Le nouveau MUSCADET et la TPTP Problem Library

Dominique PASTRE
Crip5 - Université René Descartes

Résumé :
Après avoir rappelé les principales caractéristiques du démonstrateur MUSCADET et avoir présenté les améliorations de la version 2 de ce système, cet article présente la TPTP Problem Library (Thousands of Problems for Theorem Provers) et les compétitions de démonstrateurs organisées dans le cadre des conférences CADE (Conferences on Automated DEduction) puis présente et discute le travail récent effectué autour de la TPTP Problem Library : analyse des problèmes, enrichissement de MUSCADET, enrichissement de la base TPTP, compétition 1999.

Mots-clés :
Raisonnement mathématique, Démonstration automatique de théorèmes, Systèmes à base de connaissances, Déduction naturelle

1. Introduction

Le démonstrateur MUSCADET [Pastre 1989 et 1993] est un système à base de connaissances qui utilise des méthodes issues de la déduction naturelle, proches de celles utilisées par le mathématicien. Dans la section 2, on compare ces méthodes avec le Principe de Résolution utilisé par la plupart des démonstrateurs.

Des connaissances et savoir-faire mathématiques, élémentaires ou plus spécialisées, sont donnés au système sous forme de règles et métarègles, écrites, interprétées par un moteur d'inférence. Des actions complexes sont décrites par des paquets de règles.

Dans la première version de MUSCADET[1], un langage d'expression des règles et métarègles a été défini et un moteur d'inférence a été écrit en Pascal. La version 2 de MUSCADET, appelée aussi PUSCADET[2] est une réécriture de l'ensemble du système en Prolog (connaissances et moteur d'inférences, ce dernier étant réduit à quelques prédicats Prolog complétant l'interpréteur Prolog). Cette traduction s'est accompagnée de nombreuses améliorations qui sont décrites en sections 3 puis 5, ainsi que les raisons du choix de ce

[1] Moteur Utilisant un Système de Connaissances Adapté à la DEmonstration de Théorèmes.
[2] Prolog Utilisant un Système de Connaissances Adapté à la DEmonstration de Théorèmes *ou* Modélisation Utilisant un Système de Connaissances Adapté à la DEmonstration de Théorèmes version 2.

langage unique pour exprimer toutes les connaissances, qu'elles soient déclaratives pures, procédurales ou intermédiaires.

La section 4 présente la TPTP Problem Library (Thousands of Problems for Theorem Provers) et les compétitions de démonstrateurs organisées dans le cadre des conférences CADE (Conferences on Automated DEduction).

La section 5 présente et discute le travail récent effectué autour de la TPTP Problem Library : analyse des problèmes, enrichissement de MUSCADET, enrichissement de la base TPTP, préparation de la compétition 1999 dont les problèmes et les résultats sont donnés et commentés en section 6.

2. La démonstration automatique de théorèmes (DAT)

Rappelons que le premier programme d'IA est un programme de DAT, c'est le LOGIC THEORIST de Newell, Shaw et Simon, écrit en 1956 pour résoudre des sous-problèmes posés par la réalisation de leur premier programme d'échecs.

Depuis les démonstrateurs utilisent deux grandes familles de méthodes : le Principe de résolution ou les méthodes issues de la Déduction naturelle.

2.1. Principe de résolution et Déduction naturelle

Parmi les programmes anciens, on peut encore citer le programme de Wang (1960) qui commence à avoir de bons résultats et est basé sur le calcul des séquents.

Puis Robinson (1965) révolutionne le domaine avec son "Principe de Résolution" et les méthodes précédentes (calcul des séquents, déduction naturelle) sont abandonnées jusqu'à l'article de Bledsoe (1971) qui montre une meilleure efficacité de la Déduction naturelle, combinée avec le Principe de Résolution. Par la suite, Bledsoe n'utilisera plus que la Déduction naturelle dans son UT theorem prover (1978). Il écrit à cette époque "We have *talked* a lot and proved very *few* hard theorems (by computer) during the last several years. It is time to *do*, to show that our concepts are good. It is time to get a lot more *experience* with our provers. … . One thing that would help push this field ahead, would be for authors to follow the practice of publishing the proof of at least one *hard theorem* in each new methods paper. We do not believe this field will remain vital unless we developp truly powerful provers, and not just theories." (Bledsoe 1977, page 29)

Néanmoins, la plupart des auteurs continuent à travailler avec le Principe de Résolution, en raison de son efficacité théorique supérieure. De nombreuses stratégies ont été écrites pour améliorer la méthode sur le plan pratique, mais pendant de longues années, ce

sont surtout des résultats théoriques qui sont publiés, ainsi que quelques démonstrations trouvées "à la main" ou semi-automatiquement.

Pour beaucoup, DAT est alors quelquefois devenu synonyme de Principe de Résolution. Les clauses ne sont plus seulement des énoncés intermédiaires pour le démonstrateur mais un langage d'expression des problèmes. Les nombreux exemples, difficiles, dans plusieurs domaines mathématiques, suggérés par Wos (1988), sont exprimés en anglais, puis directement dans le "langage de clauses". Il n'y a pas l'intermédiaire du langage du premier ordre. Wos (1988) écrit, à la fin de son livre, "We are […] interested in receiving any proofs of the more difficult test problems […]. If those proofs are expressed in the clause language, we would be delighted." La TPTP Problem Library, dont il sera question plus tard, n'a, pendant les premières années, contenu que des données de problèmes sous forme de clauses.

Les avantages et inconvénients du Principe de résolution et de la Déduction naturelle sont résumés dans la Fig. 1

	Principe de résolution	*Déduction naturelle*
avantages	- simplicité (une seule règle d'inférence) - résultats théoriques	- méthodes et preuves proches de celles de l'être humain - preuves faciles à lire - possibilité d'imiter les heuristiques utilisées par l'homme - mise au point plus facile (raison des échecs, localisation) - limitation des explosions combinatoires
inconvénients	- explosion combinatoire - preuves difficiles à lire - tous les concepts sont sur le même plan - signification peu évidente des fonctions de Skolem - éloignement des quantificateurs de leur portée	- non complètes (sauf Gentzen pure) - peu de possibilité d'études théoriques

Fig. 1. Avantages et inconvénients du Principe de résolution et de la Déduction naturelle

2.2 Exemples de démonstration

On trouvera en Fig. 2, à titre d'exemples, les démonstrations, dans les deux styles, d'un théorème très simple, la transitivité de l'inclusion

$$\forall A\ \forall B\ \forall C\ (A \subset B\ \wedge\ B \subset C\ \Rightarrow\ A \subset C)$$

avec la définition de l'inclusion

$$\forall A\ \forall B\ (A \subset B\ \Leftrightarrow \forall X\ (X \in A \Rightarrow X \in B))$$

On constatera que la démonstration par la déduction naturelle est facile à suivre par un lecteur humain. Au contraire, la signification de la fonction de Skolem, utilisée dans les clauses pour le Principe de Résolution n'est pas immédiate.[1]

Par le Principe de résolution	*Par la déduction naturelle*
Les clauses issues de la définition et de la négation du théorème à démontrer sont les suivantes (1) $\neg A \subset B \vee \neg X \in A \vee X \in B$ (2) $A \subset B \vee f(A,B) \in A$ (3) $A \subset B \vee \neg f(A,B) \in A$ (4) $A_o \subset B_o$ (5) $B_o \subset C_o$ (6) $\neg A_o \subset C_o$ où A_o, B_o et C_o sont des constantes de Skolem et f une fonction de Skolem une preuve[2] : (7) : (1) et (4) $\neg X \in A_o \vee X \in B_o$ (8) : (1) et (5) $\neg X \in B_o \vee X \in C_o$ (9) : (3) et (6) $\neg f(A_o,C_o) \in C_o$ (10) : (2) et (6) $f(A_o,C_o) \in A_o$ (11) : (7) et (10) $f(A_o,C_o) \in B_o$ (12) : (8) et (11) $f(A_o,C_o) \in C_o$ (13) : (9) et (12) $\square$ (clause vide)	hypothèses : $A \subset B$, $B \subset C$ objets : A, B, C conclusion : $A \subset C$ remplacement de la conclusion par sa définition : $\forall X\ (X \in A \Rightarrow X \in C)$ nouvel objet : X nouvelle hypothèse (ajout) : $X \in A$ nouvelle conclusion (remplacement) : $X \in C$ nouvelle hypothèse (ajout) : $X \in B$ (car $A \subset B$ et $X \in A$) nouvelle hypothèse (ajout) : $X \in C$ (car $B \subset C$ et $X \in B$) théorème démontré

Fig. 2. Démonstration de la transitivité de l'inclusion par le Principe de résolution et par la Déduction naturelle

On trouvera par contre dans le paragraphe 5.2.1. un exemple de théorème pour lequel la mise sous forme de clauses est triviale et la démonstration par le Principe de Résolution immédiate, alors que la déduction naturelle demande des stratégies plus élaborées.

[1] La signification de f est la suivante : si A n'est pas inclus dans B, f(A,B) est un élément appartenant à A et non à B (on peut en choisir un si on admet l'axiome du choix); si A est inclus dans B, f(A,B) peut être n'importe quoi.

[2] Avec la stratégie de l'ensemble support, la plus fréquemment utilisée dans le domaine mathématique

3. Démonstrateurs écrits par l'auteur

Les principales idées ayant conduit à l'écriture de la première puis de la deuxième version de MUSCADET étaient déjà présentes dans un ancien système appelé DATTE[1] [Pastre 1978]. Les progrès effectués depuis sont dûs en grande partie à l'analyse des nombreuses expérimentations et à des améliorations techniques.

3.1. DATTE

C'était un programme écrit en ... Fortran

Ses points forts :

- quelques réécritures et des règles
- une construction automatique de règles (avec les mêmes réécritures que pour les démonstrations)
- un traitement spécifique et efficace des hypothèses existentielles et des hypothèses disjonctives
- un graphe pour représenter les relations binaires
- une stratégie de démonstration (sous forme d'un plan programmé)

Ces idées sont fondamentales et sont toujours mises en oeuvre dans les démonstrateurs écrits depuis. Les progrès accomplis ont été des généralisations, des compléments et beaucoup d'améliorations techniques.

Ses points faibles :

- la syntaxe : pas de symboles fonctionels, notation polonaise préfixe, symboles sur 3 caractères !
- des programmes (on commençait à parler de règles, mais on ne parlait pas encore vraiment de bases de connaissances à l'époque ...)

Résultats

Le programme a eu de bons résultats dans les domaines suivants de théorie des ensembles : opérations ensemblistes, applications, ensembles images et images réciproques, relations d'équivalence et relations d'ordre, ordinaux.

3.2. MUSCADET

C'est un système à base de connaissances comportant :

- des bases et sous-bases de faits (théorèmes et sous-théorèmes) construites dynamiquement
- des règles et métarègles

[1] Démonstration Automatique de Théorèmes en Théorie des Ensembles

- un moteur d'inférence écrit en en PL1 puis réécrit en Pascal
- la définition d'un langage de règles

Ses points forts :

Un langage de règles, unique, se voulant déclaratif, est utilisé pour exprimer :
- des connaissances mathématiques générales
- des connaissances spécifiques à des domaines particuliers
- des "savoir-faire"
- des stratégies de démonstration et de gestion de la démonstration
- des "super-actions" définies par des paquets de règles interprétées par le même moteur d'inférence
- des manipulations formelles
- la construction de règles à partir des définitions et des lemmes par des métarègles
- la gestion de sous-bases (sous-théorèmes issus de découpages) et le transfert d'information d'une (sous-)base à une autre

De plus :
- l'ordre des règles est en principe quelconque
- on a une structuration analogue des faits et des règles
- on peut utiliser des symboles fonctionnels et une "mise à plat" automatique donne des noms à tous les objets définis par des symboles fonctionnels et les manipulations sont ensuite faites comme s'ils étaient définis par des prédicats. En particulier un quantificateur "pour le seul ... tel que ..." permet de gérer correctement des transformations en étant traité, suivant le contexte comme un quantificateur existentiel ou comme un quantificateur universel.

Ses points faibles :

- syntaxe : c'est mieux mais on a encore une notation de listes préfixées
- peu de manipulations formelles (le système ne connaît même pas les nombres, il est seulement capable de faire des comparaisons lexicographiques et possède quelques règles de réécritures adhoc)
- certains paquets de règles sont assez illisibles car le langage n'est pas adapté à des connaissances et savoir-faire procéduraux
- les utilisateurs ne peuvent (ne veulent) pas modifier le moteur
- la gestion des sous-bases (sous-théorèmes) est assez lourde
- les découpages sont couteux en temps
- le ramasse-miettes, qui a été écrit trop tard, se fait quelque fois à un mauvais moment
- il est quelquefois indispensable de savoir comment le moteur fonctionne pour comprendre le comportement du système
- on n'a pas résisté à la tentation d'utiliser quelques trucs pour forcer le système à faire ce qu'on veut
- quelques bogues sont quelquefois impossibles à trouver

Utilisation :

MUSCADET a été conçu pour donner des connaissances et savoir faire spécifiques. Il a été utilisé par l'auteur sur les Espaces vectoriels topologiques et en Géométrie discrète. Il a démontré quelques théorèmes difficiles. Mais on peut considérer que le système a été aidé par la donnée de connaissances qui, bien que générales, étaient adaptées aux théorèmes traités. MUSCADET a d'autre part été utilisé par des stagiaires de DEA, par des chercheurs étrangers, ainsi que par deux thésards (Bazin(1993) et Spagnol (1999)).

3.3. MUSCADET version 2 (alias **PUSCADET**)

Il s'agit d'une traduction en Prolog, accompagnée de nombreuses améliorations, de MUSCADET

Les **motivations** qui ont conduit à cette traduction sont les suivantes :

- pouvoir exprimer dans le même langage du déclaratif pur, du prodédural pur, et tous les intermédiaires;
- utiliser les possibilités syntaxiques (notations infixes) et calculatoires (nombres) de Prolog;
- donner aux utilisateurs la possibilité d'avoir accès à tout le système, de rajouter des modules (par exemple de calcul formel, traitement du ou variaire)
- tout est alors explicite ou fait par l'interpréteur Prolog (relativement) standard.

On a mis l'accent sur l'expression la plus naturelle (humaine ?) des connaissances plutôt que sur l'aspect déclaratif.

Ces facilités constituent néanmoins un **piège**, car, en tant qu'utilisateurs déformés par des années de programmation classique, la tentation est très forte de programmer. Tout est alors encore explicite, mais peut devenir illisible.

MUSCADET 2.0 (version 1997 de PUSCADET) a facilité le travail en géométrie discrète, mais il était encore nécessaire de donner des règles adhoc de manipulations formelles. Un module de calcul formel, écrit par un chercheur invité, a résolu certains des problèmes.

Puis PUSCADET a été complété (la version 1999 a été officiellement appelée **MUSCADET 2.1**) pour travailler sur la base TPTP et participer à la compétition de démonstrateurs CASC-16.

3.4. Exemples d'énoncés

3.4.1. Transitivité de l'inclusion

Les expressions dans les trois systèmes du théorème exprimant la transitivité de l'inclusion

$$A \subset B \wedge B \subset C \Rightarrow A \subset C$$

et de la définition de l'inclusion

$$A \subset B \Leftrightarrow \forall X\,(X \in A \Rightarrow X \in B)$$

se trouvent en Fig. 3.

DATTE
Théorème : `TH IMP ET . INC A0 B0 . INC B0 C0 . INC A0 C0`
Définition : `DEF . INC A B QQS X IMP . APP X A . APP X B`

MUSCADET
Théorème : `(IMP (ET (INC A B) (INC B C)) (INC A C))`
Définition : `(<=> (INC A B)(QQS X (=> (APP X A)(APP X))))`

PUSCADET
Théorème : `a inc b et b inc c => a inc c`
ou : `qqs(A, qqs(B, A inc B et B inc C => A inc C)`
ou (syntaxe de la TPTP library) :
`![A,B]: (subset(A,B) & subset(B,C) => subset(A,C))`
Définition : `A inc B <=> qqs(X,(X app A => X app B)))`
ou : `qqs(A,qqs(B, A inc B <=> qqs(X,(X app A => X app B))))`
ou (syntaxe de la TPTP Library) :
`![A,B]:(subset(A,B) <=> !X: (member(X,A) => member(X,B)))`

Fig. 3. Expressions dans les trois systèmes du théorème exprimant la transitivité de l'inclusion

On peut aussi exprimer ce théorème en utilisant des énoncés du deuxième ordre. La Fig. 4 montre un tel énoncé. Mais Prolog n'accepte pas une telle notation avec des prédicats variables. Cet énoncé doit être traduit, par un simple traitement de texte dans l'énoncé de la Fig. 5.

Théorème : `transitive(inc)`
Définition :
```
transitive(R)<=>
   qqs(X,qqs(Y,qqs(Z,R(X,Y) et R(Y,Z) => R(X,Z))))
```

Fig. 4. Enoncé du deuxième ordre

Théorème : `transitive(inc)`
Définition :
```
transitive(R)<=>
   qqs(X,qqs(Y,qqs(Z,..[R,X,Y]et..[R,Y,Z]=>..[R,X,Z])))
```

Fig. 5 Enoncé du "deuxième ordre" pour Prolog

On voudrait alors pouvoir unifier `r(a,b)` et `..[R,X,Y].` Mais l'unification standard ne le fait pas. Un traitement automatique du symbole "`..`" permet d'y remédier. Par exemple, les

métarègles ne génèrent pas des conditions de la forme `hyp(N,..[R,X,Y])` mais à la place la condition `hyp(N,H),H =.. [R,X,Y]`[1]

3.4.2. Autre exemple

La Fig. 6 contient les énoncés pour le théorème suivant de théorie des ensembles :

Théorème $A \subset A' \wedge B \subset B' \Rightarrow A \cap B \subset A' \cap B'$

Définition $A \cap B = \{X \mid X \in A \wedge X \in B)$

DATTE

```
DEF INT C A B QQS X  EQ . APP X C ET . APP X A . APP X B
TH IMP ET . INC A0 A1 ET . INC B0 B1
          ET INTER C0 A0 B0 INTER C1 A1 B1
        . INC C0 C1
```

MUSCADET

Définition : `(= (INTER A B) < X ∧(ET (APP X A)(APP X)) >)`[2]

Théorème : `IMP (ET (INC A A1) (INC B B1))`
`(INC (INTER A A1) (INTER B B1))`

hypothèses : `(:(INTER A A1)$1)` conclusion : `(INC $1 $2)`
`(:(INTER B B1)$2)`

PUSCADET

Définition : `A inter B = [X,X app A et X app B]` [3]
ou `X app A inter B <=> X app A et X app B`
ou
```
![A,B,X]: (member(X,intersection(A,B))
                <=> member(X,A) & member(X,B))
```

Théorème : `a inc a1 et b inc b1 => a inter a1 inc b inter b1`
hypothèses : `a inter a1 : a_inter_a1`
`b inter b1 : b_inter_b1`
conclusion : `a_inter_a1 inc b_inter_b1`

ou (usage pour la TPTP library)

Théorème :
```
![A,B,A1,B1]: (subset(A,A1) & subset(B,B1)
 => subset(intersection(A,A1),intersection(B,B1)))
```
hypothèses : `x inter x1 : x_inter_x1`
`x2 inter x3 : x2_inter_x3`
conclusion : `x_inter_x1 inc x_inter_x1`

Fig. 6. Expressions dans les trois systèmes d'un théorème de théorie des ensembles

[1] Il s'agit du symbole `=..` prédéfini en Prolog. On a `r(a,b) =.. [r,a,b].` On n'a pas `r(a,b) = ..[r,a,b]`, mais `r(a,b) =.. H, H = ..[r,a,b].` En fait, le choix du symbole `..`, fait pour que les notations se ressemblent, est plutôt source de difficulté !

[2] < > au lieu de { } à cause des claviers des années 80 …

[3] [ ], notation de liste au lieu de { } pour faciliter la lecture et le traitement

L'inconvénient des énoncés universels clos, pour la lisibilité de la trace, est que tous les objets créés, à partir de variables quantifiées, s'appellent x, x1, x2, ... C'est pourquoi, en dehors de la TPTP Library, on préfère donner des énoncés avec des constantes (ce qui correspond à la première étape de la skolemisation de la négation du théorème pour le Principe de résolution). MUSCADET peut d'autre part recevoir des énoncé universels non clos pour les définitions et les lemmes mais pas pour les théorèmes à démontrer.

3.5. Exemples de démonstrations

3.5.1.

La Fig. 7 donne un résumé de la démonstration par MUSCADET du théorème précédent :

$$A \subset A' \ \wedge \ B \subset B' \ \Rightarrow \ A \cap B \ \subset \ A' \cap B'$$

hypothèses *(ajoutées par des déductions successives)*	**conclusion** *(chaque conclusion remplace la précédente)*
	$A \subset A' \wedge B \subset B' \Rightarrow A \cap B \subset A' \cap B'$
Règle "élim_fonc" (donne des noms aux termes apparaissant dans l'énoncé)	
C: $A \cap B$	
C': $A' \cap B'$	$A \subset A' \wedge B \subset B' \Rightarrow C \subset C'$
règle "⇒" (traite les conclusions de la forme H ⇒ C)	
$A \subset A'$	$C \subset C'$
$B \subset B'$	
règle "def_concl_1" (définition de la conclusion)	
	$\forall x (x \in C \Rightarrow x \in C')$
règles "∀" et "⇒" (la règle "∀" traite les conclusions universelles)	
$x1 \in C$	$x1 \in C'$
règles "∩11" et "∩12" (règles construites à partir de la définition de l'intersection)	
$x1 \in A$	
$x1 \in B$	
règle "⊂" (règle construite à partir de la définition de l'inclusion)	
$x1 \in A'$	
$x1 \in B'$	
règle "∩2" (règle construite à partir de la définition de l'intersection)	
$x1 \in C'$	
règle "stop" (règle d'arrêt, on vient d'obtenir en hypothèse la conclusion que l'on voulait démontrer)	

Fig. 7. Résumé d'une démonstration de MUSCADET

3.5.2.

Soient f, g, h trois applications de A dans B, B dans C, C dans A. Si, parmi les trois applications $h_o g_o f$, $g_o f_o h$, $f_o h_o g$; deux sont injectives (resp. surjectives) et la troisième surjective (resp. injective), alors f, g et h sont bijectives.

Il y a en fait six théorèmes indépendants à démontrer : la bijectivité de chacune des trois applications f, g et h dans chacun des deux cas (deux composées injectives et une surjective ou l'inverse)

MUSCADET démontre ces théorèmes en créant des images et des antécédants jusqu'à obtenir tous ceux qui sont nécessaires à une démonstration directe[1]. Il crée à peu près autant d'éléments inutiles que d'éléments nécessaires. On trouvera en Fig. 8. les créations nécessaires pour la démontration de la bijectivité de h dans le cas où $h_o g_o f$ est injective et les deux autres composées surjectives.

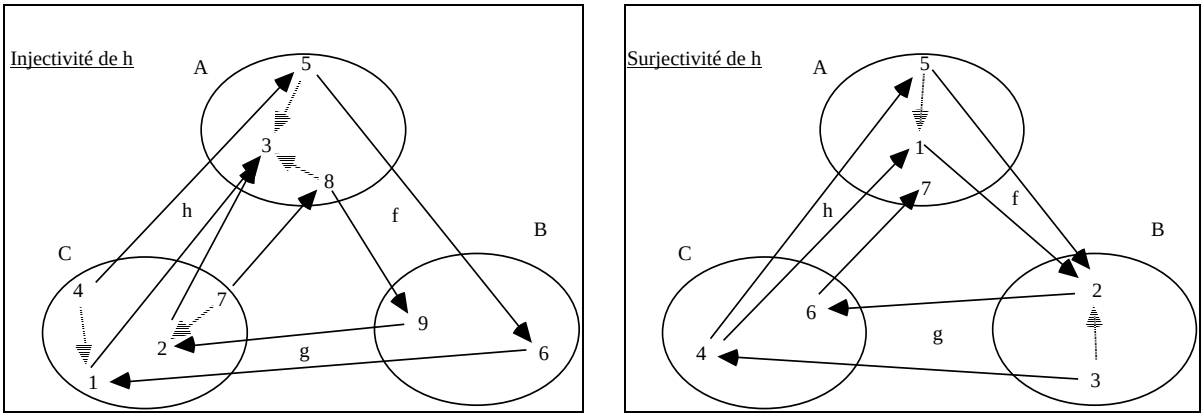

Fig. 8. Objets créés dans la démontrattion d'un théorème sur les applications

3.6. Exemples de règles et de super-actions

On trouvera en Fig. 9. et 10. des exemples de règles et super-actions en MUSCADET et en PUSCADET.

Le paramètre N, qui n'existait pas dans la première version de MUSCADET sert à reprérer la sous-base (sous-théorème) sur lequel on est en train de travailler.

On notera l'utilisation importante du "si ... alors ... sinon ..." (en Prolog (... -> ... ; ...)), moins déclarative mais plus naturelle dans certains cas (par exemple `ajhyp`) que les règles en vrac.

Le traitement des hypothèses disjonctives est un exemple montrant également comment un peu moins de déclarativité donne plus de naturel.

---

[1] DATTE démontrait déjà ces théorèmes, avec cependant une petite facilité supplémentaire : il recevait la définition de la composée de *trois* fonctions, ce qui évitait de créer quelques éléments inutiles. On trouvera dans (Pastre 78) les démontrations détaillées de deux des théorèmes (un des plus faciles et un des plus difficiles)

MUSCADET

```
REGLE STOP1 SI CONCL C HYP C ALORS NOUVCONCL VRAI

REGLE => SI CONCL(=>A B)ALORS AJHYP A NOUVCONCL B

POUR NOUVCONCL C REGLE NOUVCONCL1 ELIFON C AFFECTER(CONCL C)
                                  message(nouvelle conclusion C)

POUR AJHYP H REGLE AJHYP1 SI PREMIER(H ET) ELTLISTE(A H) NOMDIF(A ET)
                          ALORS AJHYP A
             REGLE AJHYP4 SI EGAL(H(=X Y))NOMDIF(X Y) NONHYP H
                          ALORS AJOUTER(HYP H) message(ajouter hypothese H)
                                ajobj H STOP
             REGLE AJHYP5 SI PREMIER(H P) NOMDIF(P ET = QQS SEUL) NONHYP H
                          ALORS AJOUTER(HYP H) message(ajouter hypothese H)
                                ajobj H STOP
             REGLE AJHYP6 SI ATOME H NONHYP H
                          ALORS AJOUTER(HYP H) message(ajouter hypothese H) STOP
             REGLE AJHYP7a SI EGAL(H(SEUL (:A X)Y)) HYP(:A X1)
                           ALORS LIGNE message(remplacer X par X1 dans Y)
                                 REMPLACER(Y X X1) AJHYP Y SUPHYP H STOP
             REGLE AJHYP7b SI EGAL(H(SEUL (:A X)Y)) NONHYP(:A X1) NONOBJ X
                           ALORS AJHYP (:A X) AJHYP Y SUPHYP H
                                 AJOUTER(OBJ X) message(ajouter objet X) STOP
             REGLE AJHYP7c SI EGAL(H(SEUL (:A X)Y)) NONHYP(:A X1) OBJ X
                           ALORS CREER X AJHYP (:A X) AJHYP Y SUPHYP H
                                 AJOUTER(OBJ X) message(ajouter objet X) STOP
             REGLE AJHYP8 SI EGAL(H(QQS(APP X A)C))
                          ALORS CREER REGHYP
                                CONSREG((REGHYP) H REGLACTIV)
                                ORDOREG STOP
             REGLE AJHYP9 SI EGAL(H(QQS X A)) ATOME X
                          ALORS CREER REGHYP
                                CONSREG((REGHYP) A REGLACTIV)
                                ORDOREG STOP

REGLE ET SI CONCL C PREMIER(C ET)ELTLISTE(A C)NOMDIF(A ET)ALORS DEM A

POUR DEM A REGLE DEM1 SI NUMERO N
            ALORS CREER N AJOUTER(SOUSTH((NUMERO N)(CONCL A)))
                message(ajouter le sous theoreme de numero N et de conclusion A)
                 COPITEM DEMSOUSTH RETOURDEM A
                 SUPPRIMER SOUSTH message(supprimer le sous theoreme N)
```

Fig. 9. Exemples de règles et super-actions en MUSCADET

La règle de MUSCADET

```
REGLE OU SI CONCL C HYP H EGAL(H(OU A B))
         ALORS SUPHYP H NOUVCONCL(ET(=>A C)(=>B C))
```

qui donne, après découpage par la règle `ET` et l'application de la règle `=>` deux sous-théorèmes ayant comme hypothèse supplémentaire, respectivement `A` et `B`, a d'abord été traduite en PUSCADET par la règle suivante faisant le même travail :

PUSCADET

```
regle(N,stop1):- concl(N,C), hyp(N,C), nouvconcl(N,vrai).

regle(N,=>) :- concl(N, A => B), ajhyp(N, A), nouvconcl(N,B).

nouvconcl(N, C) :- not concl(N, C), retractall(concl(N, _)),
          assert(concl(N, C)),
          ecrire1([N, nouvelle,conclusion, C]).

ajhyp(N, H) :- ( H = A et B -> ajhyp(N, A), ajhyp(N,B)
       ; hyp(N, H) ->  true
       ; H = (X = X) -> true
       ; H = ..[R, X, Y]  -> H1 =..[R, X, Y], ajhyp(N, H1)
       ; H = ..[F, X]:Y  -> Y1 =..[F, X], ajhyp(N, Y1:Y)
       ; H = seul(A:X,Y) -> (hyp(N,A:X1) -> ecrire1([remplacer,X,par,X1,dans,Y])
                             ; creer_objet(N,x,X1),ajhyp(N,A:X1)
                             ),
                             remplacer(Y,X,X1,Y1),ajhyp(N,Y1)
       ; H = non seul(FX:Y,P) -> ajhyp(N,seul(FX:Y,non P ))
       ; H = qqs(_, _) -> creer_nom_regle(reghyp,Nom),
        consreg( H, _, N, Nom, [])
       ; H = A => B -> creer_nom_regle(reghyp,Nom),
        consreg( H, _, N, Nom, [])
       ; assert(hyp(N, H)),
         ecrire1([N, ajouter,hypothese,H])
       ).

regle(N, et) :- concl(N, A et B), demconj(N, 0, A et B), concl(N, vrai).

demconj(N, I, A) :- (A = B et C -> dem(N, I, B), I1 is I+1, demconj(N, I1,C)
                    ; dem(N, I, A)
                    ).
dem(N, I, A) :-  I1 is I+1,  N1 is 10*N+I1,
        nl,ecrire1([**********************************************]),
        ecrire([sous-theoreme, N1,*****]),
        nouvconcl(N1,A), copitem(N, N1),
        assert(sousth(N, N1)),
        nl,ecrire([-----------------------------------------------]),
        ecrire([creation,du,sous-theoreme,N1]),
        appliregactiv(N1),
        (concl(N1, vrai) -> concl(N, C), retirer(A, C, C1),
                            nouvconcl(N, C1)
        ; true).
```

Fig. 10. Exemples de règles et super-actions en PUSCADET

```
regle(N, ou) :- hyp(N, A ou B), not hyp_traite(N, A ou B), [1]
                concl(N, C),
                ecrire1([N, traitement,de,l,hypothese,disjonctive,A ou B]),
                nouvconcl(N, (A => C) et (B => C)),
                ajhyp_traite(N, A ou B).
```

[1] Le prédicat `hyp_traite` permet une mémorisation qui a dû être ajoutée dans de nombreux cas pour éviter que l'hypothèse soit traitée à nouveau. Ceci était en général inutile dans MUSCADET, car le moteur d'inférence de MUSCADET n'appliquait jamais une règle deux fois pour les mêmes instanciations. D'une part, cela avait, dans quelques cas, posé des problèmes qui ont été résolus en ajoutant des conditions ne servant qu'à provoquer une instanciation différente. D'autre part, l'interpréteur Prolog ne donne pas les instanciations ayant permis les diverses unifications qu'il fait, et garder ce test aurait nécessité de réécrire l'unification. Il a été jugé préférable de mémoriser les éléments traités et de tester dans la règle que l'hypothèse n'a pas encore été traitée. (Cette condition pourrait être ajoutée automatiquement.)

Puis un théorème du type suivant a soulevé un problème :

On a les deux hypothèses disjonctives suivantes : `a1 ou a2 ou a3` et `b1 ou b2 ou b3` .

Un premier découpage (première hypothèse disjonctive) donne les sous-théorèmes suivants dont on a indiqué seulement les hypothèses non encore traitées :

1	2
b1 ou b2 ou b3 a1	b1 ou b2 ou b3 a2 ou a3

puis le découpage de la deuxième hypothèse disjonctive donne

11	12	21	22
a1 b1	a1 b2 ou b3	a2 ou a3 b1	a2 ou a3 b2 ou b3

etc.

Finalement, on aura :

11	121	122	211	212	2211	2212	2221	2222
a1 b1	a1 b2	a1 b3	b1 a2	b1 a3	a2 b2	a2 b3	a3 a2	a3 b3

Le mélange des cas est encore pire si on a une troisième hypothèse disjonctive.

On préfère avoir le découpage suivant, plus naturel :

1	2	3
b1 ou b2 ou b3 a1	b1 ou b2 ou b3 a2	b1 ou b2 ou b3 a3

puis

11	12	13	21	22	23	31	32	33
a1 b1	a1 b2	a1 b3	a2 b1	a2 b2	a2 b3	a3 b1	a3 b2	a3 b3

Ce découpage sera obtenu par la règle suivante :

```
regle(N, ou) :- hyp(N, A ou B), not hyp_traite(N, A ou B),
                concl(N, C),
                ecrire1([N, traitement,de,l,hypothese,disjonctive,A ou B]),
                hypou(A ou B => C , T),  % decoupage de tous les ou
                nouvconcl(N,T),
                ajhyp_traite(N, A ou B).
```

utilisant le prédicat Prolog `hypou`, procédural et récursif [1]

[1] immédiat à écrire pour toute personne ayant une petite habitude de la programmation Prolog

```
hypou(A ou B => C, (A => C) et BC) :- hypou(B => C, BC),!.
hypou(T,T).
```

qui construit directement l'énoncé `T = (A1 => C ) et (A2 => C) et (A3 => C) et ...` à partir d'une hypothèse `A1 ou A2 ou A3 ou ... .`

Une amélioration (mais ce n'est pas la plus urgente) consisterait à ce que MUSCADET comprenne les "…" dans une notation de la forme `A ou B ou …` et construise automatiquement ce prédicat Prolog. Les notations avec des "…" sont en effet très courantes en mathématiques.

4. The TPTP Problem Library

La TPTP Problem Library (Thousands of Problems for Theorem Provers) est une base de problèmes conçue et maintenue par Geoff Sutcliffe (Australie) et Christian Suttner (Allemagne) depuis 1993 pour tester et évaluer des systèmes de démonstration automatique de théorèmes. Elle se trouve sur le Web à l'adresse http://www.cs.jcu.edu.au/~tptp

4.1. La base et les problèmes

Jusqu'en 1996, la base ne comportait que des énoncés sous forme de clauses (format CNF : Clause Normal Form). Depuis 1997, elle comporte également des énoncés du calcul des prédicats du premier ordre (format FOF : First Order Formula).

En 1998,	il y avait	3275	problèmes en format	CNF, dont	2296 abstraits et 86 génériques,
		347		FOF	332 1
1999,		3334		CNF	2352 86
		670		FOF	632 1

Un même problème abstrait peut être exprimé sous diverses formulations et donner ainsi plusieurs problèmes. Un problème générique est un problème général (qui n'est pas du premier ordre) dont quelques exemples (d'ordre 0 ou 1) correspondant à différentes tailles du problème figurent dans la base, par exemple, le théorème générique

(p1 <=> (p2 <=>...(pN <=> (p1 <=> (p2 <=>...<=> pN)...)

a deux instances dans la base, correspondant à N = 5 et N=14 :

```
p1<=>(p2<=>(p3<=>(p4<=>(p5<=>(p1<=>(p2<=>(p3<=>(p4<=>p5)))))))
```

et

```
p1<=>(p2<=>(p3<=>(p4<=>(p5<=>(p6<=>(p7<=>(p8<=>(p9<=>(p10<=>(p11<=>(p12<=>(
p13<=>(p14<=>(p1<=>(p2<=>(p3<=>(p4<=>(p5<=>(p6<=>(p7<=>(p8<=>(p9<=>(p10<=>(
              p11<=>(p12<=>(p13<=>p14))))))))))))))))))))))))))))
```

Les problèmes relèvent de 28 domaines correspondant à 5 grands thèmes présentés dans la Fig. 11, ainsi que le nombre de problèmes, pour chaque domaine, en 1999.

	domaines	*format FOF*	*format CNF*
Logique	Logique combinatoire Calcul logique Modèles de Henkin	2	279 (257)
Mathématiques	**Théorie des ensembles (SET)** Théorie des graphes Algèbre : Groupes + (Boole, Robbins, Distr, Treillis, Anneaux, Algèbre générale) Théorie des nombres Topologie Analyse Géométrie Théorie des corps Théorie des catégories	**321** (300) [**5** en 98] 1	696 (561) [695 (562) en 98] 374 (203, 50) [363 (193) en 98]
Informatique	**Informatique théorique (COM)** Représentation des connaissances Planning Vérification de programmes	3 (1)	6 (4)
Engineering	Conception de circuits Vérification de circuits		
Sciences sociales	**Management (MGT)**	**53** (41)	0
Autre	**Syntaxique (SYN)** Puzzles Miscellaneous	**286** (283, 1) [280 (279, 1) en 98] 3 [2 en 98] 1	479 (469, 27) 57 (36, 4) 13 (9, 3)

Fig. 11. Nombre des problèmes dans les différents thèmes et domaines de la TPTP Problem Library en 1999

n1 (n2, n3) signifie qu'il y a n1 problèmes dans la base correspondant à n2 problèmes abstraits et n3 problèmes génériques. On n'a indiqué ces nombres uniquement pour les domaines où il existe des problèmes en format FOF.

On trouvera en annexes 1 à 4 des exemples de théorèmes dans les domaines SYN, MGT et COM. Le domaine SET sera étudié en section 5.2.1.

4.2. Les compétitions

Elles sont organisées depuis 1996 par Geoff Sutcliffe (Australie) et Christian Suttner (Allemagne) dans le cadre des conférences CADE (Conference on Automated DEduction). Elles comportent quatre divisions dont certaines comportent plusieurs catégories (Fig. 12).

Divisions	*Catégories*	*clauses de Horn*	*Egalité*
MIX : format CNF, sans égalité unitaire	HEQ HNE NEQ NNE PEQ	oui oui non non	oui non oui non oui, pure
UEQ : format CNF, avec égalité unitaire			
SAT : format CNF, non théorèmes (cad ensembles de clauses satisfaisables)			
FOF : format FOF	FEQ FNE		oui non

Fig. 12. Divisions et catégories des compétitions

Les logiciels doivent être installés sur les machines du site de la compétition une à deux semaines avant, puis passer des tests effectués par les organisateurs, c'est-à-dire être capables de démontrer des théorèmes faciles et être valides (ne pas démontrer des théorèmes qui n'en sont pas). Des tests de validité sont encore effectués après la compétition pour les vainqueurs de chaque catégorie.

Le classement se fait en fonction du nombre de problèmes résolus. En cas d'égalité, en fonction des temps d'exécution.

En 1998, 18 démonstrateurs ont participé à la compétition, dont 3 dans la division FOF. La Fig. 13 donne les résultats, pour ces 3 concurrents, pour toutes les divisions où ils ont participé.

On remarquera les bons résultats d'Otter en format CNF, mais non en format FOF, ce qui montre, si cela était encore nécessaire, que la mise automatique sous forme d'un bon ensemble de clauses n'est pas facile.

Les 40 théorèmes de la division FOF comprenaient 13 MGT, 25 SYN, 1 COM et 1 PUZ. PUSCADET en a démontré 8 (de la catégoire FEQ, avec égalité).

D'autre part, Bliksem a par la suite été disqualifié, car ayant été trouvé invalide au cours des tests post-compétition.

	Concurrents	SPASS	Bliksem	Otter
MIX	9	3ème		6ème
HEQ		2ème		**1er**
HNE		3ème		5ème
NEQ		3ème		4ème
NNE		**1er**		2ème
PEQ		6ème		7ème
UEQ	8	6ème		2ème
SAT	4	**1er**		
FOF	3 **40** problèmes	**1er** **39** résolus	2ème **21** résolus	3ème **2** résolus
FEQ	3 **20** problèmes	2ème **19** résolus	**1er** **19** résolus	3ème **2 résolus**
FNE	3 **20** problèmes	**1er** **20** résolus	2ème **2** résolus	3ème **0** résolu

Fig. 13. Résultats de la compétition 1998, de SPASS, Bliksem et Otter

5. PUSCADET et la base TPTP

Plusieurs phases de travail ont été nécessaires pour permettre à PUSCADET de participer raisonnablement à la compétition 1999. Il a d'abord fallu poursuivre la traduction de MUSCADET (et même de DATTE) en Prolog, puis analyser la base TPTP dans l'optique de l'utilisation de MUSCADET, expérimenter et ajouter ou modifier des connaissances pour traiter des théorèmes de la base qui n'étaient pas démontrés, proposer de nouveaux théorèmes (dans la syntaxe et l'esprit de la base) pour enrichir la base, enfin mettre au point un exécutable respectant les contraintes de la compétition.

5.1. Poursuite de la traduction en Prolog

La traduction ayant été commencée dans le cadre du travail en géométrie discrète, de nombreuses spécificités de MUSCADET n'avaient pas été incorporées, ou bien seulement dans un forme simplifiée. Certaines possibilités de DATTE, même, n'avaient jamais été traduites en MUSCADET ni en PUSCADET.

Actuellement, tout le "programme" DATTE est réalisé en PUSCADET, et tous les théorèmes démontrés par DATTE le sont par PUSCADET, sauf ceux concernant les ordinaux qui nécessitent des lemmes et qui n'ont pas été tous testés. Le démonstrateur général MUSCADET est entièrement traduit sauf le traitement des conclusions existentielles dans le cas le plus complexe. Les connaissances spécifiques à certains domaines (par exemple Espaces vectoriels topologiques) n'ont pas été traduits. Ces connaissances étaient données directement dans le langage de MUSCADET sous forme opérationnelle. L'orientation future sera plutôt de les donner uniquement par des énoncés du premier ordre et d'écrire des métarègles les traduisant dans les formes opérationnelles. L'aide donnée au démonstrateur sera ainsi plus claire et de tels problèmes pourront rentrer dans la base TPTP.

5.2. Analyse de la base et expérimentations de PUSCADET

Les essais ont été faits sur les seuls domaines ayant des énoncés dans le format FOF. L'analyse a porté sur les domaines SET, MGT, SYN et COM.

5.2.1. Théorie des ensembles

Le domaine SET ne comportait en 1998 que cinq théorèmes dont quatre tournent autour des paradoxes de la théorie des ensembles et le cinquième est un théorème facile sur l'égalité d'ensemble.

1 - *il n'existe pas d'ensemble formé des éléments qui n'appartiennent pas à eux-mêmes*

$$\neg\ \exists X\ \forall Y\ (Y \in X \Leftrightarrow Y \notin Y) \qquad \text{(SET043+1)}$$

2 - *s'il existe un ensemble formé des éléments qui appartiennent à eux-mêmes, alors il n'est pas vrai que tout ensemble a un complément*

$$\exists Y\ \forall X\ (X \in Y \Leftrightarrow X \in X)\ \Rightarrow\ \neg\ \forall X1\ \exists Y1\ \forall Z\ (Z \in Y1 \Leftrightarrow Z \notin X1) \qquad \text{(SET044+1)}$$

3 - si on a l'axiome *pour tout Z il existe un ensemble formé des éléments qui appartiennent à Z et n'appartiennent pas à eux-mêmes,* alors on a le théorème *il n'existe pas d'ensemble universel* (SET045+1)

axiome : $\forall Z\ \exists Y\ \forall X\ (X \in Y \Leftrightarrow (X \in Z \wedge X \notin X))$ théorème : $\neg\ \exists Z\ \forall X (X \in Z)$

4 - *il n'existe pas d'ensemble formé des éléments X tels qu'il n'existe pas de chaîne $X \in Z \in X$*

$$\neg\, \exists Y\ \forall X\ (X \in Y \Leftrightarrow \neg\, \exists Z\ (X \in Z \wedge Z \in X)) \qquad \text{(SET046+1)}$$

5 - *l'égalité d'ensemble est une relation symétrique*

axiome : $\forall X\ \forall Y\ (X =_{ens} Y \Leftrightarrow \forall Z\ (Z \in X \Leftrightarrow Z \in Y))$

théorème : $\forall X\ \forall Y\ (X =_{ens} Y \Leftrightarrow Y =_{ens} X)$ (SET047+1)

Or, dans tout le travail que j'avais fait précédemment, les définitions ensemblistes utilisaient des propriétés relatives à des ensembles (ce qui est un moyen, en théorie naïve, de ne pas avoir de paradoxes). Ainsi PUSCADET savait traiter des énoncés de la forme

$\forall X\ (X \in A \Rightarrow p(X))$

ou $\forall X \in A\ p(X)$ avec des quantificateurs relativisés,

ainsi que $\forall X\ (q(X) \Rightarrow p(X))$

qui donnaient des règles de la forme

si hyp $X \in A$ *alors ajhyp* $p(X)$

si hyp $q(X)$ *alors ajhyp* $p(X)$

instanciant *X* en partie <conditions>.

Mais PUSCADET ne savait pas traiter des énoncés de la forme

$\forall X\ p(X)$

Ce problème a été résolu en utilisant, dans la construction des règles, le fait d'être un objet mathématique (déjà présent pour mémoriser tous les objets mathématiques créés en cours de démonstration), uniquement s'il n'y a pas d'autre condition instanciant X (car s'il y a d'autres conditions, elles sont certainement plus sélectives). On a ainsi la règle

si objet X alors ajhyp p(X)

pour le premier cas, et pas de changement pour les règles construites dans les autres cas.

Quatre des cinq théorèmes ont alors été démontrés par la version standard, le deuxième (SET044+1) ne l'est que par une variante obtenue en modifiant l'ordre de traitement des hypothèses existentielles et disjonctives.

Le premier théorème (SET043+1) est un exemple de théorème trivial pour le Principe de Résolution, dont la démonstration par MUSCADET est plus complexe. Néanmoins, les méthodes utilisées par MUSCADET sont les mêmes que celles utilisées dans des théorèmes plus complexes où les démonstrateurs utilisant le Principe de Résolution échouent.

Pour le Principe de Résolution, la négation du théorème donne les deux clauses suivantes : $Y \in Y \vee Y \in f(X)$

$Y \notin Y \vee Y \notin f(X)$

qui donnent immédiatement la clause vide avec l'unification Y = f(X).

La démonstration de MUSCADET se trouve en Fig. 14.

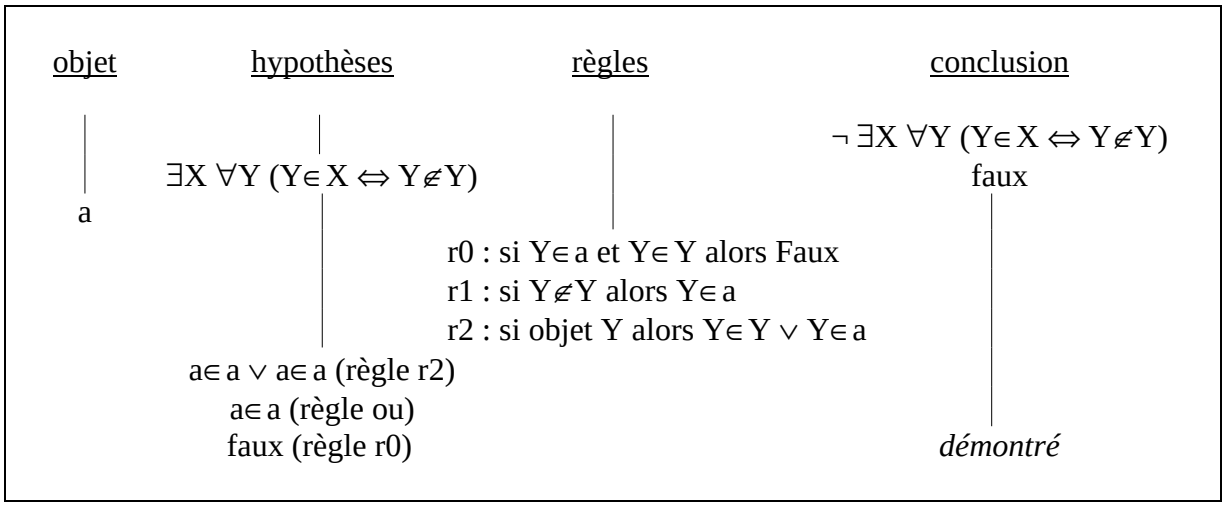

Fig. 14. Démonstration par PUSCADET du théorème SET043+1

Plusieurs centaines de problèmes classiques existaient dans le format CNF, mais, déchiffrer un ensemble de clauses CNF pour reconstruire l'énoncé du premier ordre est un travail encore plus long et fastidieux que construire les clauses à partir de l'énoncé du premier ordre. Je n'ai donc pas travaillé sur ces problèmes à partir de ces énoncés, mais sur des problèmes classiques similaires que j'ai (ou j'avais) déjà moi-même formulés (voir section 5.3).

5.2.2. Logique

Le domaine SYN comporte des énoncés "syntaxiques" n'ayant pas de sémantique évidente. Il s'agit de logique abstraite, d'ordre 0 ou 1. Des exemples d'énoncés et de démonstrations par PUSCADET sont donnés en annexes 1 et 2.

Certains énoncé sont triviaux. Il ne faut pas se fier au commentaire donné, par exemple, pour le premier exemple de l'annexe 1 (SYN040+1) où il est dit qu'il s'agit du théorème le plus difficile démontré par le Logic Theorist [1]. Le commentaire est seulement historique et valable pour des démonstrateurs uniquement syntaxiques qui ne connaîtraient pas la sémantique des connecteurs logiques $\Rightarrow$ et $\neg$, ce qui n'est pas le cas avec le Principe de Résolution ni la Déduction naturelle.

Certains théorèmes n'étaient pas démontrés par PUSCADET car ils contenaient des sous-formules que l'on ne rencontre jamais dans un énoncé traduisant un problème réel, et que MUSCADET ne générait jamais comme formule intermédiaire, par exemple $\neg\neg p$ (SYN041+1) ou $\neg\ (p \Rightarrow q)$ (SYN001+1). Il a été facile de rajouter des réécritures élémentaires.

D'autres théorèmes sont constitués de grosses formules, sans définition de prédicats intermédiaires et MUSCADET serait sans doute capable de les démontrer mais dans un temps déraisonnable que je ne lui ai jamais laissé. C'est le cas du théorème SYN007+1.14 qui est une instance, pour N=14 du théorème générique (voir section 4.1 et annexe 1)

p1 <=> (p2 <=>...(pN <=> (p1 <=> (p2 <=>...<=> pN)…)

[1] Et encore, Jacques Pitrat m'a signalé que le Logic Theorist n'avait montré l'implication que dans un sens.

et qui nécessite un nombre trop important de découpages. Le théorème SYN007+1.005, pour N=5, de taille moindre, est démontré en 90 secondes[1] (SPARC10 - 150 MHz - SWI-Prolog 2.1.3) et après 159 découpages.

Ces théorèmes sont des théorèmes artificiels pour lesquels les méthodes naturelles de MUSCADET ne sont pas du tout adaptées. En ce qui me concerne, pour démontrer ces théorèmes, ma première idée a été d'utiliser la traduction de la logique vers l'arithmétique (vrai / 1, faux / 0, $\wedge$ / *, $\neg$ / +1, $\neg \Leftrightarrow$ / +) et ils deviennent très faciles[2]. Cela confirme qu'il n'y a pas de méthode universelle pour démontrer tous les types de théorèmes. Il doit être possible de donner cette méthode au système ainsi que des caractéristiques des théorèmes pour lesquelles elle est conseillée.[3]

Certains théorèmes ne sont pas démontrés et les raisons n'en ont pas encore été étudiées.

Enfin, certains théorèmes correspondent à des traductions dans le calcul des prédicats du premier ordre de problèmes exprimés en logique K multi-modale et les énoncés sont énormes (jusqu'à 26 pages !). Ces énoncés ont été proposés par l'auteur de SPASS qui semble particulièrement adapté à ce type de problèmes (dits ALC problèmes). Ils existent aussi sous forme de clauses dans d'autres divisions que la division FOF. Dans les discussions animées qui ont précédé la compétition 1999 (voir section 6.2), certains ont suggéré que ces problèmes ne puissent pas être éligibles (voir section 6.1).

5.2.3. Management

Pour le domaine MGT, les difficultés ont d'abord été dues au fait qu'on n'énonce pas des problèmes d'économie comme des problèmes de mathématiques et que certaines formules ne pouvaient être traitées par PUSCADET. Dans certains cas, comme précédemment, il a suffi de rajouter des règles simples.

Deux cas ont été plus difficilement résolus.

Le premier concerne des axiomes ou des hypothèses de la forme $\forall X\ (p(X) \Rightarrow q(f(X)))$ (exemple MGT021+1) à partir desquels MUSCADET construit des règles de la forme

si <conditions à partir de p(X)> et si on a l'hypothèse Y:f(X) alors ajouter l'hypothèse q(Y)

Dans le contexte MGT, il faut aussi construire des règles de la forme

si <conditions à partir de p(X)> alors créer Y:f(X) et ajouter l'hypothèse q(Y)

qui crée tout de suite un objet *Y*, ou

si <conditions à partir de p(X)> alors ajouter l'hypothèse $\exists Y\ (Y{:}f(X) \wedge q(Y))$

---

[1] Les temps moyens sont plutôt de l'ordre de quelques seceondes

[2] p $\Leftrightarrow$ q étant traduit par 1 + p + q, la formule devient 1+p1+1+p2+…+1+pN+1+p1+1+p2+…+1+p(N-1) +pN qui est impair donc égale à 1, c'est-à-dire vrai.

[3] On peut aussi faire un raisonnement par récurrence et utilisant la sémantique. Supposons que le théorème soit vrai pour N-1 variables propositionnelles. Alors si p1 est vrai la formule est équivalente à (p2 <=>...(pN <=> (p2 <=>...<=> pN)…) qui est vraie par hypothèse de récurrence. Si p1 est faux, la formule est équivalente à ¬ (p2 <=>...(pN <=> ¬ (p2 <=>...<=> pN)…) qui est aussi équivalente à (p2 <=>...(pN <=> (p2 <=>...<=> pN)…). Un tel raisonnement est hors de portée des démonstrateurs actuels.

qui créera l'objet Y si et quand l'hypothèse existentielle sera traitée.

Ces règles peuvent être très gênantes dans d'autres contextes, surtout sous la première forme.

En mathématiques, s'il y a l'idée de création dans cet énoncé, il sera plutôt exprimé sous la forme $\forall X\,(p(X) \Rightarrow \exists Y\ q(Y)))$ et conduira à l'écriture d'une règle ajoutant des hypothèses existentielles qui créeront de nouveaux objets quand elles seront traitées. Le faire trop vite peut conduire à la création d'un chaîne infinie de nouveaux objets.

Des énoncés de la forme $p(X,...) \Rightarrow q(f(X,...))$ se rencontrent également, issus d'une définition $\forall X...(q(f(X,...)) \Leftrightarrow p(X,...))$ du symbole fonctionnel f (c'est, avec deux variables l'exemple de la définition de l'intersection vue en section 3.4.2.). Dans ce cas, il ne faut pas créer ce nouvel objet. MUSCADET dispose du mot-clef `definition` pour donner les énoncés qui sont des définitions. Pour la TPTP Library, il a fallu reconnaître automatiquement si un énoncé est une définition ou une propriété.

Voici quelques exemples d'axiomes, lemmes ou sous-formules issues de définitions, illustrant tous ces cas :

$<conditions\ sur\ a\ et\ b> \Rightarrow taux\_de\_croissance(a) < taux\_de\_croissance(b)$

(axiome) il faut introduire les taux de croissance

$A \subset B \Rightarrow \complement B \subset \complement A$

(lemme) ne considérer que les complémentaires qui existent déjà

$convexe(E) \wedge X \in E \wedge Y \in E \Rightarrow segment(X,Y) \subset E$

(issu de la définition de convexe) introduire les segments avec prudence

$X \in A \wedge Y \in B \Rightarrow X \in A \cap B$

(issu de la définition de l'intersection) ne considérer que les intersections qui existent déjà

$ouvert(U) \wedge \ldots \Rightarrow \exists U\,(V \subset U \wedge ouvert(V) \wedge \ldots)$

(propriété exprimée sous forme de savoir-faire) l'idée de création est explicite

Le deuxième cas difficile concerne des axiomes (lemmes) comportant des hypothèses universelles (exemple MGT031+1) de la forme

$$P(X) \wedge \forall Y\ Q(X,Y) \Rightarrow R(X)$$

Je n'en avais jamais rencontré dans les exemples que j'ai traités en mathématiques, sans doute parce que le mathématicien aurait alors défini un nouveau concept

$$S(X) \Leftrightarrow \forall Y\ Q(X,Y)$$

et l'axiome précédent serait exprimé par

$$P(X) \wedge S(X) \Rightarrow R(X)$$

Il a donc fallu rajouter des métarègles construisant, à partir du lemme

$$P(X) \wedge \forall Y\ Q(X,Y) \Rightarrow R(X)$$

des règles du type suivant

si *< conditions construites à partir de P >*
conclusion C
< conditions de ressemblance entre C et les hypothèses actuelles >
alors *nouvelle conclusion* $\forall Y\ Q(X,Y) \wedge (R(X) \Rightarrow C)$

5.2.4. Informatique théorique

La base TPTP comporte, en format FOF, trois modélisations du théorème affirmant l'indécidabilité de la terminaison des programmes (annexe 4). PUSCADET réussissait à en démontrer une fin 1998. La version actuelle de PUSCADET ne le démontre plus. Ce cas devra être analysé.

5.2.5. Résultats

Cette phase de tests sur la base TPTP a été provisoirement stoppée pour se consacrer aux suivantes (enrichissement de la base, mise au point d'un exécutable dans les conditions de la compétition).

Fin 1998, PUSCADET démontrait 4 théorèmes SET sur 5 (et le 5ème par une variante), 1 théorème COM sur 3. Il avait progressé, sans trop de difficulté dans le domaine SYN, plus laborieusement dans le domaine MGT.

Il réussissait à démontrer 8 des 40 théorèmes proposés dans la division FOF à la compétition 1998 (6 des 13 MGT, 2 des 25 SYN), ce qui était mieux que les performances d'Otter.

5.3. Enrichissement de la base TPTP

Il est prévu que la base soit enrichie par les utilisateurs, j'ai donc souhaité proposer tous les théorèmes de théorie des ensembles démontrés par MUSCADET (et même pour la plupart par DATTE). Pour certains, parmi les plus compliqués, j'étais persuadée que les démonstrateurs basés sur le Principe de Résolution, ne pourraient pas les démontrer (par exemple ceux du type de l'exemple de la section 3.5.2).

Outre la traduction dans la syntaxe TPTP, j'ai essayé d'utiliser des notations, des conventions et des définitions proches de celles utilisées pour les théorèmes en format CNF. Après déchiffrage laborieux des clauses de ces problèmes CNF, cela n'a été fait partiellement, car les choix faits par les précédents auteurs étaient assez axiomatiques alors que ma base personnelle reposait plutôt, au moins pour les notions élémentaires, sur la théorie naïve des ensembles et une pratique usuelle des mathématiques courantes.

Plusieurs difficultés plus sérieuses sont apparues.

La base de connaissances de MUSCADET, bien que générale, connaissait le symbole d'appartenance à un ensemble, utilisé dans plusieurs règles et métarègles, en particulier pour le traitement des quantificateurs relativisés ainsi que dans le traitement de plusieurs cas

particuliers de conclusions existentielles. Il a fallu réécrire ces règles sans cette facilité, car l'écriture des énoncé dans la base TPTP ne doit comporter aucun symbole non défini autre que logique.

Ensuite, la base de connaissances de MUSCADET utilisait quelques méta-symboles qui ne sont pas du premier ordre.

D'abord des prédicats `definition` et `lemme` permettaient de déclarer que des énoncés étaient soit des définitions, soit des lemmes, et ces énoncés n'étaient pas traités exactement de la même façon. En effet, on ne considère pas de la même façon la définition de l'inclusion

$$A \subset B \Leftrightarrow \forall X\ (X \in A \Rightarrow X \in B)$$

et le lemme exprimant la transitivité de l'inclusion

$$A \subset B \wedge B \subset C \Rightarrow A \subset C$$

En effet, une conclusion $a \subset b$ pourra être remplacée par sa définition $\forall X\ (X \in a \Rightarrow X \in b)$ mais pas par $a \subset B \wedge B \subset c$. Outre que ce n'est pas ainsi que l'on traite l'utilisation de conditions suffisantes, l'existence de la variable B, non instanciée, serait actuellement catastrophique.

Or dans la base TPTP, les énoncés sont donnés avec les mot-clefs `axiom`, `hypothesis` ou `conjecture` (pour le théorème à démontrer). Il m'a donc fallu donner mes définitions avec le mot-clef `axiom` et écrire des règles permettant de reconnaître si un "axiome" est une définition ou bien un lemme.

Un cas important où il est nécessaire de savoir qu'un énoncé est une définition est le suivant. MUSCADET est conçu pour travailler avec des propriétés *positives* plutôt que *négatives*. Pour cette raison, j'avais défini la propriété `non_disjoint` plutôt que `disjoint`

$$\texttt{non\_disjoint}(A,B) \Leftrightarrow \exists X\ (X \in A \wedge X \in B)$$

et $\texttt{disjoint}(A,B) \Leftrightarrow \neg\ \texttt{non\_disjoint}(A,B)$

PUSCADET reçoit maintenant la définition de `disjoint`,

$$\texttt{disjoint}(A,B) \Leftrightarrow \neg\ \exists X\ (X \in A \wedge X \in B)$$

ce qui est plus conforme au vocabulaire usuel (sinon à l'image mentale usuelle), mais pour chaque définition de la forme $p(\ldots) \Leftrightarrow \neg\ q(\ldots)$, il crée un nouveau concept de nom `non_p` et de définition $\texttt{non\_p}\ (\ldots) \Leftrightarrow q(\ldots)$

et remplace la définition de p par la définition

$$p(\ldots) \Leftrightarrow \neg\ \texttt{non\_p}\ (\ldots).$$

La définition de `non_disjoint` est ainsi générée automatiquement à partir de celle de `disjoint` et on est ainsi ramené à la situation antérieure que MUSCADET savait traiter.

Un autre symbole utilisé par MUSCADET est le symbole ":" : "`y:f(x,…)`" signifie que `y` est le nom de `f(x,…)`. Il est introduit par le système dans la mise à plat des formules (par exemple, à partir de la sous-formule `p(g(f(x))`, il crée les sous-formules `f_x:f(x)`,

`g_f_x:g(f_x)` et `p(g_f_x)` où `f_x` et `g_f_x` sont des constantes. Ce symbole apparaît aussi dans les règles de manipulation. Cette transformation ne posait pas de problème, mais ce symbole était aussi utilisé pour la donnée de quelques définitions dans le domaine des applications (composition et inverses), par exemple

```
comp(G,F,A,B,C) : H <=> qqs(X app A, qqs(Z app C,
            ..[H,X] : Z <=> ilexiste(Y app B, ..[F,X] :Y et ..[G,Y] : Z)))
```

Le symbole fonctionnel "`..`" ne posait pas de problème car il est du premier ordre et a été traduit par le prédicat `apply` déjà utilisé dans les énoncés CNF. (`apply(F,X,Y)` pour `..[F,X,]:Y`). Par contre, le symbole "`:`" doit être compris des démonstrateurs, il ne pouvait donc pas subsister. Je l'ai d'abord remplacé par "`=`" dans la définition et ai ajouté dans PUSCADET une règle qui le traduisait immédiatement par "`:`" dans ce type d'énoncés, ce qui ne changeait donc rien pour PUSCADET. Mais l'énoncé initial devenait contradictoire avec d'autres. Cette contradiction a été détectée par Geoff Sutcliffe testant la cohérence de ma base avec d'autres démonstrateurs (Otter, puis SPASS, plus performant). La raison profonde [1] est due à l'existence d'un énoncé définissant l'ensemble vide ϕ (ou à quelques autres énoncés moins simples). Considérant une application f dont le domaine est l'ensemble vide, que l'on veut composer avec g, on a

`comp(g,f,ϕ,B,C) = H <=> qqs(X app ϕ, … )`

le deuxième membre `qqs(X app ϕ, … )` est toujours vrai puisque ϕ n'a pas d'élément, le premier membre l'est donc également, ce qui signifie que n'importe quelle variable H est égale à `comp(g,f,ϕ,B,C)` donc que n'importe quoi est égal à n'importe quoi, ce qui conduit inévitablement à une contradiction.

Après quelques essais, le nouvel énoncé correct est alors le suivant :

```
qqs(X app A, qqs(Z app C, ..[comp(G,F,A,B,C),X]:Z <=>
                    ilexiste(Y app B, ..[F,X]:Y) et ..[G,Y]:Z))))
```

soit, dans la syntaxe TPTP

```
![G,F,A,B,C,X,Z]: ( (member(X,A) & member(Z,C))
                  => (apply(compose_function(G,F,A,B,C),X,Z)
                      <=> ?[Y]: ( member(Y,B) & apply(F,X,Y)
                                & apply(G,Y,Z)) ) )
```

L'inconvénient de ce nouvel énoncé est qu'il n'est pas reconnu comme définition, il est donc interprété comme lemme, ce qui n'est pas gênant pour lui, mais n'est pas très harmonieux ….

La version finale de cette base de nouveaux théorèmes TPTP a donc été mise au point à la suite de nombreux échanges avec Geoff Sutcliffe. Le plus crucial a été le problème des définitions de la composition et de l'inverse, mais il faisait suite à de nombreux échanges sur le vocabulaire utilisé (ainsi `complement` est devenu `difference`[2], …). D'autres modifications

[1] Cette raison simple n'a pas été trouvée tout de suite, car les réfutations transmises par Geoff Sutcliffe utilisait d'autres énoncés plus compliqués et aboutissaient à la clause vide après quelques détours …

[2] Les anglo-saxons appellent *différence de A et B* ce que nous appellons *complémentaire de B dans A.*

ont été faites directement par Geoff Sutcliffe pour harmoniser avec l'ensemble de la base (présentation des formules, `element` devenu `member`, …)[1], ou pour respecter les contraintes de la base TPTP, plus fortes que celles imposées à PUSCADET (le symbole `comp`, utilisé par PUSCADET à la fois comme prédicat ou comme symbole fonctionnel, qui est utilisé après que l'on ait démontré l'unicité de la composée, a été réécrit soit `compose_predicat` soit `compose_function`)[2]. Il a également rajouté, pour les démonstrateurs qui n'ont pas un traitement "built-in" de l'égalité, non seulement les axiomes standard de l'égalité, mais tous les axiomes de la forme $X = Y \wedge p(X,\ldots) \Rightarrow p(Y,\ldots)$ pour tous les prédicats p utilisés (et analogues pour les symboles fonctionnels). Je tiens à souligner ainsi que Geoff Sutcliffe ne se contente pas de rassembler des problèmes, mais construit la base de la manière la plus harmonieuse possible.

Nous avons eu également des discussions au sujet du symbole utilisé pour l'égalité d'ensembles (`=` ou `equal_set`) et de la définition du produit (annexe 5)[3] .

La nouvelle version 1999 contenait dans le domaine SET et en format FOF les 121 théorèmes que j'avais proposé, mais aussi 195 autres nouveaux théorèmes proposés par deux autres auteurs. Certains ont été démontrés sans problème par PUSCADET, d'autres ont été démontrés après l'ajout de règles, d'autres enfin ont conduits à des créations infinies de nouveaux éléments. Je n'ai pas encore eu le temps de les analyser.

5.4. Mise au point d'un exécutable respectant les contraintes de la compétition

Cela a été un travail beaucoup plus long que prévu, et certainement le plus stressant !

Chaque concurrent doit installer son système sur les machines du site de la compétition, une à deux semaines avant la compétition, puis n'y a plus accès, toutes les manipulations étant faites par l'organisateur. Le système doit être appelé par un nom d'exécutable suivi d'un nom de fichier contenant un problème. Or la pratique habituelle en Prolog est d'appeler Prolog et de lancer une commande Prolog. Cela n'était pas possible.

5.4.1. Premier essai

La première idée a été d'écrire un script shell faisant rentrer dans Prolog et indiquant les commandes à effectuer, selon la syntaxe suivante[4] :

---

[1] Ceci a d'ailleurs failli conduire à une catastrophe car , recevant la nouvelle version de la base TPTP, censée être celle de la compétition, je me suis aperçue, après des essais infructueux d'abord incompréhensibles, que la modification d'`element` en `member` n'avait pas été faite pour la dernière version de la définiton de la composition. Cela rendait indémontrable la plupart des théorèmes les plus difficiles … Heureusement, une dernière mise au point a été faite avant la compétition.

[2] Là encore, un oubli de cette nouvelle notation dans un théorème le rendait indémontrable.

[3] Doit-on imposer à l'ensemble dont on prend le produit de ne pas être vide, ce qui est indispendable en théorie axiomatique, mais ne semble pas contradictoire avec les énoncés modélisant la théorie naïve que j'ai donnés. En tout cas, SPASS n'a plus trouvé la définition contradictoire, après que la définition de `comp` ait été corrigée et je pense qu'elle ne l'est pas.

[4] Merci à Yannick Parchemal qui m'a indiqué cette syntaxe.

```
pl <<\! > /dev/null 2> /dev/null
['~pastre/puscadet/muscadet.pl'].
tptp('$1').
halt.
!
```

qui recevrait en argument (`$1`) le nom du fichier dans lequel se trouvent les données (`input_formula` de natures `axiom`, `hypothesis` et `conjecture`). `pl` est l'appel à SWI-Prolog, `muscadet.pl` est le fichier contenant toutes les données Prolog (démonstrateur et connaissances), `tptp` est le prédicat Prolog lançant la lecture d'un fichier et la démonstration.

La redirection (`2>`) de la sortie erreur standard vers le fichier Unix poubelle (`/dev/null`) évite que les `warning` ne soient affichés sur la sortie standard.

La difficulté est que le `$` est un caractère spécial en Prolog et que les paramètres d'un script shell doivent s'écrire `$1, $2, …` . Tous les essais de guillemets, quotes, anti-slashes ont été infructueux.

5.4.2. Première solution

La solution utilise trois scripts shell. Le premier script appelé modifie un deuxième script pour en générer un troisième qu'il exécute : `muscadet0` remplace dans `muscadet1` la chaîne `fichier` par le nom donné en argument `$1` dans `muscadet0` et le sauve en `muscadet2`, puis exécute `muscadet2`.

`muscadet0`
```
ed muscadet1 <<% > /dev/null
g/fichier/s//$1/
w muscadet2
q
%
muscadet2
```

`muscadet1`
```
pl <<\! > /dev/null 2> /dev/null
['~pastre/puscadet/muscadet.pl'].
tptp('fichier').
halt.
!
```

`muscadet2`
```
pl <<\! > /dev/null 2> /dev/null
['~pastre/puscadet/muscadet.pl'].
tptp('MGT025+1.p').
halt.
!
```

Ceci manquait d'élégance et surtout, l'inconvénient majeur était de devoir créer un fichier en cours d'exécution, les conditions de la compétition autoriseraient-elles la génération de ce script intermédiaire ?

5.4.3. Deuxième et meilleure solution

La deuxième idée a été d'écrire un programme C appelant Prolog en lui transmettant les paramètres. Divers essais infructueux, après consultation de la documentation SWI-Prolog[1], incompréhensible car ne donnant pratiquement pas d'exemples, m'ont forcée à abandonner.

[1] Merci à Bruno Bouzy pour l'aide dans ces essais.

Enfin, la solution a été d'écrire un petit programme C qui, au moyen des fonctions C `strcpy` et `strcat`[1], construit puis appelle le script sous forme d'une unique chaîne de caractères comportant le nom du fichier donné en entrée. `cmd` pointe vers la chaîne construite en plusieurs étapes, `tmp` vers l'argument, `\n` est le passage à la ligne. C'est tout à fait illisible, mais ça marche.

```
main(argc,argv)
int argc;
char *argv[];
{  char tmp[256];
   char tmp1[256];
   char cmd[256];
   strcpy(tmp, argv[1]);
   strcpy(cmd,"pl -f muscadet.pl <<! 2> /dev/null \n tptp('");
   strcat(cmd,tmp);
   strcpy(tmp1, "'). \n halt. \n ! ");
   strcat(cmd,tmp1);
   system(cmd);
}
```

L'option `-f` de Prolog charge le fichier `muscadet.pl` à l'appel de Prolog.

Mes efforts n'étaient pas encore terminés. Je voulais que le système n'écrive que le résultat demandé, et non la bannière Prolog, les messages de compilation, etc. C'est ce que fait l'option `-g true` [2]. Soit finalement, pour le début de la chaîne :

```
   strcpy(cmd,"pl -f muscadet.pl -g true <<! 2> /dev/null \n tptp('");
```

Il ne restait plus qu'à installer le système sur les machines du site de la compétition, rectifier quelques détails dus au changement de machine, et espérer que le système passe tous les tests … . En fait, seuls quelques cas de plantage à la lecture des données sont restés inexpliqués, sans doute dus au fait que la version de SWI-Prolog du site de la compétition n'était pas la même que la mienne.

6. La compétition 1999 (CASC-16)

Elle eut lieu à Trento (Italie) en juillet 1999. Il y avait 17 concurrents dont 6 dans la division FOF.

6.1. Choix des problèmes

Les problèmes proposés pour la compétition sont choisis parmi les problèmes les plus difficiles de la base. Leurs numéros et les noms de symboles sont remplacés par des noms sans signification, l'ordre des formules et sous-formules est aléatoirement modifié.

Les vainqueurs sont censés rendre leur source public.

---

[1] Merci encore Bruno Bouzy qui m'a indiqué cette possibilité.

[2] Merci à Miguel Pintado pour sa connaissance infinie des profondeurs de Prolog inconnues de tous les autres utilisateurs de Prolog consultés.

Le but des organisateurs, et ils font pour cela un gros travail, est de permettre une évaluation la plus objective possible des démonstrateurs (tout en rappelant régulièrement à ceux qui le prennent trop au sérieux que "it is for fun").

La difficulté de tous les problèmes de la base est évaluée en fonction des performances des participants (chacun a dû donner, quelques semaines avant, les performances de son système sur la totalité de la base, avec les temps d'exécution). Certaines considérations sont prises en compte, comme le fait que des problèmes se ressemblent ou que certains problèmes semblent faits pour certains démonstrateurs (par exemple si un système démontre un théorème qu'aucun autre ne démontre et ne démontre pas de nombreux autres théorèmes).

Un ensemble de problèmes *éligibles* est alors constitué, et le dernier choix se fait aléatoirement le matin de la compétition.

Un jury, composé de trois chercheurs, A.Bundy, C.Kirchner et F.Plenning, non impliqués dans la compétition, en contrôle le bon déroulement, avant, pendant et après.

6.2. Public ou secret ?

Un message de Geoff Sutcliffe, un mois avant la compétition, demandant à tous les inscrits leur avis sur une mise à jour des critères d'évaluation a provoqué, parmi les habitués de cette compétition, des échanges dont certains ont été assez violents, un vainqueur de 1998, ayant même été accusé de tricherie car ayant refusé de rendre public la totalité de son système. La vraie question est de savoir si un système a le droit de tourner pendant toute l'année sur la base de manière à faire des ajustements de certains paramètres, au moyen de tables dans le cas cité, et de garder ces tables secrètes. Le débat a ensuite porté sur la question de savoir si la compétition doit permettre de dégager le "meilleur", comme une course de Formule 1 (mais dans ce cas, est-il légitime d'utiliser des deniers publics ?) ou doit permettre de faire progresser la démonstration automatique (et dans ce cas, chacun doit pouvoir profiter du travail des autres). La plupart de auteurs ont d'ailleurs plusieurs versions de leur système, la version CASC étant une version particulière.

Le ton s'est calmé durant la conférence. En décrivant son système, le vainqueur précité a parlé d'apprentissage (learning) dans la mise au point de son système qu'un de ses opposants a qualifié d'ajustement (tuning).

Epilogue : immédiatement après la compétition, Geoff Sutcliffe a envoyé un message disant qu'il avait l'intention d'installer sur le Web les sources et les exécutables de tous les participants, et que si quelqu'un y voyait une objection qu'il le fasse savoir (pourquoi). Il n'y eut pas d'objection.

6.3. A propos des tests de validité

Quelques mois après la compétition, un des vainqueurs (qui ne concourait pas dans la division FOF) a été disqualifié car son système a été trouvé invalide au cours des tests post-compétition sur la totalité de la base. Il s'en est suivi un échange de courriers assez vifs, ses défenseurs argumentant qu'il n'avait pas été trouvé invalide sur les problèmes de la compétition, donc que cela n'avait pas influencé les résultats et qu'il n'avait pas à être disqualifié. Ils avancent également que même les logiciels professionnels ont des bogues et que personne ne peut garantir que son logiciel n'en a pas. Le jury a donné raison aux organisateurs et a confirmé la disqualification, considérant qu'il n'était pas sérieux de déclarer vainqueur un système non-valide ! D'abord, les problèmes de la compétition sont représentatifs d'un ensemble plus large de problèmes, sinon elle n'aurait pas d'intérêt pour la communauté ATP (Automated Theorem Proving). Ensuite, on ne peut pas avoir confiance sur la validité des démonstrations d'un système invalide, même sur les problèmes de la compétition[1]. Enfin, à titre de signal pour une communauté plus large, il est important de montrer que les organisateurs et le jury sont intransigeants, considèrent que le problème de la non-validité est sérieux et font tout ce qu'ils peuvent pour le réduire.

6.4. Les résultats de la CASC-16

Les résultats complets peuvent être consultés aux adresses

http://www.cs.jcu.edu.au/~tptp/CASC-16/WWWResults/Results.html

et http://www.cs.jcu.edu.au/~tptp/CASC-16/WWWResults/ResultsSummary.html

Les résultats pour la division FOF (FEQ et FNE) sont en Fig. 15, 16 et 17.

SPASS est de nouveau le meilleur globalement pour la division FOF. MUSCADET a été le meilleur dans la catégorie FEQ, mais le dernier (nul !) dans la catégorie FNE.

FOF Problems

	SPASS 1.00T	E-SETHEO 99csp	SPASS 0.95T	Bliksem 1.10	MUSCADET 2.1	OtterMACE 437
Attempted	30	30	30	30	30	30
Solved	22	22	19	12	9	9
Av. Time	25.46	32.30	28.36	1.58	1.48	7.54

Fig. 15. Résultats dans la division FOF

Dans la catégorie FEQ, MUSCADET a démontré 9 théorèmes sur 15 et en démontre maintenant 2 autres.

[1] En effet, le jour de la compétition, les systèmes sont évalués uniquement sur leur réponse : l'énoncé est, ou n'est pas, un théorème

FEQ Problems

	MUSCADET 2.1	SPASS 1.00T	E-SETHEO 99csp	Bliksem 1.10	SPASS 0.95T	OtterMACE 437
MGT021+1	0.70	0.10	9.40	0.00	0.00	0.30
SET055+1	unknown	timeout	3.10	timeout	timeout	0.70
SET065+1	unknown	0.30	3.00	0.10	0.10	0.70
SET071+1	1.50	3.40	40.80	timeout	timeout	timeout
SET084+1	unknown	0.40	114.30	0.70	9.40	65.40
SET094+1	unknown	1.10	18.90	0.20	1.10	timeout
SET352+4	1.00	0.40	timeout	timeout	timeout	timeout
SET697+4	no_soln	0.30	timeout	timeout	timeout	timeout
SET723+4	1.60	timeout	timeout	timeout	timeout	timeout
SET726+4	1.80	timeout	timeout	timeout	timeout	timeout
SET746+4	1.70	timeout	timeout	timeout	timeout	timeout
SET752+4	1.60	timeout	timeout	timeout	timeout	timeout
SET757+4	1.90	timeout	timeout	timeout	timeout	timeout
SET769+4	1.50	timeout	timeout	timeout	timeout	timeout
SYN072+1	no_soln	0.10	12.90	0.00	0.00	unknown
Attempted	**15**	**15**	**15**	**15**	**15**	**15**
Solved	**9**	**8**	**7**	**5**	**5**	**4**
Av. Time	**1.48**	**0.76**	**28.91**	**0.20**	**2.12**	**16.78**

Fig. 16. Résultats dans la catégorie FEQ de la division FOF

Dans la catégorie FNE, MUSCADET n'a démontré aucun des 15 théorèmes. Depuis,

- 3 autres théorèmes ont été démontrés après ajout de quelques transformations ou stratégies simples dans la base de connaissances de MUSCADET. (Elles n'y figuraient pas car correspondent à des façons de s'exprimer que l'on ne rencontre pas en mathématiques et je n'avais pas eu le temps d'étudier la base SYN dans sa totalité).
- un théorème est démontré mais uniquement par la variante consistant à traiter les hypothèses disjonctives avant les hypothèse existentielles.
- 3 théorèmes présentent des difficultés qui ne semblent pas insurmontables et devront être étudiés.
- enfin les 8 autres ont des énoncés de plus de 10 pages et n'ont pas été examinés en détail. MUSCADET n'est certainement pas adapté à ce genre de problèmes (ALC, voir section 5.2.2) !

MUSCADET démontre donc aujourd'hui 14 des 30 théorèmes (11+3). Les autres concurrents peuvent aussi avoir amélioré leur performance depuis.[1]

FNE Problems

	E-SETHEO 99csp	SPASS 0.95T	SPASS 1.00T	Bliksem 1.10	OtterMACE 437	MUSCADET 2.1
SYN036+1	8.80	0.00	0.30	18.00	timeout	unknown
SYN052+1	1.40	0.00	0.10	0.00	0.20	no_soln
SYN332+1	8.90	0.00	0.10	0.00	unknown	unknown
SYN362+1	1.40	0.00	0.10	0.00	0.20	unknown
SYN378+1	1.40	0.00	0.10	0.00	0.10	no_soln
SYN381+1	1.20	0.00	0.10	0.00	0.10	unknown
SYN411+1	1.50	0.00	0.10	0.00	0.20	unknown
SYN436+1	19.70	91.20	90.20	timeout	timeout	timeout
SYN439+1	136.20	timeout	timeout	timeout	timeout	timeout
SYN448+1	27.60	37.40	37.20	timeout	timeout	timeout
SYN466+1	121.40	99.10	131.20	timeout	timeout	timeout
SYN469+1	35.40	69.90	70.60	timeout	timeout	timeout
SYN484+1	56.50	65.20	64.80	timeout	timeout	timeout
SYN487+1	30.20	114.10	106.60	timeout	timeout	timeout
SYN508+1	56.60	51.40	52.50	timeout	timeout	timeout
Attempted	**15**	**15**	**15**	**15**	**15**	**15**
Solved	**15**	**14**	**14**	**7**	**5**	**0**
Av. Time	**33.88**	**37.74**	**39.57**	**2.57**	**0.16**	**-**

Fig. 17. Résultats dans la catégorie FNE de la division FOF

6.5. Les problèmes de la division FOF de la CASC-16

6.5.1. catégorie FEQ

SET 13 problèmes, dont :

- 8 proposés par moi-même, MUSCADET a échoué pour 1 (pour une raison inconnue, il réussit à Paris 5), SPASS en a démontré 2, les autres démonstrateurs aucun

- 2 sur les opérations ensemblistes élémentaires :

$\sigma(\{A,B\}) = A \cup B$ (SET352+4)

[1] Citons par exemple une anecdote. Comme les organisateurs s'étonnaient qu'un démonstrateur, dans une autre division, ayant de très bons résultats les années précédentes, n'avait que des résultats médiocres, son auteur a indiqué qu'il s'était trompé de version en installant son système …

$A \subset B$ <=> $A \cap \complement(B,E) = \phi$ (SET697+4, non démontré par MUSCADET le jour de la compétition)

- 3 sur les applications :

si f,g,h sont des applications de A dans A, et si f est injective, alors $f_o g = f_o h$ => g=h (SET723+4)

si f est une application de A dans B, g et h deux applications de B dans A telle que $g_o f = id_A$ et $f_o h = id_B$, alors $g = f^{-1}$ (SET726+4)

soient f et g deux applications resp. de A dans B et de B dans C, si f et g sont croissantes alors $g_o f$ est croissante (SET746+4)

- 2 sur les ensembles images/images réciproques :

$f(X \cup Y) = f(X) \cup f(Y)$ (SET752+4)

$f^{-1}(X \cap Y) = f^{-1}(X) \cap f^{-1}(Y)$ (SET757+4)

- 1 sur les relations d'équivalence :

si on a une relation d'équivalence sur E, alors deux classes d'équivalence sont égales si et seulement si elles ne sont pas disjointes (SET769+4)

- 5 issus de Quaife (1992) (SET...+1.p). PUSCADET en a démontré un.

Ces théorèmes sont basés sur une axiomatique qui devra être examinée en détails.

MGT 1 problème, démontré par tous les démonstrateurs (MGT021+1, voir annexe 3).

SYN 1 problème, non démontré le jour de la compétition, démontré depuis.

6.5.2. catégorie FNE

MUSCADET n'a démontré aucun des théorèmes de cette catégorie qui étaient tous des problèmes du domaine SYN.

8 théorèmes (ALC, voir section 5.2.2) ont des énoncés de plus de 10 pages … . Seuls E et SPASS ont réussi.

Les 7 autres sont les suivants :

SYN411+1 $\forall X \forall Y \forall Z\ p(X,Y,Z) \Leftrightarrow \neg\ \exists U \exists V \exists W \neg\ p(U,V,W)$

théorème trivial de logique pure que tous les démonstrateurs ont démontré sauf MUSCADET car il ne savait pas réécrire une hypothèse $\neg \exists U\ F$ en $\forall U \neg\ F$. Il savait traiter $\neg \exists U\ F$ en tant que conclusion et savait construire des règles à partir d'hypothèses universelles et pouvait ainsi démontrer l'implication dans le premier sens (Fig. 18).

Pour que MUSCADET démontre l'implication dans l'autre sens, d'une manière analogue, il a suffi d'ajouter la prise en compte des hypothèses de la forme $\neg\ \exists U\ F$ et de la forme $\neg\neg F$ pour la construction des règles à partir des hypothèses.

Les réécritures $\neg\ \exists U\ F$ en $\forall U \neg\ F$ et $\neg\neg F$ en F étant de celles effectuées pour mettre sous forme de clause, ce théorème est trivial pour le Principe de Résolution. On pourrait envisager de donner aussi directement ces réécritures pour démontrer immédiatement ce genre de théorèmes. Les méthodes employées sont plus générales.

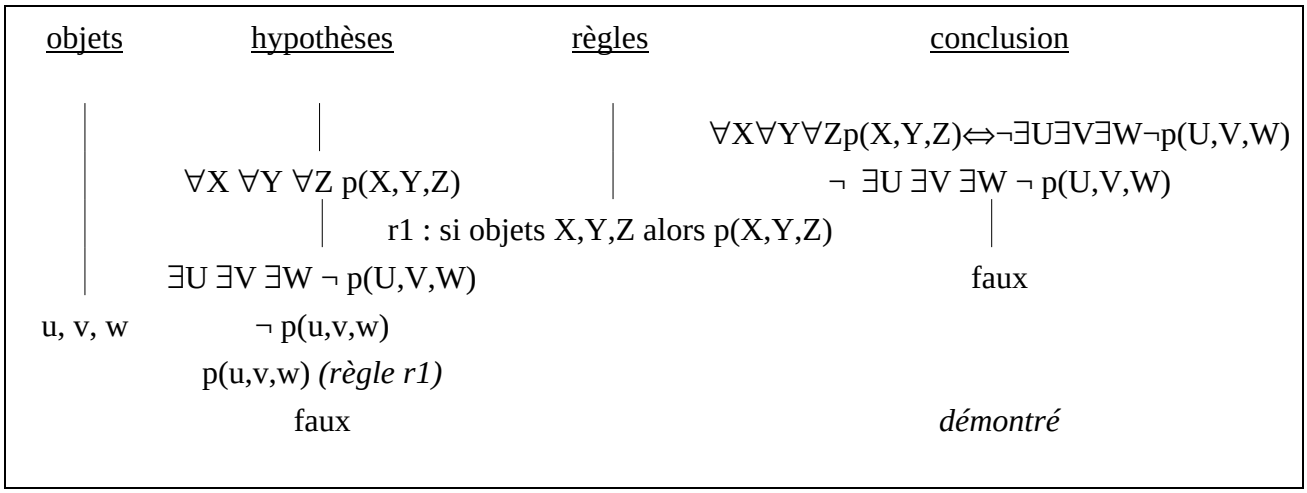

Fig. 18. Démonstration du premier sous-théorème du théorème SYN411+1

SYN036+1 $[\exists X\ \forall Y\ (p(X) \Leftrightarrow p(Y)) \Leftrightarrow (\exists U\ q(U) \Leftrightarrow \forall W\ q(W))] \Leftrightarrow [\exists X1\ \forall Y1\ (q(X1) \Leftrightarrow q(Y1)) \Leftrightarrow (\exists U1\ p(U1) \Leftrightarrow \forall W1\ p(W1))]$

Cet énoncé est assez artificiel. Sa sémantique n'est pas évidente et une fois comprise, il est artificiellement compliqué. En effet les énoncés $\exists X\ \forall Y(p(X) \Leftrightarrow p(Y))$ et $\exists U1\ p(U1) \Leftrightarrow \forall W1\ p(W1)$ ont la même interprétation : "le prédicat *p* prend toujours la même valeur". Une fois cette remarque faite, il suffit, en désignant par $a(p)$ la propriété $\exists X\ \forall Y\ (p(X) \Leftrightarrow p(Y))$ et par $b(p)$ la propriété $\exists U1\ p(U1) \Leftrightarrow \forall W1\ p(W1)$ de démontrer que $\forall P(a(P) \Leftrightarrow b(P)) \Rightarrow \forall P\ \forall Q([a(P) \Leftrightarrow b(Q)] \Leftrightarrow [a(Q) \Leftrightarrow a(P)]$ (qui est un énoncé du deuxième ordre, si l'on se souvient que P et Q sont des propriétés).

PUSCADET n'est cependant pas capable de démontrer que

$$\exists X\ \forall Y\ [p(X) \Leftrightarrow p(Y)] \Leftrightarrow [\exists U1\ p(U1) \Leftrightarrow \forall W1\ p(W1)]$$

mais démontre le théorème (facile)

$$\forall P\ [a(P) \Leftrightarrow b(P)] \Rightarrow \forall P\ \forall Q\ ([a(P) \Leftrightarrow b(Q)] \Leftrightarrow [a(Q) \Leftrightarrow a(P)])$$

SYN378+1 $\forall X\ p(X) \Rightarrow (\neg\ \exists Y\ q(Y) \vee \exists Z\ [p(Z) \Rightarrow q(Z)])$

PUSCADET ajoute l'hypothèse $\exists Y\ q(Y)$ et cherche à démontrer $\exists Z\ [p(Z) \Rightarrow q(Z)]$. Il crée un objet *y* tel que $q(y)$. Le jour de la compétition, il ne pouvait plus rien faire. Il démontre maintenant ce théorème après que le traitement des conclusions existentielles simples ait été complété.

SYN362+1 $[\forall Y\ \exists W\ r(Y,W) \wedge \exists Z\ \forall X\ (p(X) \Rightarrow \neg\ r(Z,X))] \Rightarrow \exists X\ \neg\ p(X)$

est également maintenant démontré par MUSCADET (traitement de la conclusion existentielle par l'absurde : on ajoute l'hypothèse universelle$\forall X\ p(X)$

SYN052+1 $\forall X\ [p \Leftrightarrow f(X)] \Rightarrow [p \Leftrightarrow \forall X1\ f(X1)]$

PUSCADET sait démontrer dans un sens $p \Rightarrow \forall X1\ f(X1)$ mais pas dans l'autre $\forall X1\ f(X1) \Rightarrow p$ car aucun objet n'est créé.

SYN381+1 $\forall X\ [\exists Y\ q(X,Y) \Rightarrow p(X)]$
$\wedge\ \forall V\ \exists U\ q(U,V)$
$\wedge\ \forall W\ \forall Z\ [q(W,Z) \Rightarrow (Z,W) \vee q(Z,Z)]$
$\Rightarrow \forall Z\ p(Z)$

Ce théorème est démontré avec la variante consistant à traiter les hypothèses disjonctives avant les hypothèses existentielles (de même que le théorème SET044+1 cité en 5.2.1). Sinon, une chaîne infinie de nouveaux objets est créée.

SYN332+1 $\exists X\ \exists Y\ \forall Z\ \big(\ [f(X,Y) \wedge f(Y,X) \Leftrightarrow f(X,Z)]$
$\Rightarrow \big[[f(X,Z) \Leftrightarrow f(Z,X)]$
$\Rightarrow \big(\ [f(X,Z) \Leftrightarrow f(Y,Z)]$
$\Rightarrow \big[\big([f(Y,X) \Rightarrow f(X,Y)] \Leftrightarrow f(Z,Z)\big)$
$\Rightarrow \big(\ [f(X,Y) \Leftrightarrow f(Y,X)] \Leftrightarrow f(Z,Y)\big)\big]\big]\big)$

PUSCADET n'est pas actuellement capable de démontrer ce théorème. J'ai moi-même mis assez longtemps pour le démontrer à la main. On trouvera en annexe 6 plusieurs démonstrations et le monitoring de leur recherche. On remarquera l'utilité de combiner plusieurs approches.

7. Conclusion

Les nombreuses améliorations du système MUSCADET, incluant sa réécriture complète en Prolog l'ont rendu plus performant. Il a ainsi pu démontrer des théorèmes énoncés dans un style différent de ceux sur lesquels les expérimentations avaient été auparavant faites, dans les domaines de l' économie, la logique pure, et les paradoxes de la théorie des ensembles.

MUSCADET a obtenu à la compétition 1999 organisée dans le cadre de la conférence CADE des résultats qui ont montré la supériorité des méthodes utilisées sur le Principe de résolution pour certains types de problèmes. En effet, il a eu les meilleurs résultats dans la catégorie "Premier ordre avec égalité". Il a au contraire eu de très mauvais résultats (bien qu'améliorés depuis) dans la catégorie "Premier ordre sans égalité".

La différence des résultats ne provient pas uniquement de la présence ou non de l'égalité mais il se trouve que les théorèmes comportant les difficultés pour lesquelles MUSCADET est efficace comportent aussi l'égalité. Ce sont des théorèmes manipulant de nombreux concepts définis par d'autres énoncés à partir desquels MUSCADET construit des règles, par opposition à des énoncés uniques pouvant être énormes et/ou dénués de sémantique. Ce sont aussi des théorèmes traitant de propriétés susceptibles de créer de nouveaux objets ou éléments. Le traitement des propriétés existentielles incorporé dans MUSCADET lui permet de procéder à des créations dans toutes les directions nécessaires, sans toutefois s'enfermer dans des chaînes infinies d'objets créés.

Ces résultats montrent également, si cela était encore nécessaire, l'intérêt de combiner plusieurs types de méthodes. Il est en effet possible d'analyser les familles d'énoncés pour lesquels un ou un autre type de méthode sera le plus efficace.

REFERENCES

Bazin, J.M. (1993). *GEOMUS: un résolveur de problèmes de géométrie qui mobilise ses connaissances en fonction du problème posé*, thèse université Paris 6

Bledsoe, W.W. (1971). *Splitting and reduction heuristics in automatic theorem proving*, Journal of Artificial Intelligence 2, 55-77

Bledsoe, W.W. (1977). *Non-resolution theorem proving*, Journal of Artificial Intelligence 9, 1-35

Bledsoe, W.W., Tyson, M. (1978). *The UT interactive prover*, University of Texas, math. dept Memo ATP 17A

Lenat, D.B. (1982), *AM : discovery in mathematics as heuristic search*, in Davis, R., Lenat, D.B, Knowledge-based systems in artificial intelligence, Mc Graw Hill

Pastre, D. (1978). *Automatic Theorem Proving in Set Theory*, Journal of Artificial Intelligence 10, 1-27

Pastre, D. (1989). *MUSCADET: an automatic theorem proving system using knowledge and metaknowledge in mathematics*, Journal of Artificial Intelligence 38, 257-318

Pastre, D. (1993). *Automated Theorem Proving in Mathematics*, Annals of Mathematics and Artificial Intelligence, Volume 8, No. 3-4, 425-447

Pitrat, J. (1990), *Métaconnaissances, futur de l'intelligence artificielle*, Hermès

Robinson, J.A. (1965). *A machine oriented logic based on the resolution principle*, J.ACM 12, 23-41

Spagnol, J.P. (1999). *Automatisation du raisonnement et de la rédaction des preuves en géométrie des configurations*, dans cette publication

Wang H, (1960) *Towards mechanical mathematics*, IBM J. Res. Dev. 4, 2-22

Wos, L. (1988). *Automated Reasoning : 33 Basic Research Problems*, Prentice Hall

```
%------------------------------------------------------------------------------
% File     : SYN040+1 : TPTP v2.2.0. Released v2.0.0.
% Domain   : Syntactic
% Problem  : Pelletier Problem 1
% Version  : Especial.
% English  : A biconditional version of the 'most difficult' theorem
%            proved by the original Logic Theorist.
% Refs     : [NSS63] Newell et al. (1963), Empirical Explorations with the
%          : [Pel86] Pelletier (1986), Seventy-five Problems for Testing Au
%          : [Hah94] Haehnle (1994), Email to G. Sutcliffe
% Source   : [Hah94]
% Names    : Pelletier 1 [Pel86]
% Status   : theorem
% Rating   : 0.00 v2.1.0
% Syntax   : Number of formulae    :    1 (   0 unit)
%            Number of atoms       :    4 (   0 equality)
%            Maximal formula depth :    4 (   4 average)
%            Number of connectives :    5 (   2 ~  ;   0  |;   0  &)
%                                         (   1 <=>;   2 =>;   0 <=)
%                                         (   0 <~>;   0 ~|;   0 ~&)
%            Number of predicates  :    2 (   2 propositional; 0-0 arity)
%            Number of functors    :    0 (   0 constant; --- arity)
%            Number of variables   :    0 (   0 singleton;   0 !;   0 ?)
%            Maximal term depth    :    0 (   0 average)
% Comments : [NSS63] first appeared in 1957, as cited in [Pel86]. The 1963
%            version is a reprint.
%------------------------------------------------------------------------------
input_formula(pel1,conjecture,(
    ( p
   => q )
<=> ( ~ q
   => ~ p )   )).
%------------------------------------------------------------------------------


%------------------------------------------------------------------------------
% File     : SYN001+1 : TPTP v2.2.0. Released v2.0.0.
...
%------------------------------------------------------------------------------
input_formula(pel2,conjecture,(
    ~ ~ p
<=> p   )).
%------------------------------------------------------------------------------


%------------------------------------------------------------------------------
% File     : SYN007+1.005 : TPTP v2.2.0. Released v2.0.0.
% Domain   : Syntactic
% Problem  : Pelletier Problem 71
% Version  : Especial.
%            Theorem formulation : For N = SIZE.
% English  : Clausal forms of statements of the form :
%            (p1 <-> (p2 <->...(pN <-> (p1 <-> (p2 <->...<-> pN)...)
% Refs     : [Pel86] Pelletier (1986), Seventy-five Problems for Testing Au
%          : [Urq87] Urquart (1987), Hard Problems for Resolution
% Source   : [Pel86]
% Names    : Pelletier 71 [Pel86]
% Status   : theorem
% Rating   : ? v2.0.0
% Syntax   : Number of formulae    :    1 (   0 unit)
%            Number of atoms       :   10 (   0 equality)
%            Maximal formula depth :   10 (  10 average)
%            Number of connectives :    9 (   0 ~  ;   0  |;   0  &)
%                                         (   9 <=>;   0 =>;   0 <=)
%                                         (   0 <~>;   0 ~|;   0 ~&)
```

```
%            Number of predicates  :    5 (   5 propositional; 0-0 arity)
%            Number of functors    :    0 (   0 constant; --- arity)
%            Number of variables   :    0 (   0 singleton;   0 !;   0 ?)
%            Maximal term depth    :    0 (   0 average)
% Comments : The number of distinct letters in U-N is N. The number of
%            occurrences of sentence letters in 2N. The number of clauses
%            goes up dramatically as N increases, but I don't think it
%            shows that the problems are dramatically more difficult as N
%            increases. Rather, it's that the awkward clause form
%            representation comes to the fore most dramatically with
%            embedded biconditionals. On all other measures of complexity,
%            one should say that the problems increase linearly in
%            difficulty. Urquhart says that the proof size of any resolution
%            system increases exponentially with increase in N.
%          : This problem can also be done in terms of graphs, as described
%            in [Pel86] Problem 74.
%          : tptp2X: -ftptp -s5 SYN007+1.g
%--------------------------------------------------------------------------
input_formula(prove_this,conjecture,(
    p_1
<=> ( p_2
  <=> ( p_3
    <=> ( p_4
      <=> ( p_5
        <=> ( p_1
          <=> ( p_2
            <=> ( p_3
              <=> ( p_4
                <=> p_5 ) ) ) ) ) ) ) )   )).
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
% File     : SYN007+1.014 : TPTP v2.2.0. Released v2.0.0.
...
%--------------------------------------------------------------------------
input_formula(prove_this,conjecture,(
    p_1
<=> ( p_2
  <=> ( p_3
    <=> ( p_4
      <=> ( p_5
        <=> ( p_6
          <=> ( p_7
            <=> ( p_8
              <=> ( p_9
                <=> ( p_10
                  <=> ( p_11
                    <=> ( p_12
                      <=> ( p_13
                        <=> ( p_14
                          <=> ( p_1
                            <=> ( p_2
                              <=> ( p_3
                                <=> ( p_4
                                  <=> ( p_5
                                    <=> ( p_6
                                      <=> ( p_7
                                        <=> ( p_8
                                          <=> ( p_9
                                            <=> ( p_10
                                              <=> ( p_11
                                                <=> ( p_12
                                                  <=> ( p_13
                                                    <=> p_14 ) ) ) ) ) ) ) ) ) )
) ) ) ) ) ) ) ) ) ) ) ) ) ) ) )   )).
%--------------------------------------------------------------------------
```

```
%--------------------------------------------------------------------------
% File     : SYN041+1 : TPTP v2.2.0. Released v2.0.0.
...
%--------------------------------------------------------------------------
input_formula(pel3,conjecture,(
    ~ ( p
     => q )
 => ( q
   => p )   )).
%--------------------------------------------------------------------------



%--------------------------------------------------------------------------
% File     : SYN062+1 : TPTP v2.2.0. Released v2.0.0.
...
%--------------------------------------------------------------------------
input_formula(pel32_1,axiom,(
    ! [X] :
      ( ( big_f(X)
        & ( big_g(X)
          | big_h(X) ) )
     => big_i(X) )   )).

input_formula(pel32_2,axiom,(
    ! [X] :
      ( ( big_i(X)
        & big_h(X) )
     => big_j(X) )   )).

input_formula(pel32_3,axiom,(
    ! [X] :
      ( big_k(X)
     => big_h(X) )   )).

input_formula(pel32,conjecture,(
    ! [X] :
      ( ( big_f(X)
        & big_k(X) )
     => big_j(X) )   )).
%--------------------------------------------------------------------------



%--------------------------------------------------------------------------
% File     : SYN056+1 : TPTP v2.2.0. Released v2.0.0.
...
%--------------------------------------------------------------------------
input_formula(pel26_1,axiom,(
    ? [X] : big_p(X)
<=> ? [Y] : big_q(Y)   )).

input_formula(pel26_2,axiom,(
    ! [X,Y] :
      ( ( big_p(X)
        & big_q(Y) )
     => ( big_r(X)
      <=> big_s(Y) ) )   )).

input_formula(pel26,conjecture,(
    ! [X] :
      ( big_p(X)
     => big_r(X) )
<=> ! [Y] :
      ( big_q(Y)
     => big_s(Y) )   )).
%--------------------------------------------------------------------------
```

res_SYN040+1.p

```
pel1
**************************************************
theoreme a demontrer
(p => q) <=> (non q => non p)
0 nouvelle conclusion (p => q) <=> (non q => non p)

regles actives:elifon stop1 stop2 stop3 stop4 egal egaldef ou1 ou23 ou4 ou5 non
hypnon hypnonnon hypnonimp hypimp non_ou_1 non_ou_2 concl_ou_et => <=> qqs1 qqs2
seul .. .. 1 ilec1 ilec2 ilec1a ilec1b et_trivial_1 et_trivial_2 stop_trivial
stop_trivial_ou conclet bidon_ reflexivity_ et defconcl1 defconcl1a defconcl1b
defconcl1bb ilexiste ou defconcl2 defconcl2app defconcl2elt defconcl2a
defconcl2b defconcl3

0 nouvelle conclusion ((p => q) => (non q => non p)) et ((non q => non p) => (p
=> q))
---------------------------------------------------------- regle <=>

********************************************** sous - theoreme 1 *****
1 nouvelle conclusion (p => q) => (non q => non p)
---------------------------------------------- creation du sous - theoreme 1
1 ajouter la regle active locale

- (regle(A, reghyp) :- hyp(A, p), not hyp(A, q), ajhyp(A, q)).

a la fin des regles universelles

1 nouvelle conclusion non q => non p
---------------------------------------------------------- regle =>
1 ajouter hypothese non q
1 nouvelle conclusion non p
---------------------------------------------------------- regle =>
1 ajouter hypothese p
1 nouvelle conclusion faux
---------------------------------------------------------- regle non
supprimer hypothese non q
1 nouvelle conclusion q
---------------------------------------------------------- regle hypnon
1 ajouter hypothese q
---------------------------------------------------------- regle reghyp
1 nouvelle conclusion vrai
---------------------------------------------------------- regle stop1
theoreme 1 demontre
====================
0 nouvelle conclusion (non q => non p) => (p => q)

********************************************** sous - theoreme 2 *****
2 nouvelle conclusion (non q => non p) => (p => q)
---------------------------------------------- creation du sous - theoreme 2
2 ajouter la regle active locale

- (regle(A, reghyp1) :- hyp(A, non q), hyp(A, p), not hyp(A, faux), ajhyp(A,
faux)).

a la fin des regles universelles

2 ajouter la regle active locale

- (regle(A, reghyp2) :- not hyp(A, q ou non p), ajhyp(A, q ou non p)).

a la fin des regles universelles

2 nouvelle conclusion p => q
---------------------------------------------------------- regle =>
```

```
2 ajouter hypothese p
2 nouvelle conclusion q
---------------------------------------------------------- regle =>
2 ajouter hypothese q ou non p
---------------------------------------------------------- regle reghyp2
2 traitement de l hypothese disjonctive q ou non p
2 nouvelle conclusion (q => q) et (non p => q)
2 ajouter hypothese-traitee q ou non p
---------------------------------------------------------- regle ou

********************************************** sous - theoreme 21 *****
21 nouvelle conclusion q => q
---------------------------------------------- creation du sous - theoreme 21
21 ajouter hypothese q
21 nouvelle conclusion q
---------------------------------------------------------- regle =>
21 nouvelle conclusion vrai
---------------------------------------------------------- regle stop1
theoreme 21 demontre
====================
2 nouvelle conclusion non p => q

********************************************** sous - theoreme 22 *****
22 nouvelle conclusion non p => q
---------------------------------------------- creation du sous - theoreme 22
22 ajouter hypothese non p
22 nouvelle conclusion q
---------------------------------------------------------- regle =>
22 nouvelle conclusion vrai
---------------------------------------------------------- regle stop4
theoreme 22 demontre
====================
2 nouvelle conclusion vrai
---------------------------------------------------------- regle et
theoreme 2 demontre
====================
0 nouvelle conclusion vrai
---------------------------------------------------------- regle et
theoreme 0 demontre
====================
```

---

res_SYN001+1.p

```
pel2
**************************************************
theoreme a demontrer
non non p <=> p
0 nouvelle conclusion non non p <=> p

regles actives:elifon stop1 stop2 stop3 stop4 egal egaldef ou1 ou23 ou4 ou5 non
hypnon hypnonnon hypnonimp hypimp non_ou_1 non_ou_2 concl_ou_et => <=> qqs1 qqs2
seul .. .. 1 ilec1 ilec2 ilec1a ilec1b et_trivial_1 et_trivial_2 stop_trivial
stop_trivial_ou conclet bidon_ reflexivity_ et defconcl1 defconcl1a defconcl1b
defconcl1bb ilexiste ou defconcl2 defconcl2app defconcl2elt defconcl2a
defconcl2b defconcl3

0 nouvelle conclusion (non non p => p) et (p => non non p)
---------------------------------------------------------- regle <=>

********************************************** sous - theoreme 1 *****
1 nouvelle conclusion non non p => p
---------------------------------------------- creation du sous - theoreme 1
1 ajouter hypothese non non p
1 nouvelle conclusion p
---------------------------------------------------------- regle =>
1 ajouter hypothese p
---------------------------------------------------------- regle hypnonnon
1 nouvelle conclusion vrai
```

```
---------------------------------------------------------- regle stop1
theoreme 1 demontre
====================
0 nouvelle conclusion p => non non p

********************************************** sous - theoreme 2 *****
2 nouvelle conclusion p => non non p
---------------------------------------------- creation du sous - theoreme 2
2 ajouter hypothese p
2 nouvelle conclusion non non p
---------------------------------------------------------- regle =>
2 ajouter hypothese non p
2 nouvelle conclusion faux
---------------------------------------------------------- regle non
2 nouvelle conclusion vrai
---------------------------------------------------------- regle stop4
theoreme 2 demontre
====================
0 nouvelle conclusion vrai
---------------------------------------------------------- regle et
theoreme 0 demontre
====================
```

---

res_SYN041+1.p

```
pel3
**************************************************
theoreme a demontrer
non (p => q) => (q => p)
0 nouvelle conclusion non (p => q) => (q => p)

regles actives:elifon stop1 stop2 stop3 stop4 egal egaldef ou1 ou23 ou4 ou5 non
hypnon hypnonnon hypnonimp hypimp non_ou_1 non_ou_2 concl_ou_et => <=> qqs1 qqs2
seul .. .. 1 ilec1 ilec2 ilec3 ilec4 ilec5 ilec1a ilec1b et_trivial_1
et_trivial_2 stop_trivial stop_trivial_ou conclet bidon_ et defconcl1 defconcl1a
defconcl1b defconcl1bb ilexiste ou defconcl2 defconcl2app defconcl2elt
defconcl2a defconcl2b defconcl3

0 ajouter hypothese non (p => q)
0 nouvelle conclusion q => p
---------------------------------------------------------- regle =>
0 ajouter hypothese p
0 ajouter hypothese non q
---------------------------------------------------------- regle hypnonimp
0 ajouter hypothese q
0 nouvelle conclusion p
---------------------------------------------------------- regle =>
0 nouvelle conclusion vrai
---------------------------------------------------------- regle stop1
theoreme 0 demontre ( pel3 ) en 1.233333 seconds
====================
```

---

res_SYN062+1.p

```
pel32
**************************************************
theoreme a demontrer

- qqs(A, big_f(A) et big_k(A) => big_j(A)).

0 nouvelle conclusion
- qqs(A, big_f(A) et big_k(A) => big_j(A)).

regles actives:elifon stop1 stop2 stop3 stop4 egal egaldef ou1 ou23 ou4 ou5 non
hypnon hypnonnon hypnonimp hypimp non_ou_1 non_ou_2 concl_ou_et => <=> qqs1 qqs2
seul .. .. 1 ilec1 ilec2 ilec1a ilec1b et_trivial_1 et_trivial_2 stop_trivial
stop_trivial_ou conclet bidon_ pel32_1_ pel32_1_1 pel32_2_ pel32_3_ reflexivity_
```

```
et defconcl1 defconcl1a defconcl1b defconcl1bb ilexiste ou defconcl2
defconcl2app defconcl2elt defconcl2a defconcl2b defconcl3

0 ajouter objet x
0 nouvelle conclusion big_f(x) et big_k(x) => big_j(x)
---------------------------------------------------------- regle qqs1
0 ajouter hypothese big_f(x)
0 ajouter hypothese big_k(x)
0 nouvelle conclusion big_j(x)
---------------------------------------------------------- regle =>
0 ajouter hypothese big_h(x)
---------------------------------------------------------- regle pel32_3_
0 ajouter hypothese big_i(x)
---------------------------------------------------------- regle pel32_1_1
0 ajouter hypothese big_j(x)
---------------------------------------------------------- regle pel32_2_
0 nouvelle conclusion vrai
---------------------------------------------------------- regle stop1
theoreme 0 demontre
====================
```

---

res_SYN056+1.p

```
pel26
**************************************************
theoreme a demontrer

- (qqs(A, big_p(A) => big_r(A)) <=> qqs(B, big_q(B) => big_s(B))).

0 nouvelle conclusion
- (qqs(A, big_p(A) => big_r(A)) <=> qqs(B, big_q(B) => big_s(B))).

regles actives:elifon stop1 stop2 stop3 stop4 egal egaldef ou1 ou23 ou4 ou5 non
hypnon hypnonnon hypnonimp hypimp non_ou_1 non_ou_2 concl_ou_et => <=> qqs1 qqs2
seul .. .. 1 ilec1 ilec2 ilec1a ilec1b et_trivial_1 et_trivial_2 stop_trivial
stop_trivial_ou conclet bidon_ pel26_1_ pel26_1_1 pel26_2_ pel26_2_1
reflexivity_ et defconcl1 defconcl1a defconcl1b defconcl1bb ilexiste ou
defconcl2 defconcl2app defconcl2elt defconcl2a defconcl2b defconcl3

0 nouvelle conclusion
- (qqs(A, big_p(A) => big_r(A)) => qqs(B, big_q(B) => big_s(B))) et (qqs(B,
big_q(B) => big_s(B)) => qqs(A, big_p(A) => big_r(A))).

---------------------------------------------------------- regle <=>

********************************************** sous - theoreme 1 *****
1 nouvelle conclusion
- (qqs(A, big_p(A) => big_r(A)) => qqs(B, big_q(B) => big_s(B))).

------------------------------------------------ creation du sous - theoreme 1
1 ajouter la regle active locale

- (regle(A, reghyp) :- hyp(A, big_p(B)), not hyp(A, big_r(B)), ajhyp(A,
big_r(B))).

a la fin des regles universelles

1 nouvelle conclusion
- qqs(A, big_q(A) => big_s(A)).

---------------------------------------------------------- regle =>
1 ajouter objet x
1 nouvelle conclusion big_q(x) => big_s(x)
---------------------------------------------------------- regle qqs1
1 ajouter hypothese big_q(x)
1 nouvelle conclusion big_s(x)
---------------------------------------------------------- regle =>
1 ajouter hypothese
```

```
- ilexiste(A, big_p(A)).
---------------------------------------------------------- regle pel26_1_1
1 ajouter objet x1
1 ajouter hypothese big_p(x1)
1 ajouter hypothese-traitee
- ilexiste(A, big_p(A)).
---------------------------------------------------------- regle ilexiste
1 ajouter hypothese big_r(x1)
---------------------------------------------------------- regle reghyp
1 ajouter hypothese
- ilexiste(A, big_q(A)).
---------------------------------------------------------- regle pel26_1_
1 ajouter hypothese big_s(x)
---------------------------------------------------------- regle pel26_2_
1 nouvelle conclusion vrai
---------------------------------------------------------- regle stop1
theoreme 1 demontre
===================
0 nouvelle conclusion
- (qqs(A, big_q(A) => big_s(A)) => qqs(B, big_p(B) => big_r(B))).


********************************************** sous - theoreme 2 *****
2 nouvelle conclusion
- (qqs(A, big_q(A) => big_s(A)) => qqs(B, big_p(B) => big_r(B))).

------------------------------------------------ creation du sous - theoreme 2
2 ajouter la regle active locale

- (regle(A, reghyp1) :- hyp(A, big_q(B)), not hyp(A, big_s(B)), ajhyp(A,
big_s(B))).

a la fin des regles universelles

2 nouvelle conclusion
- qqs(A, big_p(A) => big_r(A)).

---------------------------------------------------------- regle =>
2 ajouter objet x2
2 nouvelle conclusion big_p(x2) => big_r(x2)
---------------------------------------------------------- regle qqs1
2 ajouter hypothese big_p(x2)
2 nouvelle conclusion big_r(x2)
---------------------------------------------------------- regle =>
2 ajouter hypothese
- ilexiste(A, big_q(A)).
---------------------------------------------------------- regle pel26_1_
2 ajouter objet x3
2 ajouter hypothese big_q(x3)
2 ajouter hypothese-traitee
- ilexiste(A, big_q(A)).
---------------------------------------------------------- regle ilexiste
2 ajouter hypothese big_s(x3)
---------------------------------------------------------- regle reghyp1
2 ajouter hypothese
- ilexiste(A, big_p(A)).
---------------------------------------------------------- regle pel26_1_1
2 ajouter hypothese big_r(x2)
---------------------------------------------------------- regle pel26_2_1
2 nouvelle conclusion vrai
---------------------------------------------------------- regle stop1
theoreme 2 demontre
===================
0 nouvelle conclusion vrai
---------------------------------------------------------- regle et
theoreme 0 demontre
===================
```

```
%------------------------------------------------------------------------------
% File     : MGT005+1 : TPTP v2.2.0. Released v2.0.0.
% Domain   : Management (Organisation Theory)
% Problem  : Complexity increases the risk of death due to reorganization.
% Version  : [PB+94] axioms.
% English  :
% Refs     : [PB+92] Peli et al. (1992), A Logical Approach to Formalizing
%          : [PB+94] Peli et al. (1994), A Logical Approach to Formalizing
%          : [Kam94] Kamps (1994), Email to G. Sutcliffe
% Source   : [Kam94]
% Names    : THEOREM 5 [PB+92]
% Status   : theorem
% Rating   : 0.50 v2.1.0
% Syntax   : Number of formulae    :   25 (   1 unit)
%            Number of atoms       :   72 (  28 equality)
%            Maximal formula depth :    4 (   3 average)
%            Number of connectives :   47 (   0 ~  ;   0  |;  23  &)
%                                         (   0 <=>;  24 =>;   0 <=)
%                                         (   0 <~>;   0 ~|;   0 ~&)
%            Number of predicates  :    9 (   0 propositional; 2-3 arity)
%            Number of functors    :    0 (   0 constant; --- arity)
%            Number of variables   :   90 (   0 singleton;  90 !;   0 ?)
%            Maximal term depth    :    1 (   1 average)
% Comments : mp11, mp12 and mp13 corrospond to mp10, mp11 and mp12
%            respectively from [PB+92]
%------------------------------------------------------------------------------
%----Include equality axioms
include('Axioms/EQU001+0.ax').
%------------------------------------------------------------------------------
%----Substitution axioms
input_formula(class_substitution_1,axiom,(
    ! [A,B,C,D] :
      ( ( equal(A,B)
        & class(A,C,D) )
     => class(B,C,D) )   )).

..... etc .....

%----Problem axioms
input_formula(mp6_1,axiom,(
    ! [X,Y] :
      ~ ( greater(X,Y)
        & equal(X,Y) )   )).

input_formula(mp6_2,axiom,(
    ! [X,Y] :
      ~ ( greater(X,Y)
        & greater(Y,X) )   )).

input_formula(mp11,axiom,(
    ! [X,Y,Z] :
      ( ( greater(X,Y)
        & greater(Y,Z) )
     => greater(X,Z) )   )).

input_formula(mp12,axiom,(
    ! [X,T] :
      ( organization(X,T)
     => ? [P] : survival_chance(X,P,T) )   )).

input_formula(mp13,axiom,(
    ! [X,T,T1,T2] :
      ( ( organization(X,T1)
        & organization(X,T2)
        & greater(T,T1)
```

```
        & greater(T2,T) )
     => organization(X,T) )   )).

input_formula(mp7,axiom,(
    ! [X,Ta,Tb] :
      ( reorganization(X,Ta,Tb)
     => greater(Tb,Ta) )   )).

input_formula(t3_FOL,hypothesis,(
    ! [X,P1,P2,T1,T2] :
      ( ( organization(X,T1)
        & organization(X,T2)
        & reorganization_free(X,T1,T2)
        & survival_chance(X,P1,T1)
        & survival_chance(X,P2,T2)
        & greater(T2,T1) )
     => greater(P2,P1) )   )).

%----t4_FOL - alias a9_FOL
input_formula(t4_FOL,hypothesis,(
    ! [X,P1,P2,T1,T2,Ta,Tb] :
      ( ( organization(X,T1)
        & organization(X,T2)
        & reorganization(X,Ta,Tb)
        & survival_chance(X,P1,T1)
        & survival_chance(X,P2,T2)
        & ~ greater(Ta,T1)
        & greater(T2,T1)
        & ~ greater(T2,Tb) )
     => greater(P1,P2) )   )).

%----Complexity increases the expected duration of reorganization.
input_formula(a10_FOL,hypothesis,(
    ! [X,Y,Re,C,C1,C2,Ta,Tb,Tc] :
      ( ( organization(X,Ta)
        & organization(Y,Ta)
        & organization(Y,Tc)
        & class(X,C,Ta)
        & class(Y,C,Ta)
        & reorganization(X,Ta,Tb)
        & reorganization(Y,Ta,Tc)
        & reorganization_type(X,Re,Ta)
        & reorganization_type(Y,Re,Ta)
        & complexity(X,C1,Ta)
        & complexity(Y,C2,Ta)
        & greater(C2,C1) )
     => greater(Tc,Tb) )   )).

%----Complexity is no survival advantage during reorganization.
input_formula(a11_FOL,hypothesis,(
    ! [X,Y,Re,C,P,P1,P2,C1,C2,Ta,Tb,Tc] :
      ( ( organization(X,Ta)
        & organization(Y,Ta)
        & organization(X,Tb)
        & organization(Y,Tb)
        & class(X,C,Ta)
        & class(Y,C,Ta)
        & survival_chance(X,P,Ta)
        & survival_chance(Y,P,Ta)
        & reorganization(X,Ta,Tb)
        & reorganization(Y,Ta,Tc)
        & reorganization_type(X,Re,Ta)
        & reorganization_type(Y,Re,Ta)
        & survival_chance(X,P1,Tb)
        & survival_chance(Y,P2,Tb)
        & complexity(X,C1,Ta)
        & complexity(Y,C2,Ta)
        & greater(C2,C1) )
     => ( greater(P1,P2)
```

```
        | equal(P1,P2) ) )   )).

input_formula(t5_FOL,conjecture,(
    ! [X,Y,Re,C,P,P1,P2,C1,C2,Ta,Tb,Tc] :
      ( ( organization(X,Ta)
        & organization(Y,Ta)
        & organization(X,Tc)
        & organization(Y,Tc)
        & class(X,C,Ta)
        & class(Y,C,Ta)
        & survival_chance(X,P,Ta)
        & survival_chance(Y,P,Ta)
        & reorganization(X,Ta,Tb)
        & reorganization(Y,Ta,Tc)
        & reorganization_type(X,Re,Ta)
        & reorganization_type(Y,Re,Ta)
        & reorganization_free(X,Tb,Tc)
        & survival_chance(X,P1,Tc)
        & survival_chance(Y,P2,Tc)
        & complexity(X,C1,Ta)
        & complexity(Y,C2,Ta)
        & greater(C2,C1) )
     => greater(P1,P2) )   )).
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
% File     : MGT021+1 : TPTP v2.2.0. Released v2.0.0.
% Problem  : Difference between disbanding rates does not decrease
%--------------------------------------------------------------------------
%----Subsitution axioms
...
%----Problem axioms
%----MP. If first movers and efficient producers are present in an
%----environment  at a certain point of time, then this time-point belongs
%----to the the environment.
input_formula(mp_time_point_in_environment,axiom,(
    ! [E,T] :
      ( ( environment(E)
        & subpopulations(first_movers,efficient_producers,E,T) )
     => in_environment(E,T) )   )).
%----MP. If first movers and efficient producers are present in an
%----environment  at a certain point of time, then then the environment
%----is not empty at this time.
input_formula(mp_environment_not_empty,axiom,(
    ! [E,T] :
      ( ( environment(E)
        & subpopulations(first_movers,efficient_producers,E,T) )
     => greater(number_of_organizations(E,T),zero) )   )).
%----MP. If something increases, then it does not decrease.
input_formula(mp_increase_not_decrease,axiom,(
    ! [X] :
      ( increases(X)
     => ~ decreases(X) )   )).
%----MP. on "greater or equal to"
input_formula(mp_greater_or_equal,axiom,(
    ! [X,Y] :
      ( greater_or_equal(X,Y)
     => ( greater(X,Y)
        | equal(X,Y) ) )   )).
%----A3. Resource availability decreases until equilibrium is reached.
input_formula(a3,hypothesis,(
    ! [E,T] :
      ( ( environment(E)
        & in_environment(E,T)
        & greater(number_of_organizations(E,T),zero) )
     => ( ( greater(equilibrium(E),T)
         => decreases(resources(E,T)) )
        & ( ~ greater(equilibrium(E),T)
         => constant(resources(E,T)) ) ) )   )).
```

```
%----L4. A decreasing resource availability affects the disbanding rate of
%----first movers more than the disbanding rate of efficient producers.
input_formula(l4,hypothesis,(
    ! [E,T] :
      ( ( environment(E)
        & subpopulations(first_movers,efficient_producers,E,T) )
     => ( ( decreases(resources(E,T))
         => increases(difference(disbanding_rate(first_movers,T),
disbanding_rate(efficient_producers,T))) )
        & ( constant(resources(E,T))
         => ~ decreases(difference(disbanding_rate(first_movers,T),
disbanding_rate(efficient_producers,T))) ) ) )   )).
%----GOAL: L3. The difference between the disbanding rates of first movers
%----and efficient producers does not decrease.
input_formula(prove_l3,conjecture,(
    ! [E,T] :
      ( ( environment(E)
        & subpopulations(first_movers,efficient_producers,E,T) )
     => ~ decreases(difference(disbanding_rate(first_movers,T),
disbanding_rate(efficient_producers,T))) )   )).
%----------------------------------------------------------------------------

%----------------------------------------------------------------------------
% File     : MGT023+1 : TPTP v2.2.0. Released v2.0.0.
% Problem  : Stable environments have a critical point.
%----------------------------------------------------------------------------
%----Subsitution axioms
...
%----Problem axioms
%----D1=>. A time point is the critical point of an environmental patch,
%----if and only if, it is the earliest time past which the growth rate of
%----efficient producers permanently exceeds growth rate of first movers.
input_formula(d1,hypothesis,(
    ! [E,To] :
      ( ( environment(E)
        & ~ greater(growth_rate(efficient_producers,To),
growth_rate(first_movers,To))
        & in_environment(E,To)
        & ! [T] :
            ( ( subpopulations(first_movers,efficient_producers,E,T)
              & greater(T,To) )
           => greater(growth_rate(efficient_producers,T),
growth_rate(first_movers,T)) ) )
     => equal(To,critical_point(E)) )   )).
%----L12. There is an earliest time point, past which FM's growth rate
%----exceeds EP's growth rate.
input_formula(l12,hypothesis,(
    ! [E] :
      ( ( environment(E)
        & stable(E) )
     => ? [To] :
          ( in_environment(E,To)
          & ~ greater(growth_rate(efficient_producers,To),
growth_rate(first_movers,To))
          & ! [T] :
              ( ( subpopulations(first_movers,efficient_producers,E,T)
                & greater(T,To) )
             => greater(growth_rate(efficient_producers,T),
growth_rate(first_movers,T)) ) ) )   )).

%----GOAL: L5. Stable environments have a critical point.
input_formula(prove_l5,conjecture,(
    ! [E] :
      ( ( environment(E)
        & stable(E) )
     => in_environment(E,critical_point(E)) )   )).
%----------------------------------------------------------------------------
```

```
%--------------------------------------------------------------------------
% File     : COM003+2 : TPTP v2.2.0. Bugfixed v2.2.0.
% Domain   : Computing Theory
% Problem  : The halting problem is undecidable
% Version  : [Bru91] axioms.
% English  :
% Refs     : [Gan98] Ganzinger (1998), Email to Geoff Sutcliffe
%          : [Bur87b] Burkholder (1987), A 76th Automated Theorem Proving Pr
%          : [Bru91] Brushi (1991), The Halting Problem
% Source   : [Bru91]
% Names    : - [Bru91]
% Status   : theorem
% Rating   : ? v2.2.0
% Syntax   : Number of formulae    :   16 (   1 unit)
%            Number of atoms       :   52 (   0 equality)
%            Maximal formula depth :    8 (   4 average)
%            Number of connectives :   39 (   3 ~  ;   0  |;  15  &)
%                                         (  11 <=>;  10 =>;   0 <=)
%                                         (   0 <~>;   0 ~|;   0 ~&)
%            Number of predicates  :   17 (   0 propositional; 1-4 arity)
%            Number of functors    :    2 (   2 constant; 0-0 arity)
%            Number of variables   :   42 (   0 singleton;  37 !;   7 ?)
%            Maximal term depth    :    1 (   1 average)
% Comments :
% Bugfixes : v2.2.0 - Clauses program_halts2_halts3_outputs_def, program_
%            not_halts2_halts3_outputs_def, program_halts2_halts2_outputs_
%            def, program_not_halts2_halts2_outputs_def, p4 by [Gan98].
%--------------------------------------------------------------------------
input_formula(program_decides_def,axiom,(
    ! [X] :
      ( program_decides(X)
    <=> ! [Y] :
          ( program(Y)
         => ! [Z] : decides(X,Y,Z) ) )   )).

input_formula(program_program_decides_def,axiom,(
    ! [X] :
      ( program_program_decides(X)
    <=> ( program(X)
        & program_decides(X) ) )   )).

input_formula(algorithm_program_decides_def,axiom,(
    ! [X] :
      ( algorithm_program_decides(X)
    <=> ( algorithm(X)
        & program_decides(X) ) )   )).

input_formula(program_halts2_def,axiom,(
    ! [X,Y] :
      ( program_halts2(X,Y)
    <=> ( program(X)
        & halts2(X,Y) ) )   )).

input_formula(halts3_outputs_def,axiom,(
    ! [X,Y,Z,W] :
      ( halts3_outputs(X,Y,Z,W)
    <=> ( halts3(X,Y,Z)
        & outputs(X,W) ) )   )).

input_formula(program_not_halts2_def,axiom,(
    ! [X,Y] :
      ( program_not_halts2(X,Y)
    <=> ( program(X)
        & ~ halts2(X,Y) ) )   )).
```

```
input_formula(halts2_outputs_def,axiom,(
    ! [X,Y,W] :
      ( halts2_outputs(X,Y,W)
    <=> ( halts2(X,Y)
        & outputs(X,W) ) )   )).

input_formula(program_halts2_halts3_outputs_def,axiom,(
    ! [X,Y,Z,W] :
      ( program_halts2_halts3_outputs(X,Y,Z,W)
    <=> ( program_halts2(Y,Z)
       => halts3_outputs(X,Y,Z,W) ) )   )).

input_formula(program_not_halts2_halts3_outputs_def,axiom,(
    ! [X,Y,Z,W] :
      ( program_not_halts2_halts3_outputs(X,Y,Z,W)
    <=> ( program_not_halts2(Y,Z)
       => halts3_outputs(X,Y,Z,W) ) )   )).

input_formula(program_halts2_halts2_outputs_def,axiom,(
    ! [X,Y,W] :
      ( program_halts2_halts2_outputs(X,Y,W)
    <=> ( program_halts2(Y,Y)
       => halts2_outputs(X,Y,W) ) )   )).

input_formula(program_not_halts2_halts2_outputs_def,axiom,(
    ! [X,Y,W] :
      ( program_not_halts2_halts2_outputs(X,Y,W)
    <=> ( program_not_halts2(Y,Y)
       => halts2_outputs(X,Y,W) ) )   )).

input_formula(p1,axiom,(
    ? [X] : algorithm_program_decides(X)
 => ? [W] : program_program_decides(W)   )).

input_formula(p2,axiom,(
    ! [W] :
      ( program_program_decides(W)
     => ! [Y,Z] :
          ( program_halts2_halts3_outputs(W,Y,Z,good)
          & program_not_halts2_halts3_outputs(W,Y,Z,bad) ) )   )).

input_formula(p3,axiom,(
    ? [W] :
      ( program(W)
      & ! [Y] :
          ( program_halts2_halts3_outputs(W,Y,Y,good)
          & program_not_halts2_halts3_outputs(W,Y,Y,bad) ) )
 => ? [V] :
      ( program(V)
      & ! [Y] :
          ( program_halts2_halts2_outputs(V,Y,good)
          & program_not_halts2_halts2_outputs(V,Y,bad) ) )   )).

input_formula(p4,axiom,(
    ? [V] :
      ( program(V)
      & ! [Y] :
          ( program_halts2_halts2_outputs(V,Y,good)
          & program_not_halts2_halts2_outputs(V,Y,bad) ) )
 => ? [U] :
      ( program(U)
      & ! [Y] :
          ( ( program_halts2(Y,Y)
           => ~ halts2(U,Y) )
          & program_not_halts2_halts2_outputs(U,Y,good) ) )   )).

input_formula(prove_this,conjecture,(
    ~ ( ? [X] : algorithm_program_decides(X) )   )).
%----------------------------------------------------------------------------
```

$A \subset B \Leftrightarrow \forall X\,(X \in A \Rightarrow X \in B)$

```
definition( A inc B <=> qqs(X,(X app A => X app B))).
```

```
input_formula(subset,axiom,(
    ! [A,B] :
      ( subset(A,B)
    <=> ! [X] :
          ( member(X,A)
         => member(X,B) ) )   )).
```

$A \cap B = \{X \mid X \in A \wedge X \in B)$

```
definition(A inter B = [X,X app A et X app B]).
```

```
input_formula(intersection,axiom,(
    ! [X,A,B] :
      ( member(X,intersection(A,B))
    <=> ( member(X,A)
        & member(X,B) ) )   )).
```

produit(A) = $\{Y \mid \forall X\,(X \in A \Rightarrow Y \in X)\} = \bigcap_{X \in A} X$

```
definition( produit(A) = [Y, qqs(X, X app A => Y app X)]).
```

```
input_formula(product,axiom,(
    ! [X,A] :
      ( member(X,product(A))
    <=> ! [Y] :
          ( member(Y,A)
         => member(X,Y) ) )   )).
```

$\forall X \in A\ \forall Z \in C\ (Z : H(X) \Leftrightarrow \exists X \in B\ (Y : F(X) \wedge Z : G(Y))$

```
definition(comp(G,F,A,B,C):H <=> qqs(X app A, qqs(Z app C,
                H(X):Z <=> ilexiste(Y app B, F(X):Y et G(Y):Z)))).
```

```
definition(comp(G,F,A,B,C):H <=> qqs(X app A, qqs(Z app C,
            ..[H,X]:Z <=> ilexiste(Y app B, ..[F,X]:Y et ..[G,Y]:Z)))).
```

```
definition(comp(G,F,A,B,C)=H <=> qqs(X app A, qqs(Z app C,                FAUX !
            ..[H,X]:Z <=> ilexiste(Y app B, ..[F,X]:Y et ..[G,Y]:Z)))).
```

nouvel énoncé : $\forall X \in A\ \forall Z \in C\ (Z : G \circ F_{A,B,C}(X) \Leftrightarrow \exists X \in B\ (Y : F(X) \wedge Z : G(Y))$

```
     qqs(X app A, qqs(Z app C, ..[comp(G,F,A,B,C),X]:Z <=>
                              ilexiste(Y app B, ..[F,X]:Y) et ..[G,Y]:Z)))).
```

```
input_formula(compose_function,axiom,(
    ! [G,F,A,B,C,X,Z] :
      ( ( member(X,A)
        & member(Z,C) )
     => ( apply(compose_function(G,F,A,B,C),X,Z)
       <=> ? [Y] :
             ( member(Y,B)
             & apply(F,X,Y)
             & apply(G,Y,Z) ) ) )   )).
```

transitive(R,E) $\Leftrightarrow \forall X \in E\ \forall Y \in E\ \forall Z \in E\ (R(X,Y) \Rightarrow R(Y,Z) \wedge R(X,Z))$

```
definition(transitive(R,E) <=>
            qqs(X app E,qqs(Y app E,qqs(Z app E,
                        (..[R,X,Y] et ..[R,Y,Z] => ..[R,X,Z])))))).
```

ANNEXE 6 Démonstrations du théorème SYN332+1 (monitoring)

Soit $F = \exists X\, \exists Y\, \forall Z \big([f(X,Y) \wedge f(Y,X) \Leftrightarrow f(X,Z)]$ (1)
$\Rightarrow \big[[f(X,Z) \Leftrightarrow f(Z,X)]$
$\Rightarrow \big([f(X,Z) \Leftrightarrow f(Y,Z)]$
$\Rightarrow \big[\big([f(Y,X) \Rightarrow f(X,Y)] \Leftrightarrow f(Z,Z) \big)$
$\Rightarrow \big([f(X,Y) \Leftrightarrow f(Y,X)] \Leftrightarrow f(Z,Y) \big) \big] \big] \big)$

Cet énoncé est d'abord réécrit de la façon suivante (assez naturelle et que MUSCADET est capable de faire) :

$F \equiv \exists X\, \exists Y\, \forall Z \big([f(X,Y) \wedge f(Y,X) \Leftrightarrow f(X,Z)]$ (2)
$\wedge [f(X,Z) \Leftrightarrow f(Z,X)]$
$\wedge [f(X,Z) \Leftrightarrow f(Y,Z)]$
$\wedge \big([f(Y,X) \Rightarrow f(X,Y)] \Leftrightarrow f(Z,Z) \big)$
$\Rightarrow \big([f(X,Y) \Leftrightarrow f(Y,X)] \Leftrightarrow f(Z,Y) \big) \big)$

1.
Je me suis aidée de considérations sémantiques et j'ai réécrit cette formule sous la forme suivante, équivalente[1] :
$\exists X\, \exists Y\, [f(X,Y) \wedge f(Y,X) \wedge \forall Z\, (f(X,Z) \wedge f(Z,X) \wedge f(Y,Z) \wedge f(Z,Z) \Rightarrow f(Z,Y))]$ (3)
$\vee\, \exists X\, \exists Y\, [f(X,Y) \wedge \neg f(Y,X) \wedge \forall Z\, (\neg f(X,Z) \wedge \neg f(Z,X) \wedge \neg f(Y,Z) \wedge f(Z,Z) \Rightarrow \neg f(Z,Y))]$
$\vee\, \exists X\, \exists Y\, [\neg f(X,Y) \wedge f(Y,X) \wedge \forall Z\, (\neg f(X,Z) \wedge \neg f(Z,X) \wedge \neg f(Y,Z) \wedge \neg f(Z,Z) \Rightarrow \neg f(Z,Y))]$
$\vee\, \exists X\, \exists Y\, [\neg f(X,Y) \wedge \neg f(Y,X) \wedge \forall Z\, (\neg f(X,Z) \wedge \neg f(Z,X) \wedge \neg f(Y,Z) \wedge f(Z,Z) \Rightarrow f(Z,Y))]$
et j'ai fait le raisonnement suivant :
- si on a $\exists X\, f(X,X)$, soit a tel que f(a,a), en prenant X=Y=a la première sous-formule est vérifiée
- sinon, on a $\forall X \neg f(X,X)$, la formule devient alors
  $\exists X\, \exists Y\, [f(X,Y) \wedge f(Y,X)]$
  $\vee\, \exists X\, \exists Y\, [f(X,Y) \wedge \neg f(Y,X)]$
  $\vee\, \exists X\, \exists Y\, [\neg f(X,Y) \wedge f(Y,X) \wedge \forall Z\, (\neg f(X,Z) \wedge \neg f(Z,X) \wedge \neg f(Y,Z) \Rightarrow \neg f(Z,Y))]$
  $\vee\, \exists X\, \exists Y\, [\neg f(X,Y) \wedge \neg f(Y,X)]$
  - soient deux éléments a et b s'ils existent
    - si f(a,b) = f(b,a), en prenant X=a et Y=b, la première ou la quatrième sous-formule est vérifiée
    - si on a f(a,b) et $\neg$ f(b,a), en prenant X=a et Y=b, la deuxième sous-formule est vérifiée
    - si on a $\neg$ f(a,b) et f(b,a), en prenant X=b et Y=a, la deuxième sous-formule est vérifiée
  - s'il n'existe pas deux éléments distincts, mais un seul[2], a, on a $\neg$ f(a,a) et en prenant X=Y=a, la quatrième sous-formule est vérifiée.

Ce raisonnement ayant permis de trouver l'idée de faire un raisonnement par cas, $\exists X\, f(X,X)$ ou bien $\forall X \neg f(X,X)$, on peut maintenant faire un raisonnement plus formel.

[1] L'idée et de considérer explicitement les quatre cas possibles pour f(X,Y) et f(Y,X)
[2] Il y en a toujours au moins un car le domaine d'une interprétation est supposé non vide.

2.
$\neg$ F peut s'écrire
$\forall X\ \forall Y\ [f(X,Y) \wedge f(Y,X) \Rightarrow \exists Z\ (f(X,Z) \wedge f(Z,X) \wedge f(Y,Z) \wedge f(Z,Z) \wedge \neg f(Z,Y))]$ (3')
$\wedge\ \forall X\ \forall Y\ [f(X,Y) \wedge \neg f(Y,X) \Rightarrow \exists Z\ (\neg f(X,Z) \wedge \neg f(Z,X) \wedge \neg f(Y,Z) \wedge f(Z,Z) \wedge f(Z,Y))]$
$\wedge\ \forall X\ \forall Y\ [\neg f(X,Y) \wedge f(Y,X) \Rightarrow \exists Z\ (\neg f(X,Z) \wedge \neg f(Z,X) \wedge \neg f(Y,Z) \wedge \neg f(Z,Z) \wedge f(Z,Y))]$
$\wedge\ \forall X\ \forall Y\ [\neg f(X,Y) \wedge \neg f(Y,X) \Rightarrow \exists Z\ (\neg f(X,Z) \wedge \neg f(Z,X) \wedge \neg f(Y,Z) \wedge f(Z,Z) \wedge \neg f(Z,Y))]$
On démontre alors facilement, à partir de la première sous-formule, que
$\neg F \Rightarrow \forall X\ [f(X,X) \Rightarrow \exists Z\ (f(X,Z) \wedge f(Z,X) \wedge f(Z,Z) \wedge \neg f(Z,X)]$
d'où $\neg F \Rightarrow \forall X \neg f(X,X)$
puis, à partir de la quatrième sous-formule, que
$\neg F \wedge \forall X \neg f(X,X) \Rightarrow \forall X\ \forall Y\ [f(X,Y) \vee f(Y,X)]$
puis $\neg F \wedge \forall X \neg f(X,X) \Rightarrow \forall X\ f(X,X)$
donc $\neg F \Rightarrow \forall X \neg f(X,X) \wedge \forall X\ f(X,X)$
$\neg$ F est donc fausse et F est un théorème.

Je me suis alors aperçue que le premier raisonnement utilisait la première, la deuxième et la quatrième sous-formules alors que le deuxième raisonnement n'utilisait que la première et la quatrième sous-formules. En en cherchant les raisons, j'ai simplifié la première démonstration dont la fin devient:

3. comme 1. puis
- si on a $\exists X\ f(X,X)$, soit a tel que f(a,a), en prenant X=Y=a la première sous-formule est vérifiée
- sinon, on a $\forall X \neg f(X,X)$, soit a quelconque, en prenant X=Y=a la quatrième sous-formule est vérifiée.

Il est intéressant de remarquer que le premier raisonnement, basé sur la sémantique et la recherche d'interprétation a permis de trouver le deuxième raisonnement plus formel qui a permis de simplifier le premier raisonnement.

Cette démonstration ne me satisfaisait pas vraiment, car je considérais quatre sous-formules correspondant à quatre cas dont deux étaient inutiles. Je me suis alors aperçue, que l'on pouvait, version sémantique, conclure directement que

4.
- si on a $\exists X\ f(X,X)$, soit a tel que f(a,a), en prenant X=Y=a, F (sous la forme (2)) devient
$\forall Z\ (f(a,Z) \wedge f(Z,a) \wedge f(Z,Z) \Rightarrow f(Z,a))$ qui est vraie
- sinon, on a $\forall X \neg f(X,X)$, soit a quelconque, en prenant X=Y=a, F devient
$\forall Z\ (\neg f(a,Z) \wedge \neg f(Z,a) \wedge f(Z,Z) \Rightarrow f(Z,a))$ qui est vraie puisque f(Z,Z) est fausse.

Pour retrouver une démonstration formelle, après quelques manipulations, et après avoir remarqué que les interprétations conduisaient à toujours prendre X=Y, j'ai introduit la formule $G = \exists X\ \forall Z\ \big(\ [f(X,X) \Leftrightarrow f(X,Z)] \wedge [f(X,Z) \Leftrightarrow f(Z,X)] \wedge [f(X,Z) \Leftrightarrow f(X,Z)]$
$\wedge\ ([f(X,X) \Rightarrow f(X,X)] \Leftrightarrow f(Z,Z))$
$\Rightarrow\ (\ [f(X,X) \Leftrightarrow f(X,X)] \Leftrightarrow f(Z,X))\ \big)$
équivalente à

$\exists X\ \forall Z\ \big([f(X,X) \Leftrightarrow f(X,Z)] \wedge [f(X,Z) \Leftrightarrow f(Z,X)] \wedge f(Z,Z) \Rightarrow f(Z,X)\big)$

On a $G \Rightarrow F$ et $\neg F \Rightarrow \neg G$
Il suffit de montrer que G est un théorème ou que $\neg G$ n'en est pas un.
Le plus facile est de démontrer que $\neg G$ n'est pas un théorème, soit

5.

$$\neg G \equiv \forall X\ \exists Z\ \big([f(X,X) \Leftrightarrow f(X,Z)] \wedge [f(X,Z) \Leftrightarrow f(Z,X)] \wedge f(Z,Z) \wedge \neg f(Z,X)\big)$$
$$\equiv \forall X\ \exists Z\ \big([f(X,X) \wedge f(X,Z) \wedge f(Z,X) \wedge f(Z,Z) \wedge \neg f(Z,X)]$$
$$\vee [\neg f(X,X) \wedge \neg f(X,Z) \wedge \neg f(Z,X) \wedge f(Z,Z)\ ]\big)$$
$$\equiv \forall X\ \exists Z\ \big(\neg f(X,X) \wedge \neg f(X,Z) \wedge \neg f(Z,X) \wedge f(Z,Z)\big)$$
$$\equiv \forall X \neg f(X,X) \wedge \forall X\ \exists Z\ \big(\neg f(X,Z) \wedge \neg f(Z,X) \wedge f(Z,Z)\big)$$

qui est fausse car le f(Z,Z) de la deuxième sous-formule est faux à cause de la première sous-formule.

On peut montrer directement que G est un théorème :

6.

$$G = \exists X\ \forall Z\ \big([f(X,X) \Leftrightarrow f(X,Z)] \wedge [f(X,Z) \Leftrightarrow f(Z,X)] \wedge [f(X,Z) \Leftrightarrow f(X,Z)] \wedge f(Z,Z) \Rightarrow f(Z,X)\big)$$
$$\equiv \exists X\ \forall Z\ \big([f(X,X) \wedge f(X,Z) \wedge f(Z,X) \wedge f(Z,Z) \Rightarrow f(Z,X)]$$
$$\wedge [\neg f(X,X) \wedge \neg f(X,Z) \wedge \neg f(Z,X) \wedge f(Z,Z) \Rightarrow f(Z,X)]\ \big)$$
$$\equiv \exists X\ \forall Z\ (\neg f(X,X) \wedge \neg f(X,Z) \wedge \neg f(Z,X) \wedge f(Z,Z) \Rightarrow f(Z,X)\ )$$
$$\equiv \exists X\ \forall Z\ (\ f(X,X) \vee f(X,Z) \vee f(Z,X) \vee \neg f(Z,Z))$$
$$\equiv \exists X\ f(X,X) \vee \exists X\ \forall Z\ (\ f(X,Z) \vee f(Z,X) \vee \neg f(Z,Z)\ )$$
$$\equiv \exists X\ f(X,X) \vee \big(\forall X \neg f(X,X) \wedge \exists X\ \forall Z\ (\ f(X,Z) \vee f(Z,X) \vee \neg f(Z,Z)\ )\ \big)$$
$$\equiv \exists X\ f(X,X) \vee \forall X \neg f(X,X)$$

qui est vraie, ainsi que F puisqu'on a $G \Rightarrow F$

Cette dernière démonstration est celle que je considère comme la *meilleure*. Il faut insister sur le fait que c'est la succession des autres démonstrations qui m'a permis d'aboutir à celle-ci. Il faut aussi remarquer que toutes les démonstrations utilisent de nombreux savoir-faire de manipulations symboliques et que celles-ci ont été guidées par la sémantique. L'apprentissage a aussi joué son rôle : au début, les sous-formules semblaient obscures, au bout d'un certain temps, elles sont devenues limpides …

On remarquera aussi que l'idée d'exploiter le cas X=Y (coalescence) est une heuristique classique en Intelligence artificielle (Lenat 1982, Pitrat 1990) mais dont l'utilisation doit être prudente.