use $DistEtSurEt1$
with disabled $(RegleDOr, etCommutativite, etAssociativite)$ *rewrite*

theorem rule Contradiction
$\forall\, p : BoolGries \bullet p \wedge \neg p =$ faux
proof
with disabled $(fauxDef)$ *rewrite*
apply $equivCommutativite1$ *to expression* $p \equiv \neg p$
simplify
use $TroisPointDixHuit1$
rewrite

theorem Absorption1
$\forall\, p, q : BoolGries \bullet p \wedge (p \vee q) = p$
proof
rewrite
apply $DistOuSurEquiv2$ *to expression* $p \vee (q \equiv p \vee q)$
simplify
use $ouAssociativite2[q := p, r := q]$
rewrite

theorem Absorption2
$\forall\, p, q : BoolGries \bullet p \vee (p \wedge q) = p$
proof
rewrite
*apply DistOuSurEquiv*2 *to expression* $p \vee (q \equiv p \vee q)$
simplify
*use ouAssociativite*2$[q := p, r := q]$
rewrite

Ajout de lemme :

theorem lemme13
$\forall\, p, q : BoolGries \bullet q \equiv p \vee q = q \vee \neg p$
proof
apply ouCommutativite to expression $p \vee q$
simplify
*apply equivCommutativite*1 *to expression* $q \equiv q \vee p$
simplify
*use TroisPointQuaranteDeux*3$[p := q, q := p]$
rewrite

Fin de l'ajout de lemme

theorem Absorption3
$\forall\, p, q : BoolGries \bullet p \wedge (\neg p \vee q) = p \wedge q$
proof
rewrite
*use DistOuSurOu*1$[r := \neg p]$
rewrite
*use lemme*13
prove

theorem Absorption4
$\forall\, p, q : BoolGries \bullet p \vee (\neg\, p \wedge q) = p \vee q$
proof
rewrite
use $DistOuSurOu1[r := \neg\, p]$
rewrite

Ajout de lemme :

theorem lemme14
$\forall\, p, q : BoolGries \bullet p \vee (p \vee q) = p \vee q$
proof
use $ouAssociativite2[q := p, r := q]$
rewrite

Fin de l'ajout de lemme

theorem rule DistOuSurEt1
$\forall\, p, q, r : BoolGries \bullet p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$
proof
rewrite
apply ouCommutativite to expression $q \vee (p \vee r)$
simplify
use $lemme14[q := r \vee q]$
with disabled $(ouCommutativite)$ *rewrite*
prove

theorem DistOuSurEt2
$\forall\, p, q, r : BoolGries \bullet (p \vee q) \wedge (p \vee r) = p \vee (q \wedge r)$
proof
with disabled $(RegleDOr)$ *prove*

theorem rule DistEtSurOu1
$\forall\, p, q, r : BoolGries \bullet p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$
proof
rewrite
*apply DistOuSurEquiv*2 *to expression* $p \vee (r \equiv p \vee q)$
simplify
*apply equivCommutativite*1 *to expression* $q \vee r \equiv (p \equiv p \vee q)$
simplify
apply ouCommutativite to expression $r \vee (p \vee q)$
simplify
apply ouAssociativite to expression $p \vee q \vee r$
simplify
rewrite
*apply equivCommutativite*1 *to expression* $q \vee r \equiv (p \equiv p \vee r)$
simplify
*apply DistOuSurEquiv*2 *to expression* $p \vee (q \equiv p \vee q)$
simplify
*use lemme*14
*use lemme*14$[q := r]$
with disabled (*equivAssociativite*1, *equivCommutativite*1,
*equivCommutativite*3, *ouAssociativite*, *ouCommutativite*) *rewrite*
*use lemme*14$[q := q \vee r]$
*use equivAssociativite*1$[p := p \vee r \equiv p \vee (q \vee r), q := p, r := p \vee q]$
*use equivCommutativite*1$[p := p \vee r \equiv p \vee (q \vee r), q := p \equiv p \vee r]$
*use equivIdentite*1$[p := p \vee r]$
*use equivCommutativite*1$[p := p \vee (q \vee r), q := q \vee r]$
with disabled (*equivAssociativite*1, *equivCommutativite*1,
*equivCommutativite*3, *ouAssociativite*, *ouCommutativite*,
*DistOuSurEquiv*1, *DistOuSurEquiv*2) *prove*
rewrite

theorem DistEtSurOu2
$\forall\, p, q, r : BoolGries \bullet (p \wedge q) \vee (p \wedge r) = p \wedge (q \vee r)$
proof
*use DistEtSurOu*1
prove

theorem rule DeMorgan1
$\forall\, p, q : BoolGries \bullet \neg\, (p \wedge\ q) = \neg\, p \vee\ \neg\, q$
proof
prove
use $TroisPointQuaranteDeux2[p := \neg\, p, q := \neg\, q]$
rewrite
apply DoubleNegation to expression $\neg\, (\neg\, q)$
simplify
apply ouCommutativite to expression $\neg\, p \vee\ q$
simplify
use $TroisPointQuaranteDeux4[p := q, q := p]$
apply $equivCommutativite1$ *to expression* $q \vee\ p \equiv p$
simplify
prove

theorem rule DeMorgan2
$\forall\, p, q : BoolGries \bullet \neg\, (p \vee\ q) = \neg\, p \wedge\ \neg\, q$
proof
apply RegleDOr to expression $\neg\, p \wedge\ \neg\, q$
simplify
use $TroisPointQuaranteDeux2$
apply $equivCommutativite1$ *to expression* $p \vee\ \neg\, q \equiv p$
simplify
apply ouCommutativite to expression $p \vee\ \neg\, q$
simplify
use $TroisPointQuaranteDeux2[p := \neg\, q, q := p]$
prove

theorem TroisPointCinquanteNeuf1
$\forall\, p, q : BoolGries \bullet p \wedge\ q \equiv p \wedge\ \neg\ q = \neg\ p$
proof
prove
apply ouCommutativite to expression $\neg\ p \vee\ \neg\ q$
simplify
use $TroisPointQuaranteDeux2[p := \neg\ q, q := \neg\ p]$
apply DoubleNegation to expression $\neg\ (\neg\ p)$
simplify
apply $DistNegSurEquiv2$ *to expression* $\neg\ (q \equiv \neg\ q \vee\ \neg\ p)$
simplify
prove

theorem TroisPointCinquanteNeuf2
$\forall\, p, q : BoolGries \bullet p \wedge\ q = p \wedge\ \neg\ q \equiv \neg\ p$
proof
use $TroisPointCinquanteNeuf1$
use $ModificationTheo[p := p \wedge\ q \equiv p \wedge\ \neg\ q, q := \neg\ p]$
use $ModificationTheo[p := p \wedge\ q, q := p \wedge\ \neg\ q \equiv \neg\ p]$
with disabled $(RegleDOr, etCommutativite, etAssociativite)$ *prove*

theorem TroisPointSoixante
$\forall\, p, q, r : BoolGries \bullet p \wedge\ (q \equiv r) = p \wedge\ q \equiv p \wedge\ r \equiv p$
proof
prove

theorem TroisPointSoixanteEtUn
$\forall\, p, q : BoolGries \bullet p \wedge\ (q \equiv p) = p \wedge\ q$
proof
use $TroisPointSoixante[r := p]$
prove

Ajout de lemme :

theorem lemme15
$\forall\, p, q, r : BoolGries \bullet p \equiv (q \equiv r) = q \equiv p \equiv r$
proof
prove

Fin de l'ajout de lemme

theorem Remplacement
$\forall\, p, q, r : BoolGries \bullet (p \equiv q) \wedge (r \equiv p) = (p \equiv q) \wedge (r \equiv q)$
proof
prove
*apply equivCommutativite*1 *to expression* $q \vee r \equiv (p \equiv p \vee q)$
simplify
*apply equivCommutativite*1 *to expression* $p \vee q \equiv (p \equiv q \equiv q \vee r)$
simplify
*apply equivCommutativite*1 *to expression* $p \equiv q$
simplify
*use lemme*15$[p := r \equiv p \vee r, r := p \equiv q \vee r \equiv p \vee q]$
*use equivCommutativite*1$[p := q \vee r, q := p \vee q]$
prove

Ajout de lemme :

theorem lemme16
$\forall\, p, q : BoolGries \bullet \neg q \vee (p \vee q) = \text{vrai}$
proof
apply ouCommutativite to expression $\neg q \vee (p \vee q)$
rewrite

Fin de l'ajout de lemme

theorem equivDef
$\forall\, p, q : BoolGries \bullet p \equiv q = (p \wedge q) \vee (\neg\, p \wedge \neg\, q)$
proof
apply RegleDOr
simplify
*apply DistOuSurEquiv*2
simplify
*apply DistOuSurEquiv*2
simplify
apply ouCommutativite
simplify
*apply DistOuSurEquiv*2
simplify
*use lemme*16
*use lemme*16$[p := q, q := p]$
apply ouAssociativite to expression $\neg\, q \vee \neg\, p \vee (q \vee p)$
simplify
apply ouCommutativite to expression $q \vee p$
simplify
*apply ZeroDeOu*1
simplify
*use equivIdentite*5$[p := \neg\, p \vee (p \equiv q)]$
*use equivIdentite*5$[p := \neg\, q \vee (p \equiv q)]$
*use equivIdentite*5$[p := \neg\, q \vee \neg\, p \vee (p \equiv q)]$
*apply DistOuSurEquiv*2 *to expression* $\neg\, p \vee (p \equiv q)$
simplify
*apply DistOuSurEquiv*2 *to expression* $\neg\, q \vee (p \equiv q)$
simplify
*apply DistOuSurEquiv*2 *to expression* $\neg\, q \vee \neg\, p \vee (p \equiv q)$
simplify
apply ouAssociativite to expression $\neg\, q \vee \neg\, p \vee p$
simplify
apply ouCommutativite to expression $\neg\, q \vee \neg\, p \vee q$
simplify
*use TiersExclu*2
*use TiersExclu*2$[p := q]$

theorem equivDef (suite de la preuve précédante)
$\forall\, p, q : BoolGries \bullet p \equiv q = (p \wedge q) \vee (\neg\, p \wedge \neg\, q)$
proof
simplify
*apply ZeroDeOu*1
simplify
*apply equivIdentite*3 *to expression* vrai $\equiv \neg\, p \vee q$
*apply equivIdentite*3 *to expression* vrai $\equiv q \vee (\neg\, q \vee \neg\, p)$
simplify
*apply equivCommutativite*1 *to expression* $\neg\, q \vee p \equiv$ vrai
simplify
*apply equivAssociativite*1 *to expression* vrai $\equiv \neg\, q \vee p \equiv$ vrai
*apply equivCommutativite*1 *to expression* $\neg\, q \vee p \equiv$ vrai
simplify
*apply equivIdentite*3 *to expression* vrai $\equiv \neg\, q \vee p$
simplify
apply ouCommutativite to expression $\neg\, q \vee \neg\, p$
simplify
*use lemme*16$[p := \neg\, p, q := \neg\, q]$
apply DoubleNegation
simplify
apply ouCommutativite to expression $\neg\, p \vee q$
apply ouCommutativite to expression $\neg\, q \vee p$
simplify
*use TroisPointQuaranteDeux*4
*use TroisPointQuaranteDeux*4$[p := q, q := p]$
rewrite
*apply equivCommutativite*1 *to expression* $q \equiv p \vee q$
simplify
prove

theorem ouExclusif
$\forall\, p, q : BoolGries \bullet p \not\equiv q = (\neg\, p \wedge q) \vee (p \wedge \neg\, q)$
proof
apply RegleDOr
prove
*use lemme*16$[p := \neg\, p]$
apply ouCommutativite to expression $\neg\, p \vee q$
simplify
*use lemme*16$[p := q, q := \neg\, p]$
apply DoubleNegation
simplify
apply ouCommutativite to expression $\neg\, p \vee \neg\, q$
simplify
*apply ZeroDeOu*1 *to expression* $p \vee$ vrai
simplify
*apply equivIdentite*3 *to expression* vrai $\equiv (p \vee q \equiv p \vee (\neg\, q \vee \neg\, p))$
simplify
*apply equivIdentite*3 *to expression* vrai $\equiv (p \vee q \equiv p \vee (\neg\, q \vee \neg\, p))$
simplify
*apply equivCommutativite*1 *to expression* $\neg\, q \vee \neg\, p \equiv$ vrai
simplify
*apply equivIdentite*3 *to expression* vrai $\equiv \neg\, q \vee \neg\, p$
simplify
*use lemme*16$[p := \neg\, q, q := \neg\, p]$
apply DoubleNegation
simplify
*apply equivCommutativite*1 *to expression* $p \vee q \equiv$ vrai
simplify
*apply equivIdentite*3 *to expression* vrai $\equiv p \vee q$
rewrite
apply ouCommutativite to expression $p \vee q$
simplify
*use TroisPointQuaranteDeux*2$[p := q, q := p]$
*use TroisPointQuaranteDeux*4$[p := \neg\, p]$
prove

theorem TroisPointSoixanteHuit
$\forall\, p, q, r : BoolGries \bullet p \wedge\ q \wedge\ r = p \equiv q \equiv r \equiv p \vee\ q \equiv q \vee\ r \equiv p \vee\ r \equiv p \vee\ q \vee\ r$
proof
apply RegleDOr
simplify
apply ouCommutativite to expression $(p \equiv q \equiv p \vee\ q) \vee\ r$
simplify
*use DistOuSurEquiv*2$[p := r, q := p \equiv q, r := p \vee\ q]$
*use DistOuSurEquiv*2$[p := r, q := p, r := q]$
apply ouCommutativite to expression $r \vee\ p$
simplify
apply ouCommutativite to expression $r \vee\ q$
simplify
prove
apply ouCommutativite to expression $r \vee\ (p \vee\ q)$
simplify
*apply equivCommutativite*1 *to expression* $p \vee\ q \vee\ r \equiv (p \vee\ (q \equiv r) \equiv (r \equiv (p \equiv q \equiv q \vee\ r)))$
simplify
*apply equivCommutativite*1 *to expression* $r \equiv (p \equiv q \equiv q \vee\ r)$
simplify
*apply equivAssociativite*1 *to expression* $p \equiv q \equiv q \vee\ r$
simplify
*apply equivCommutativite*1 *to expression* $p \equiv (q \equiv q \vee\ r)$
simplify
*use equivAssociativite*1$[p := q \equiv q \vee\ r, q := p]$
*use lemme*15$[p := p \vee\ (q \equiv r), q := q \equiv q \vee\ r, r := p \equiv r]$
prove

Théorèmes de l'implication :

theorem DefAltDeImplication1
$\forall\, p, q : BoolGries \bullet p \Rightarrow\ q = \neg\ p \vee\ q$
proof
apply implicationDef
simplify
apply ouCommutativite to expression $\neg\ p \vee\ q$
simplify
*use TroisPointQuaranteDeux*4$[p := q, q := p]$
apply ouCommutativite to expression $q \vee\ p$
simplify
prove

theorem DefAltDeImplication2
$\forall\, p, q : BoolGries \bullet p \Rightarrow\ q = p \wedge\ q \equiv p$
proof
*use RegleDOr*2
use implicationDef
*apply equivCommutativite*1 *to expression* $p \vee\ q \equiv q$
simplify
prove

theorem Contrapositivite
$\forall\, p, q : BoolGries \bullet p \Rightarrow\ q = \neg\ q \Rightarrow\ \neg\ p$
proof
*use DefAltDeImplication*1
*use TroisPointQuaranteDeux*2$[p := \neg\ p]$
apply ouCommutativite to expression $\neg\ p \vee\ \neg\ q$
simplify
use implicationDef$[p := \neg\ q, q := \neg\ p]$
prove

theorem rule TroisPointSoixanteQuinze
$\forall\, p, q, r : BoolGries \bullet p \Rightarrow\ (q \equiv r) = p \wedge\ q \equiv p \wedge\ r$
proof
*use DefAltDeImplication*2$[q := q \equiv r]$
use TroisPointSoixante
prove

theorem DistImplicationSurEquiv
$\forall\, p, q, r : BoolGries \bullet p \Rightarrow (q \equiv r) = p \Rightarrow q \equiv p \Rightarrow r$
proof
use $DefAltDeImplication2[q := q \equiv r]$
use *TroisPointSoixante*
use $equivCommutativite1[p := p \wedge r, q := p]$
use $DefAltDeImplication2[q := r]$
prove

Ajout de lemme :

theorem lemme18
$\forall\, p, q, r : BoolGries \bullet r \vee (\neg p \vee \neg q) \equiv r \vee (\neg p \vee (p \vee \neg q)) =$
$\neg p \vee (r \vee \neg q)$
proof
use $lemme16[p := \neg q, q := p]$
apply *ouCommutativite to expression* $\neg q \vee p$
with disabled ($DistNegSurEquiv1$, $DistNegSurEquiv2$,
$equivAssociativite2$, *ouAssociativite*, *ouCommutativite*,
$equivAssociativite1$, $equivCommutativite1$, $equivCommutativite3$,
$DistOuSurEquiv2$, $DistOuSurEquiv1$, $equivIdentite2$) *prove*
apply $equivCommutativite1$ *to expression* $r \vee (\neg p \vee \neg q) \equiv$ vrai
simplify
apply $equivIdentite3$ *to expression* vrai $\equiv r \vee (\neg p \vee \neg q)$
simplify
apply *ouCommutativite to expression* $r \vee (\neg p \vee \neg q)$
simplify
apply *ouAssociativite to expression* $\neg p \vee \neg q \vee r$
simplify
apply *ouCommutativite to expression* $\neg q \vee r$
simplify
with disabled ($DistNegSurEquiv1$, $DistNegSurEquiv2$,
$equivAssociativite2$, *ouAssociativite*, *ouCommutativite*,
$equivAssociativite1$, $equivCommutativite1$, $equivCommutativite3$,
$DistOuSurEquiv2$, $DistOuSurEquiv1$, $equivIdentite2$) *prove*

Fin de l'ajout de lemme

theorem TroisPointSoixanteDixSept
$\forall\, p, q, r : BoolGries \bullet p \Rightarrow (q \Rightarrow r) = (p \Rightarrow q) \Rightarrow (p \Rightarrow r)$
proof
use $DefAltDeImplication1[p := q, q := r]$
use $DefAltDeImplication1[q := \neg\, q \vee r]$
use $DefAltDeImplication1[p := p \Rightarrow q, q := p \Rightarrow r]$
use $DefAltDeImplication1[q := r]$
use $DefAltDeImplication1$
with disabled (*implicationDef*) *prove*
apply $DistNegSurEquiv2$ *to expression* $\neg\, (q \equiv p \vee \neg\, q)$
simplify
use $DistOuSurEquiv2[p := \neg\, p, q := \neg\, q, r := p \vee \neg\, q]$
use $DistOuSurEquiv2[p := r, q := \neg\, p \vee \neg\, q, r := \neg\, p \vee (p \vee \neg\, q)]$
use $lemme18$
with disabled ($DistNegSurEquiv1$, $DistNegSurEquiv2$, $equivAssociativite2$, $ouAssociativite$, $ouCommutativite$, $equivAssociativite1$, $equivCommutativite1$, $equivCommutativite3$, $DistOuSurEquiv2$, $DistOuSurEquiv1$, $equivIdentite2$) *prove*

theorem Transfert
$\forall\, p, q, r : BoolGries \bullet p \wedge q \Rightarrow r = p \Rightarrow (q \Rightarrow r)$
proof
use $DefAltDeImplication2[p := p \wedge q, q := r]$
use $DefAltDeImplication2[p := q, q := r]$
use $DefAltDeImplication2[q := q \wedge r \equiv q]$
with disabled (*RegleDOr*, *implicationDef*) *prove*

Ajout de lemme :

theorem lemme19
$\forall\, p, q : BoolGries \bullet p \wedge (p \wedge q) = p \wedge q$
proof
apply etCommutativite to expression $p \wedge (p \wedge q)$
simplify
apply etCommutativite to expression $p \wedge q$
simplify
apply etAssociativite to expression $q \wedge p \wedge p$
simplify
with disabled (*RegleDOr*, *etCommutativite*, *etAssociativite*) *prove*

Fin de l'ajout de lemme

theorem TroisPointSoixanteDixNeuf
$\forall\, p, q : BoolGries \bullet p \wedge (p \Rightarrow q) = p \wedge q$
proof
*use DefAltDeImplication*2
with disabled (*RegleDOr*, *implicationDef*) *prove*
*apply equivCommutativite*1 *to expression* $p \equiv p \wedge q$
simplify
use TroisPointSoixanteEtUn$[q := p \wedge q]$
*use lemme*19
with disabled (*RegleDOr*, *etCommutativite*, *etAssociativite*) *prove*

theorem TroisPointQuatreVingt
$\forall\, p, q : BoolGries \bullet p \wedge (q \Rightarrow p) = p$
proof
*use DefAltDeImplication*2$[p := q, q := p]$
use TroisPointSoixante$[q := q \wedge p, r := q]$
apply etCommutativite to expression $q \wedge p$
simplify
*use lemme*19
*use equivIdentite*1$[p := p \wedge q]$
with disabled (*RegleDOr*, *etCommutativite*,
etAssociativite, *equivAssociativite*1, *equivCommutativite*1,
*equivCommutativite*3) *prove*

theorem TroisPointQuatreVingtUn
$\forall\, p, q : BoolGries \bullet p \vee (p \Rightarrow q) = \text{vrai}$
proof
apply implicationDef
simplify
*use DistOuSurEquiv*2$[q := p \vee q, r := q]$
apply ouCommutativite to expression $p \vee (p \vee q)$
simplify
apply ouCommutativite to expression $p \vee q$
simplify
apply ouAssociativite to expression $q \vee p \vee p$
simplify
apply ouIdempotence
simplify
with disabled (*ouAssociativite*, *ouCommutativite*) *prove*

theorem TroisPointQuatreVingtDeux
$\forall\, p, q : BoolGries \bullet p \vee (q \Rightarrow p) = q \Rightarrow p$
proof
apply implicationDef
simplify
*use DistOuSurEquiv*2$[q := q \vee p, r := p]$
apply ouCommutativite to expression $q \vee p$
simplify
*use lemme*14
apply ouIdempotence
simplify
with disabled (*ouAssociativite*, *ouCommutativite*) *prove*

theorem TroisPointQuatreVingtTrois
$\forall\, p, q : BoolGries \bullet p \vee\ q \Rightarrow\ p \wedge\ q = p \equiv q$
proof
apply implicationDef to expression $p \vee\ q \Rightarrow\ p \wedge\ q$
simplify
use $DistOuSurEt1[p := p \vee\ q, q := p, r := q]$
apply ouAssociativite to expression $p \vee\ q \vee\ p$
simplify
apply ouCommutativite to expression $q \vee\ p$
simplify
*use lemme*14
apply ouAssociativite to expression $p \vee\ q \vee\ q$
simplify
apply ouIdempotence to expression $q \vee\ q$
simplify
apply etIdempotence to expression $p \vee\ q \wedge\ (p \vee\ q)$
simplify
apply RegleDOr to expression $p \wedge\ q$
simplify
apply $equivCommutativite1$ *to expression* $p \vee\ q \equiv (p \equiv q \equiv p \vee\ q)$
simplify
use $equivAssociativite1[p := p \equiv q, q := p \vee\ q, r := p \vee\ q]$
apply $equivIdentite1$ *to expression* $p \vee\ q \equiv p \vee\ q$
simplify
with disabled $(ouAssociativite, ouCommutativite)$ *prove*

theorem rule ReflexiviteDeImplication
$\forall\, p : BoolGries \bullet p \Rightarrow\ p =$ vrai
proof
prove

theorem ZeroADroiteDeImplication
$\forall\, p : BoolGries \bullet p \Rightarrow$ vrai = vrai
proof
prove

theorem IdentiteAGaucheDeImplication
$\forall\, p : \mathit{BoolGries} \bullet \text{vrai} \Rightarrow\ p = p$
proof
prove

theorem TroisPointQuatreVingtSept
$\forall\, p : \mathit{BoolGries} \bullet p \Rightarrow\ \text{faux} = \neg\ p$
proof
prove

theorem TroisPointQuatreVingtHuit
$\forall\, p : \mathit{BoolGries} \bullet \text{faux} \Rightarrow\ p = \text{vrai}$
proof
prove
*use ouIdentite*1
use fauxDef
prove

theorem Affaiblissement1
$\forall\, p, q : \mathit{BoolGries} \bullet p \Rightarrow\ p \vee\ q = \text{vrai}$
proof
prove
*use lemme*14
prove

theorem Affaiblissement2
$\forall\, p, q : \mathit{BoolGries} \bullet p \wedge q \Rightarrow p = \text{vrai}$
proof
use $\mathit{DefAltDeImplication}2[p := p \wedge q, q := p]$
apply etAssociativite to expression $p \wedge q \wedge p$
simplify
apply etCommutativite to expression $p \wedge (q \wedge p)$
simplify
apply etAssociativite to expression $q \wedge p \wedge p$
simplify
apply etIdempotence
simplify
apply etCommutativite to expression $q \wedge p$
simplify
apply $\mathit{equivIdentite}1$ *to expression* $p \wedge q \equiv p \wedge q$
simplify
with disabled $(\mathit{RegleDOr}, \mathit{implicationDef})$ *prove*

theorem Affaiblissement3
$\forall\, p, q : \mathit{BoolGries} \bullet p \wedge q \Rightarrow p \vee q = \text{vrai}$
proof
prove
use $\mathit{DistOuSurEquiv}2[r := p \vee q]$
use $\mathit{lemme}14$
simplify
apply $\mathit{equivIdentite}1$ *to expression* $p \vee q \equiv p \vee q$
simplify
prove

theorem Affaiblissement4
$\forall\, p, q, r : BoolGries \bullet p \vee (q \wedge r) \Rightarrow p \vee q = \text{vrai}$
proof
with disabled (*DistOuSurEquiv*1) *prove*
apply ouCommutativite to expression $q \vee (p \vee (q \vee r))$
simplify
apply ouCommutativite to expression $p \vee (q \vee r)$
simplify
apply ouAssociativite to expression $q \vee r \vee p$
simplify
apply ouCommutativite to expression $q \vee (r \vee p)$
simplify
apply ouAssociativite to expression $r \vee p \vee q \vee q$
simplify
apply ouIdempotence to expression $q \vee q$
simplify
apply ouCommutativite to expression $r \vee p \vee q$
simplify
apply ouCommutativite to expression $r \vee p$
simplify
*use equivIdentite*1$[p := p \vee (q \vee (p \vee r))]$
with disabled (*ouAssociativite*, *ouCommutativite*, *equivAssociativite*1, *equivCommutativite*1, *equivCommutativite*3, *DistOuSurEquiv*1, *DistOuSurEquiv*2) *prove*

Ajout de lemmes :

theorem lemme20
$\forall\, p, q, r : BoolGries \bullet p \wedge (q \wedge r) = p \wedge q \wedge r$
proof
with disabled (*RegleDOr*) *prove*

theorem lemme21
$\forall\, p, q, r : BoolGries \bullet p \vee (q \vee r) = p \vee q \vee r$
proof
prove

theorem lemme22
$\forall\, p, q, r : BoolGries \bullet p \Rightarrow q \vee r = (p \Rightarrow q) \vee (p \Rightarrow r)$
proof
use $DefAltDeImplication1[q := q \vee r]$
use $DefAltDeImplication1$
use $DefAltDeImplication1[q := r]$
use $DistOuSurOu1[p := \neg p]$
with disabled ($implicationDef$, $ouCommutativite$, $ouAssociativite$, $DeMorgan1$, $DeMorgan2$) *prove*

Fin de l'ajout de lemmes

theorem Affaiblissement5
$\forall\, p, q, r : BoolGries \bullet p \wedge q \Rightarrow p \wedge (q \vee r) = \text{vrai}$
proof
apply $DistEtSurOu1$
simplify
use $lemme22[p := p \wedge q, q := p \wedge q, r := p \wedge r]$
apply $ReflexiviteDeImplication$
simplify
apply $ZeroDeOu2$ *to expression* $\text{vrai} \vee (p \wedge q \Rightarrow p \wedge r)$
simplify
with disabled ($implicationDef$, $ouCommutativite$, $ouAssociativite$, $DeMorgan1$, $DeMorgan2$, $etCommutativite$, $etAssociativite$, $DistOuSurEt1$, $DistEtSurOu1$, $RegleDOr$) *prove*

theorem ModusPonens
$\forall\, p, q : BoolGries \bullet p \wedge (p \Rightarrow q) \Rightarrow q = \text{vrai}$
proof
use $DefAltDeImplication1$
use $DefAltDeImplication1[p := p \wedge (\neg p \vee q)]$
use $DeMorgan2[p := p \wedge (\neg p \vee q)]$
use $DeMorgan1[q := \neg p \vee q]$
use $DeMorgan2[p := \neg p]$

theorem ModusPonens (suite de la preuve précédante)
$\forall\, p, q : \mathit{BoolGries} \bullet p \wedge (p \Rightarrow q) \Rightarrow q = \text{vrai}$
proof
apply DoubleNegation to expression $\neg\ (\neg\ p)$
simplify
with disabled (*RegleDOr*, *implicationDef*, *DistOuSurEt*1, *DistEtSurOu*1, *DeMorgan*1, *DeMorgan*2, *ouCommutativite*, *ouAssociativite*, *etAssociativite*, *etCommutativite*) *prove*
apply RegleDOr to expression $p \wedge \neg\ q$
simplify
with disabled (*RegleDOr*, *implicationDef*, *DistOuSurEt*1, *DistEtSurOu*1, *DeMorgan*1, *DeMorgan*2, *ouCommutativite*, *ouAssociativite*, *etAssociativite*, *etCommutativite*, *DistNegSurEquiv*1, *DistOuSurEquiv*1) *prove*
apply ouCommutativite to expression $\neg\ p \vee p$
simplify
apply TiersExclu to expression $p \vee \neg\ p$
simplify
apply ouCommutativite to expression $p \vee \neg\ q$
simplify
*use lemme*16$[p := \neg\ q, q := p]$
with disabled (*RegleDOr*, *implicationDef*, *DistOuSurEt*1, *DistEtSurOu*1, *DeMorgan*1, *DeMorgan*2, *ouCommutativite*, *ouAssociativite*, *etAssociativite*, *etCommutativite*, *DistNegSurEquiv*1, *DistOuSurEquiv*1) *prove*
apply ouAssociativite to expression $\neg\ p \vee \neg\ q \vee q$
simplify
apply ouCommutativite to expression $\neg\ q \vee q$
simplify
apply TiersExclu to expression $q \vee \neg\ q$
simplify
with disabled (*RegleDOr*, *implicationDef*, *DistOuSurEt*1, *DistEtSurOu*1, *DeMorgan*1, *DeMorgan*2, *ouCommutativite*, *ouAssociativite*, *etAssociativite*, *etCommutativite*, *DistNegSurEquiv*1, *DistOuSurEquiv*1) *prove*

theorem TroisPointQuatreVingtOnze
$\forall\, p, q, r : \mathit{BoolGries} \bullet (p \Rightarrow r) \wedge (q \Rightarrow r) = (p \vee q \Rightarrow r)$
proof
use $\mathit{DefAltDeImplication}1[q := r]$
use $\mathit{DefAltDeImplication}1[p := q, q := r]$
use $\mathit{DefAltDeImplication}1[p := p \vee q, q := r]$
use $\mathit{DeMorgan}2$
with disabled ($\mathit{RegleDOr}$, $\mathit{implicationDef}$, $\mathit{DistOuSurEt}1$, $\mathit{DistEtSurOu}1$, $\mathit{DeMorgan}1$, $\mathit{DeMorgan}2$, $\mathit{ouCommutativite}$, $\mathit{ouAssociativite}$, $\mathit{etAssociativite}$, $\mathit{etCommutativite}$, $\mathit{DistNegSurEquiv}1$, $\mathit{DistOuSurEquiv}1$) *prove*
apply $\mathit{RegleDOr}$ *to expression* $\neg p \wedge \neg q$
simplify
apply $\mathit{RegleDOr}$ *to expression* $\neg p \vee r \wedge (\neg q \vee r)$
simplify
apply $\mathit{ouCommutativite}$ *to expression* $\neg q \vee r$
simplify
apply $\mathit{ouAssociativite}$ *to expression* $\neg p \vee r \vee (r \vee \neg q)$
simplify
use $\mathit{lemme}14[p := r, q := \neg q]$
apply $\mathit{ouCommutativite}$ *to expression* $(\neg p \equiv \neg q \equiv \neg p \vee \neg q) \vee r$
simplify
with disabled ($\mathit{RegleDOr}$, $\mathit{implicationDef}$, $\mathit{DistOuSurEt}1$, $\mathit{DistEtSurOu}1$, $\mathit{DeMorgan}1$, $\mathit{DeMorgan}2$, $\mathit{ouCommutativite}$, $\mathit{ouAssociativite}$, $\mathit{etAssociativite}$, $\mathit{etCommutativite}$, $\mathit{DistNegSurEquiv}1$, $\mathit{DistOuSurEquiv}1$, $\mathit{equivAssociativite}1$) *prove*
apply $\mathit{ouCommutativite}$ *to expression* $\neg p \vee (r \vee \neg q)$
simplify
apply $\mathit{equivCommutativite}1$ *to expression* $r \vee \neg q \equiv r \vee \neg q \vee \neg p$
simplify
apply $\mathit{equivAssociativite}2$ *to expression* $\neg p \vee r \equiv$
$(r \vee \neg q \vee \neg p \equiv r \vee \neg q)$
simplify
apply $\mathit{equivCommutativite}1$ *to expression* $\neg p \vee r \equiv r \vee \neg q \vee \neg p$
simplify

theorem TroisPointQuatreVingtOnze (suite de la preuve précédante)
$\forall\, p, q, r : BoolGries \bullet (p \Rightarrow r) \wedge (q \Rightarrow r) = (p \vee q \Rightarrow r)$
proof
apply ouCommutativite to expression $\neg\, p \vee r$
simplify
*apply equivAssociativite*1 *to expression* $r \vee \neg\, q \vee \neg\, p \equiv r \vee \neg\, p \equiv r \vee \neg\, q$
simplify
apply ouAssociativite to expression $r \vee \neg\, q \vee \neg\, p$
simplify
apply ouCommutativite to expression $\neg\, q \vee \neg\, p$
with disabled (*RegleDOr*, *implicationDef*, *DistOuSurEt*1, *DistEtSurOu*1, *DeMorgan*1, *DeMorgan*2, *ouCommutativite*, *ouAssociativite*, *etAssociativite*, *etCommutativite*, *DistNegSurEquiv*1, *DistOuSurEquiv*1, *equivAssociativite*1) *prove*

theorem TroisPointQuatreVingtDouze
$\forall\, p, r : BoolGries \bullet (p \Rightarrow r) \wedge (\neg\, p \Rightarrow r) = r$
proof
apply RegleDOr
simplify
*use DefAltDeImplication*1$[q := r]$
*use DefAltDeImplication*1$[p := \neg\, p, q := r]$
with disabled (*implicationDef*) *prove*
*use lemme*16$[p := r, q := p]$
apply ouCommutativite to expression $p \vee r$
simplify
with disabled (*implicationDef*) *prove*
apply ouCommutativite to expression $p \vee r$
simplify
*use TroisPointQuaranteDeux*2$[p := r, q := p]$
with disabled (*implicationDef*) *prove*

theorem ImplicationMutuelle
$\forall\, p, q : \mathit{BoolGries} \bullet (p \Rightarrow q) \wedge (q \Rightarrow p) = p \equiv q$
proof
use $\mathit{DefAltDeImplication}1$
use $\mathit{DefAltDeImplication}1[p := q, q := p]$
with disabled $(\mathit{implicationDef}, \mathit{fauxDef})$ *prove*
use $\mathit{lemme}16[p := \neg\, p, q := \neg\, q]$
apply DoubleNegation to expression $\neg\, (\neg\, q)$
simplify
with disabled $(\mathit{implicationDef}, \mathit{fauxDef})$ *prove*
apply ouCommutativite to expression $p \vee q$
simplify
apply ouCommutativite to expression $\neg\, p \vee \neg\, q$
simplify
use $\mathit{lemme}16[p := \neg\, q, q := \neg\, p]$
use $\mathit{lemme}16[p := q, q := p]$
apply DoubleNegation to expression $\neg\, (\neg\, p)$
simplify
with disabled $(\mathit{implicationDef}, \mathit{fauxDef})$ *prove*

theorem antisymetrie
$\forall\, p, q : \mathit{BoolGries} \bullet (p \Rightarrow q) \wedge (q \Rightarrow p) \Rightarrow (p \equiv q) = \text{vrai}$
proof
use $\mathit{DefAltDeImplication}1$
use $\mathit{DefAltDeImplication}1[p := q, q := p]$
with disabled $(\mathit{RegleDOr}, \mathit{implicationDef}, \mathit{etCommutativite}, \mathit{etAssociativite}, \mathit{ouAssociativite}, \mathit{ouCommutativite}, \mathit{DistOuSurEquiv}2, \mathit{DistOuSurEquiv}1, \mathit{DeMorgan}1, \mathit{DeMorgan}2, \mathit{DistOuSurEt}1, \mathit{DistEtSurOu}1, \mathit{DistNegSurEquiv}1, \mathit{equivAssociativite}1, \mathit{TroisPointSoixanteQuinze})$ *prove*
apply $\mathit{DistEtSurOu}1$ *to expression* $\neg\, p \vee q \wedge (\neg\, q \vee p)$
simplify
apply etCommutativite to expression $\neg\, p \vee q \wedge \neg\, q$
simplify
apply etCommutativite to expression $\neg\, p \vee q \wedge p$
simplify

theorem antisymetrie (suite de la preuve précédante)
$\forall\, p, q : \mathit{BoolGries} \bullet (p \Rightarrow q) \wedge (q \Rightarrow p) \Rightarrow (p \equiv q) = \text{vrai}$
proof
*apply DistEtSurOu*1 *to expression* $\neg\, q \wedge (\neg\, p \vee q)$
simplify
*apply DistEtSurOu*1 *to expression* $p \wedge (\neg\, p \vee q)$
simplify
apply Contradiction
simplify
use Contradiction$[p := \neg\, q]$
apply DoubleNegation
simplify
*apply ouIdentite*1
simplify
*use ouIdentite*2$[p := p \wedge q]$
use equivDef
apply ouCommutativite to expression $\neg\, q \wedge \neg\, p \vee (\text{faux} \vee (p \wedge q))$
simplify
apply etCommutativite to expression $\neg\, q \wedge \neg\, p$
simplify
use ReflexiviteDeImplication$[p := p \wedge q \vee (\neg\, p \wedge \neg\, q)]$
*apply equivIdentite*1
simplify
with disabled (*RegleDOr*, *implicationDef*, *etCommutativite*, *etAssociativite*, *ouAssociativite*, *ouCommutativite*, *DistOuSurEquiv*2, *DistOuSurEquiv*1, *DeMorgan*1, *DeMorgan*2, *DistOuSurEt*1, *DistEtSurOu*1, *DistNegSurEquiv*1, *equivAssociativite*1, *TroisPointSoixanteQuinze*, *fauxDef*, *equivIdentite*2) *prove*