Automatisation du Raisonnement Mathématique et Démonstration Automatique de Théorèmes

Dominique Pastre

Université René Descartes - Paris 5
UFR de mathématiques et informatique
Crip5
Cours DEA MIASH - 2002

Chapitre 1 : Déduction Naturelle - Travaux de W.W. Bledsoe

Déduction Naturelle
Travaux de W.W. Bledsoe

Dominique Pastre - DEA MIASH - Université René Descartes (Paris5) - 2002

1 Introduction

La démonstration automatique de théorèmes peut être étudiée d'un point de vue logique, on s'intéressera à des théorèmes de la forme

$$\forall x[P(x) \Rightarrow Q(x)] \wedge P(a) \Rightarrow Q(a)$$

par exemple, ou plutôt d'un point de vue mathématique, on s'intéressera plutôt alors à des théorèmes de la forme

$$\forall A\ \forall B\ \mathcal{P}(A \cup B) = \mathcal{P}(A) \cup \mathcal{P}(B)$$

où $\mathcal{P}(A)$ est l'ensemble des parties de A défini par l'énoncé

$$\forall A\ \forall X(X \in \mathcal{P}(A) \Leftrightarrow X \subset A)$$

ou encore d'un point de vue *résolution de problèmes* pour travailler dans tout domaine dont les connaissances peuvent s'exprimer dans le calcul propositionnel ou le calcul des prédicats.

Le travail logique peut être d'abord théorique, s'intéresser à la validité, à la complétude et à l'efficacité des méthodes utilisées, puis donner lieu à des applications dans n'importe quel domaine, mathématique ou autre. On peut aussi, d'un point de vue applicatif, sans renoncer bien sûr à la validité des méthodes employées, ne pas en exiger la complétude, et privilégier l'efficacité pratique à l'efficacité théorique, selon le principe qu'il vaut mieux échouer à démontrer certains théorèmes mais en démontrer un certain nombre d'autres en un temps raisonnable [1], plutôt qu'être théoriquement capable d'en démontrer beaucoup plus, mais dans un temps déraisonnable [2].

Le *Principe de Résolution* est la méthode la plus employée, à tel point que certains auteurs l'ont utilisée comme synonyme de *Démonstration Automatique de Théorèmes*. Mais d'autres méthodes sont aussi employées, comme la *Déduction Naturelle* ou comme des méthodes dites *naturelles* qui s'inspirent de la Déduction Naturelle sans en avoir la rigueur ni le formalisme et utilisent des méthodes proches de celles utilisées par les êtres humains.

Le Principe de Résolution et les principales stratégies utilisées ont été présentés dans le cours "Logique et Principe de Résolution" [15]. De nombreuses autres stratégies ont été définies pour en améliorer l'efficacité pratique.

Le but de ce cours-ci est de présenter et étudier des **méthodes naturelles**.

2 Historique

Le premier programme d'intelligence artificielle a été un programme de démonstration automatique de théorèmes en calcul propositionnel, appelé LOGIC THEORIST [12]. Le premier objectif de Newel, Shaw et Simon était en 1954 un programme qui puisse jouer aux échecs. Ce travail a d'abord

[1] Notion floue et subjective pouvant être selon les applications et le but poursuivi de quelques dixièmes de secondes à plusieurs heures ou même jours.

[2] Pouvant atteindre des années ou des millions d'années.

conduit à la création du langage IPL1 (ancêtre de LISP) en 1956 puis au LOGIC THEORIST, également en 1956. Le programme d'échecs, appelé NSS (pour Newel, Shaw, Simon) [13] a la même structure que LOGIC THEORIST et introduit les notions de situations souhaitables et d'heuristiques puis a été suivi du programe GPS (General Problem Solver [14]), resté longtemps célèbre, qui introduit la construction de buts et de sous-buts et étudie les différences entre situations.

En 1958, Newel et Simon pensent qu'avant 1968 un programme sera champion d'échecs et démontrera un important théorème de mathématiques.

En 1960, Gelertner [11] réalise un programme en géométrie. Ses performances ne sont pas très grandes mais les idées sont intéressantes. Il a en particulier donné une démonstration plus simple que la démonstration classique du théorème affirmant que si un triangle a deux côtés égaux alors il a deux angles égaux [3].

En 1960, Wang [17] réalise un démonstrateur utilisant des *séquents*, complet pour le calcul propositionnel, avec des résultats encourageants pour le calcul des prédicats du premier ordre. Quelques exemples de séquents et de règles seront donnés dans la section 3.

En 1965, c'est le grand tournant, le *Principe de Résolution* de Robinson [16] donne une méthode complète pour le calcul des prédicats du premier ordre. Son programme est utilisé en démonstration automatique de théorèmes mais aussi en vérification de programmes et pour la manipulation d'objets. De plus, le Principe de Résolution (dans une version restrictive et non complète) a donné naissance au langage PROLOG qui est devenu à la fois un langage de description logique et un langage de programmation. Les avantages du Principe de Résolution (on dira alors simplement "de la Résolution") sont sa simplicité et son efficacité théorique. Sa faiblesse est son manque d'efficacité pratique : qu'un théorème ne puisse pas être démontré ou puisse l'être dans trois millions d'années ne change rien pour un observateur humain! Il y eut ensuite beaucoup d'articles théoriques, et de nombreuses stratégies, complètes ou non, ont été écrites pour améliorer l'efficacité pratique. Beaucoup plus tard, grâce à de nombreuses autres techniques et aussi grâce à l'augmentation de puissance des ordinateurs, plusieurs démonstrateurs basés sur le Principe de Résolution ont maintenant de bons résultats pratiques en démonstration automatique, que ce soit pour des théorèmes de mathématiques ou en résolution de problèmes.

En 1971, Bledsoe réagit à l'engouement pour le Principe de Résolution en réalisant le démonstrateur PROVER [5] qui combine des méthodes naturelles (SPLIT et REDUCE) et la Résolution. Il décrit quelques résultats de son programme en théorie des ensembles, meilleurs que ceux obtenus avec la Résolution seule. En 1972 il ajoute un puis deux modules, IMPLY et HOA [7, 10], et n'utilise plus du tout la Résolution. Il introduit la terminologie de *Déduction Naturelle (Natural Deduction)* bien qu'il ne s'agisse pas exactement de la Déduction Naturelle, théorique, des logiciens.

3 Les séquents de Wang

Dans le programme de Wang [17], en 1960, un énoncé se présente ou la forme de *séquents*

$$\phi 1, \phi_2, ..., \phi_n \Rightarrow \psi_1, ..., \psi_m$$

qui correspondent à l'énoncé logique

$$\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_n \rightarrow \psi_1 \vee ... \vee \psi_m$$

Des schéma de règles transforment ces séquents, par exemple

[3] Dans la démonstration classique et enseignée, on suppose que $AB = AC$, on trace la médiane AI du triangle ABC, les deux triangles AIB et AIC sont égaux car il ont leurs trois côtés égaux donc leurs angles $\widehat{ABI}$ et $\widehat{ACI}$ sont égaux. Le programme, lui, considère simplement l'égalité des triangles ABC et ACB pour conclure à l'égalité des angles $\widehat{B}$ et $\widehat{C}$. Il semblerait que cette démonstration soit maintenant enseignée. Noter deux raisons qui font que le programme a découvert une démonstration plus simple que la démonstration classique. D'abord l'être humain est trop savant, dès qu'on lui parle de côtés égaux il pense *isocèle* et voit tout de suite la *hauteur-médiane-médiatrice*. D'autre part il ne pense pas à considérer le triangle ACB car, contrairement au programme, il le *voit* identique au triangle ABC.

$(... \Rightarrow ... , X \rightarrow Y, ...)$	devient	$(... , X \Rightarrow ... , Y, ...)$
$(... \Rightarrow ... , X \wedge Y, ...)$	donne	$(... \Rightarrow ... , X, ...)$
	et	$(... \Rightarrow ... , Y, ...)$
$(... , X \wedge Y, ... \Rightarrow ...)$	devient	$(... , X, Y, \ ... \Rightarrow ...)$
$(... , X \rightarrow Y, ... \Rightarrow ...)$	donne	$(... \Rightarrow X, ...)$
	et	$(... , Y, \ ... \Rightarrow ...)$
$(... , X, ... \Rightarrow ..., X, ...)$	devient	*vrai*

Un théorème à démontrer est ainsi remplacé par un ensemble de séquents qui devront tous être vrais.

Ce programme est complet pour le calcul propositionnel et a eu des résultats encourageants pour le calcul des prédicats.

4 Le Principe de Résolution de Robinson

En 1965, bien que simple et efficace sur le plan théorique (complet pour le calcul propositionel et le calcul des prédicats du premier ordre), le Principe de Résolution de Robinson [16] s'est révélé peu efficace sur le plan pratique à cause de la croisance exponentielle du nombre de clauses générées. Beaucoup de clauses sont inutiles ou redondantes. Il n'y a pas de but intermédiaire. Tout est sur le même plan, les prédicats intermédiaires comme les concepts importants.

Il y eut beaucoup de travaux dont certains seulement théoriques, de nombreuses améliorations, diverses stratégies [4] préservant ou non la complétude.

La restriction aux clauses de Horn est une simplification intéressante, qui a conduit à la définition du langage PROLOG, mais exclut les problèmes comportant des buts disjonctifs.

Nénamoins, grâce aux progrès des techniques et des ordinateurs, les systèmes basés sur le Principe de Résolution ont maintenant de bons résultats pratiques.

5 La Déduction Naturelle de Bledsoe

En 1971, Bledsoe [5] montre qu'un prétraitement par des *méthodes naturelles* conduit à des preuves plus rapides (et plus lisibles ...). Avant d'utiliser la Résolution, son programme PROVER applique d'abord les deux modules SPLIT, général pour les mathématiques, et REDUCE, qui comporte des réecritures générales ou spéciales pour la théorie des ensembles. Le traitement de l'égalité d'ensembles et de l'inclusion est également spécifique, ainsi que l'application de quelques théorèmes connus (*built-in*).

5.1 SPLIT

Ce module décompose et réécrit le théorème à démontrer.

$A \wedge B$	donne	deux sous-théorèmes A et B
$A \leftrightarrow B$	devient	$(A \rightarrow B) \wedge (B \rightarrow A)$
$\forall x P(x)$	"	$P(x)$
$A = B$	"	*vrai* si A est identique à B
$p \rightarrow (A \rightarrow B)$	"	$p \wedge A \rightarrow B$
$p \rightarrow A \wedge B$	"	$(p \rightarrow A) \wedge (p \rightarrow B)$
$p \vee q \rightarrow A$	"	$(p \rightarrow A) \wedge (q \rightarrow A)$
$A \rightarrow \forall x P(x)$	"	$A \rightarrow P(y)$ où y est une nouvelle variable
$\exists x P(x) \rightarrow D$	"	$P(y) \rightarrow D$ "
$x = y \rightarrow P(x, y)$	"	$P(y, y)$ si y est une variable

De plus, une subroutine OR-OUT essaie de convertir les formules en $p \vee q \rightarrow A$ ou $p \rightarrow A \wedge B$ de façon à permettre l'application de SPLIT.

[4] stratégies positive, négative, de l'ensemble support ; hyper résolution, démodulation, paramodulation, weighting, etc.

Par exemple $A \wedge (B \vee C) \rightarrow D$ devient $(A \wedge B) \vee (A \wedge C) \rightarrow D$ qui pourra ainsi être découpé en $(A \wedge B) \rightarrow D$ et $(A \wedge C) \rightarrow D$

5.2 REDUCE

Il s'agit ici de réécritures spéciales pour la théorie des ensembles.

$s \in A \cap B$	devient	$s \in A \wedge s \in B$
$s \in A \cup B$	"	$s \in A \vee s \in B$
$s \in \mathcal{P}(A)$	"	$S \subset A$
$s \in \{A\}$	"	$S = A$
$s \in \sigma F$	"	$\exists y\ (y \in F \wedge x \in y)$
$C \subset A \cap B$	"	$C \subset A \wedge C \subset B$
$A \cup B \subset C$	"	$A \subset C \wedge B \subset C$
$\{C\} \subset A$	"	$C \in A$
$A \subset \phi$	"	$A = \phi$

ou générales

$A \wedge A$	devient	A
$A \vee A$	"	A
$\neg(A \wedge B)$	"	$\neg A \vee \neg B$
$\neg(A \vee B)$	"	$\neg A \wedge \neg B$
$\neg\ \forall x A$	"	$\exists x \neg A$
$\neg\ \exists x A$	"	$\forall x \neg A$
...		

5.3 Traitement de l'égalité et de l'inclusion d'ensembles

Définitions

$A = B$	devient	$A \subset B \wedge B \subset A$
$p \Rightarrow A = B$	"	$p \Rightarrow A \subset B \wedge B \subset A$
$A \subset B$	"	$\forall t\ (t \in A \Rightarrow t \in B)$
$p \Rightarrow A \subset B$	"	$p \Rightarrow \forall t\ (t \in A \Rightarrow t \in B)$

built-in théorèmes

$\phi \subset A$	devient	*vrai*

5.4 Induction

Un raisonnement par induction peut être effectué si une formule contient l'ensemble des entiers ω. Un prétraitement commence par la convertir dans la forme

$$N \in \omega \Rightarrow \phi(N)$$

ce qui donne deux formules

$$\phi(0)$$

$$\text{et } N \in \omega \wedge \phi(N) \Rightarrow \phi(suc(N))$$

qui sont démontrées séparément.

La difficulté est bien sûr de trouver la bonne hypothèse d'induction.

5.5 Contrôle

Le contrôle est programmé : PROVER applique successivement SPLIT, REDUCE, puis le Principe de Résolution avec un temps limite, puis essaie un raisonnement par induction, et enfin traite les égalités et inclusions.

5.6 Exemples

1. *Théorème à démontrer*

$$\mathcal{P}(A) \cap \mathcal{P}(B) = \mathcal{P}(A \cap B)$$

Démonstration
Remplacement de l'égalité

$$\mathcal{P}(A) \cap \mathcal{P}(B) \subset \mathcal{P}(A \cap B) \ \wedge \ \mathcal{P}(A \cap B) \subset \mathcal{P}(A) \cap \mathcal{P}(B)$$

Découpage

(1) $\mathcal{P}(A) \cap \mathcal{P}(B) \subset \mathcal{P}(A \cap B)$
$t \in \mathcal{P}(A) \cap \mathcal{P}(B) \Rightarrow t \in \mathcal{P}(A \cap B)$
$t \in \mathcal{P}(A) \wedge t \in \mathcal{P}(B) \Rightarrow t \in \mathcal{P}(A \cap B)$
$t \subset A \wedge t \subset B \Rightarrow t \subset A \cap B$
$t \subset A \wedge t \subset B \Rightarrow t \subset A \wedge t \subset B$
vrai car chaque partie de la conclusion est contenue dans les hypothèses

(2) $\mathcal{P}(A \cap B) \subset \mathcal{P}(A) \cap \mathcal{P}(B)$
démonstration analogue
vrai

Par la Résolution seule, la démonstration est longue et fastidieuse (une dizaine de clauses au départ).

2. *Théorème à démontrer*

$$\omega = \sigma\omega$$

où ω est l'ensemble des entiers,
avec comme théorèmes connus (reference theorems) [5]

$x \in suc(x)$
$x \in \omega \rightarrow suc(s) \in \omega$
$suc(x) = x \cup \{x\}$

Ce théorème est démontré par induction.

Remplacement de l'égalité et découpage en deux sous-théorèmes

(1)
$\omega \subset \sigma\omega$
$t \in \omega \rightarrow \exists x(x \in \omega \wedge t \in x)$
démontré par la Résolution

(2)
$\sigma\omega \subset \omega$
$\exists x(x \in \omega \wedge t \in x) \rightarrow t \in \omega$
$x \in \omega \wedge t \in x \rightarrow t \in \omega$
essaie l'induction en transformant en
$x \in \omega \rightarrow (t \in x \rightarrow t \in \omega)$
qui donne deux nouveaux sous-théorèmes (21) et (22)

(21)
$t \in 0 \rightarrow t \in \omega$
$false \rightarrow t \in \omega$
$true$

(22)
$x \in \omega \wedge \forall s(s \in x \rightarrow s \in \omega) \rightarrow (t \in suc(x) \rightarrow t \in \omega)$
$t \in suc(x) \wedge x \in \omega \wedge \forall s(s \in x \rightarrow s \in \omega) \rightarrow t \in \omega$
$t \in x \cup \{x\} \wedge x \in \omega \wedge \forall s(s \in x \rightarrow s \in \omega) \rightarrow t \in \omega$
$(t \in x \vee t = x) \wedge x \in \omega \wedge \forall s(s \in x \rightarrow s \in \omega) \rightarrow t \in \omega$
qui donne de nouveau deux sous-théorèmes (221) et (222)

(221)
$t \in x \wedge x \in \omega \wedge \forall s(s \in x \rightarrow s \in \omega) \rightarrow t \in \omega$
démontré par la Résolution

(222)
$(t = x \wedge x \in \omega \wedge \forall s(s \in x \rightarrow s \in \omega) \rightarrow t \in \omega$
$t \in \omega \wedge \forall s(s \in t \rightarrow s \in \omega) \rightarrow t \in \omega$
$true$

[5] En théorie des ensembles, les entiers sont définis de la façon suivantes : 0 est l'ensemble vide Φ, $n+1$ est $suc(n)$ qui est défini comme $n \cup \{n\}$

TAB. 1 – Quelques règles de IMPLY où IMPLY(E, R) est noté $[E, R]$

	INPUT				OUTPUT
1.	$[H \to C, R]$	If	$H = C$,	then	*true*
		If	there is a substitution σ which unifies H and C, (i.e. $H_\sigma = C_\sigma$)	then	σ
2.	$[A \wedge B, R]$	If	$[A, R]$ yields $\sigma 1$ and $[B, R]$ yields $\sigma 2$	then	$\sigma 1_o \sigma 2$
3.	$[A \vee B, R]$	If	$[A, R]$ yields $\sigma 1$	then	$\sigma 1$
		If	$[B, R]$ yields $\sigma 2$	then	$\sigma 2$
4.	$[(A \to B) \to C, R]$	If	$[B \to C, R]$ yields $\sigma 1$ and $[R \to A_{\sigma 1}, nil]$ yields $\sigma 2$	then	$\sigma 1_o \sigma 2$
5.	$[H \to (A \to B), R]$				$[H \wedge A \to B, R]$
6.	$[A \vee B \to C, R]$	If	$[A \to C, R]$ yields $\sigma 1$ and $[B_{\sigma 1} \to C, R]$ yields $\sigma 2]$	then	$\sigma 1_o \sigma 2$
7.	$[H \to A \vee B, R]$	If	$[H \to A, R]$ yields $\sigma 1$	then	$\sigma 1$
		If	$[H \to B, R]$ yields $\sigma 2$	then	$\sigma 2$
8.	$[H \to A \wedge B, R]$	If	$[H \to A, R]$ yields $\sigma 1$ and $[H \to B_{\sigma 1}, R]$ yields $\sigma 2]$	then	$\sigma 1_o \sigma 2$
9.	$[A \wedge B \to C, R$	If	$[A \to C, R \wedge B]$ yields $\sigma 1$	then	$\sigma 1$
		If	$[B \to C, R \wedge A]$ yields $\sigma 2$	then	$\sigma 2$
10.	$[H \to \neg A \vee B, R]$				$[H \wedge A \to B, R]$
11.	$(\neg A \wedge B \to C, R]$				$[B \to A \vee C, R]$
12.	$(\neg H \to C, R]$				$[R \to C \vee H, nil]$
13.	$(H \to \neg C, R]$				$[H \wedge C \to nil, R]$
14.	$(A = B \to C, R]$		R' and C' are gotten by replacing B by A in R and in C		$[R' \to C', nil]$
15.	$(H \to A = B, R]$	If	there is a substitution σ, which unifies A and B	then	σ

6 Les améliorations

6.1 IMPLY

Puis, dans [6], il n'y a plus du tout de Résolution, les précédentes techniques, et d'autres, sont effectuées par une procédure IMPLY(E, R) dont on trouvera un aperçu dans la Table 1.

E est l'expression examinée, R une *réserve*.

IMPLY cherche la substitution la plus générale σ telle que $(R \to E)_\sigma$ soit vraie et renvoie σ, ou *true* si σ est vide, ou *nil* en cas d'échec.

Pour démontrer E, on appelle IMPLY(E, nil) après avoir effectué une sorte de skolémisation partielle "à l'envers". Ce sont les transformations qui seraient faites sur la négation du théorème à démontrer :

- un quantificateur universel en conclusion (énoncé $\forall x P(x)$ ou $H \to \forall x P(x)$) donne une constante x_o, que l'on doit interpréter comme "soit x_o, montrer $P(x_o)$" ;
- un quantificateur existentiel en conclusion (énoncé $\exists x P(x)$ ou $H \to \exists x P(x)$) donne une variable x, que l'on doit interpréter comme "on cherche x tel que $P(x)$" ;

pour un énoncé $\exists x \forall y P(x, y)$, "on cherche x tel que $P(x, s(x))$" où s(x) est une fonction de Skolem ;

- un quantificateur universel en hypothèse (énoncé $\forall x P(x) \to C)$) donne une variable x, que l'on doit interpréter comme "on cherche x qui vérifie P et qui permettra de montrer C" ;
- un quantificateur existentiel en hypothèse (énoncé $\exists x P(x) \to C$) donne une constante x_o, que

l'on doit interpréter comme "on a x_o qui vérifie P, montrer C".

Exemple1
Théorème à démontrer

$$P(y) \wedge \forall x(P(x) \rightarrow Q(x)) \rightarrow Q(y)$$

transformé en

$$P(y_o) \wedge (P(x) \rightarrow Q(x)) \rightarrow Q(y_o)$$

Appels successifs de IMPLY :
1. IMPLY$(P(y_o) \wedge (P(x) \rightarrow Q(x)) \rightarrow Q(y_o), nil)$
   1.1 IMPLY$(P(y_o) \rightarrow Q(y_o), P(x) \rightarrow Q(x)))$ par règle 9
      échec
   1.2 IMPLY$((P(x) \rightarrow Q(x)) \rightarrow Q(y_o), P(y_o))$ par règle 9
      1.2.1 IMPLY$(Q(x) \rightarrow Q(y_o), P(y_o))$ par règle 4.1
         renvoie $\sigma = [y_o|x]$ par règle 1
      1.2.2 IMPLY$(P(y_o) \rightarrow P(x)_\sigma, nil)$ par règle 4.2
         renvoie *true* (règle 1)

Exemple2
Théorème à démontrer

$$\exists x \forall y P(x, y) \rightarrow \forall s \exists t P(t, s)$$

transformé en

$$P(x_o, y) \rightarrow P(t, s_o)$$

IMPLY$(P(x_o, y) \rightarrow P(t, s_o), nil)$ renvoie $[x_o|t, s_o|y]$

Exemple3
Non-théorème

$$\forall y \exists x P(x, y) \rightarrow \exists t \forall s P(t, s)$$

transformé en

$$P(x(y), y) \rightarrow P(t, s(t))$$

IMPLY$(P(x(y), y) \rightarrow P(t, s(t)), nil)$ échoue car on ne peut unifier y et $s(x(y))$ (test d'occurrence).

6.2 IMPLY et HOA

Puis, dans [10], IMPLY(E, R) est remplacé par deux procédures, IMPLY(H, C) (nouvelle version) et HOA(B, C), décrites dans les Tables 2 et 3, qui s'appellent récursivement l'une l'autre et renvoient une substitution θ telle que $H\theta = C\theta$, ou *true* si le théorème $H \rightarrow C$ a été démontré, ou *nil* s'il n'a pas été démontré.

La complétude n'est pas assurée.

Au départ, pour démontrer un énoncé E, on appelle IMPLY(nil, E).

IMPLY est spécialisé pour les découpages (conclusions conjonctives ou hypothèses disjonctives), diverses manipulations de la conclusion (négations, définitions) et le chainage en avant (forward chaining), optionnel.

HOA est spécialisé pour les disjonctions (conclusions disjonctives ou hypothèses conjonctives), le traitement des égalités et des nombres, et le chainage arrière (backward chaining), optionnel. Quand IMPLY ne peut rien faire, il appelle HOA.

Quand c'est possible, HOA transforme les énoncés pour qu'ils puissent être traités par les règles de IMPLY.

IMPLY : les règles sont données dans la Table 2

• Le REDUCE de [5] est enrichi de quelques réécritures, par exemple

$t \in \{x\|P(x)\}$	devient	$P(t)$	
$t \in A$	"	$P(t)$	si A est défini comme $\{x\|P(x)\}$
range $\lambda x f(x)$	"	$\{y \mid \exists x(y = f(x)\}$	

• Des définitions sont données (Table 4).
Elles sont skolémisées au sens de la section 6.1. Par exemple, en conclusion l'inclusion $A \subset B$, de définition

$$\forall x(x \in A \rightarrow x \in B)$$

est remplacée par

$$(x_o \in A \rightarrow x_o \in B) \quad .$$

en hypothèse, elle est remplacée par

$$(x \in A \rightarrow x \in B) \quad .$$

• "forward chaining" (étape 7.1), optionel, fonctionne ainsi :
si $P'\theta = P\theta$ alors une hypothèse $P' \wedge (P \rightarrow Q)$ devient $P' \wedge (P \rightarrow Q) \wedge Q\theta$
"PEEK forward chaining" (étape 7.2), optionnel, fonctionne ainsi :
si $P'\theta = P\theta$ et A a la définition $P \rightarrow Q$ alors une hypothèse $P' \wedge A$ devient $P' \wedge A \wedge Q\theta$

HOA : les règles sont données dans la Table 3.

• ANDS (étape 7) est une mini version de IMPLY.
• Le H de l'étape 7.2 est le H ayant appelé HOA.
• CHOOSE(a, b) est égal à a (ou à b), et OTHER(a, b) est égal à b (ou à a).

Exemple1
Théorème à démontrer

$$\forall y[P(y) \wedge \forall x(P(x) \rightarrow Q(x)) \rightarrow Q(y)]$$

transformé en

$$P(y_o) \wedge (P(x) \rightarrow Q(x)) \rightarrow Q(y_o)$$

Dans la suite on écrira $H \Rightarrow C$ pour un appel à IMPLY(H, C)
soit pour commencer [6]

	$nil \Rightarrow P(y_o) \wedge (P(x) \rightarrow Q(x)) \rightarrow Q(y_o)$	
par I7	$P(y_o) \wedge (P(x) \rightarrow Q(x)) \Rightarrow Q(y_o)$	
par H6	$P(y_o) \Rightarrow Q(y_o)$	renvoie nil
par H6.2	$(P(x) \rightarrow Q(x)) \Rightarrow Q(y_o)$	
par H7	ANDS$(Q(x), Q(y_o)$	renvoie $[y_o\|x]$ (back-chaining)
par I7.2	$P(y_o) \wedge (P(x) \rightarrow Q(x)) \Rightarrow P(y_o)$	
par H6.1 et H2		renvoie $true$
par H7.4	renvoie $[y_o\|x]$ pour (1)	

Avec les nouvelles notations de IMPLY/HOA, cette démonstration par chainage arrière est identique à celle de la section 6.1

Avec le chainage avant, optionnel, on obtient plus rapidement, à partir de

$$P(y_o) \wedge (P(x) \rightarrow Q(x)) \Rightarrow Q(y_o)$$

par I7.1 $P(y_o) \wedge (P(x) \rightarrow Q(x)) \wedge Q(y_o) \Rightarrow Q(y_o)$
puis comme précédemment par H6.2 et H2 puis H7.4

[6] In (resp. Hn) est la règle n de IMPLY (rep. HOA).

TAB. 2 – IMPLY(H, C)

	IF	ACTION	RETURN
1.	$C \equiv true$ or $H \equiv false$		$true$
2.	TYPELIST		
3.	$H \equiv A \vee B$		IMPLY$(nil,\ A \rightarrow C) \wedge (B \rightarrow C))$
4.	(AND SPLIT) $C \equiv A \wedge B$	put $\theta :=$ IMPLY(H, A)	
4.1	$\theta \equiv nil$		nil
4.2	$\theta \not\equiv nil$	put $\lambda :=$ IMPLY$(H, B\theta)$	
4.3	$\lambda \equiv nil$		nil
4.4	$\lambda \not\equiv nil$		$\theta o \lambda$
5	(REDUCE)	put $H :=$ REDUCE(H) put $C :=$ REDUCE(C)	
5.1	$C \equiv true$ or $H \equiv false$	goto 1	
5.2	$H \equiv A \vee B$	goto 3	
5.3	$C \equiv A \wedge B$	goto 4	
5.4	else	goto 6	
6	$C \equiv A \vee B$		HOA(H, C)
7	(PROMOTE) $C \equiv A \rightarrow B$		IMPLY$(H \wedge A, B)$
7.1	forward chaining		
7.2	PEEK forward chaining		
8	$C \equiv A \leftrightarrow B$		IMPLY$(H,\ (A \rightarrow B) \wedge (B \rightarrow A))$
9	$C \equiv (A = B)$	put $\theta :=$ UNIFY(A, B)	
9.1	$\theta \not\equiv nil$		θ
9.2	$\theta \equiv nil$	goto 10	
10	$C \equiv \neg A$		IMPLY$(H \wedge A, nil)$
11	INEQUALITY		
12	(call HOA)	put $\theta :=$ HOA(H, C)	
12.1	$\theta \not\equiv nil$		θ
12.2	(PEEK) $\theta \equiv nil$	put PEEK light ON	
12.3	$\theta \not\equiv nil$		θ
12.4	$\theta \equiv nil$	goto 13	
13	(DEFINE)	put $C' :=$ DEFINE(C) $H' :=$ DEFINE(H)	
13.1	H' or $C' \not\equiv nil$		IMPLY$(H',\ C')$
14	else		nil

TAB. 3 – HOA(B, C)

	IF	ACTION	RETURN
1	time limit exceeded		nil
2	(MATCH)	put $\theta :=$ UNIFY(B, C)	
2.1	$\theta \not\equiv nil$		θ
2.2	PEEK		HOA(B, C)
3	PAIRS		
4	(OR SPLIT) $C \equiv A \vee D$	put $C' :=$ AND-OUT(C)	
4.1	$C' \not\equiv C$		IMPLY(H, C')
4.2	$C' \equiv C$	put $\theta :=$ HOA$(B \wedge \neg D, A)$	
4.3	$\theta \not\equiv nil$		θ
4.4	$\theta \equiv nil$		HOA$(B \wedge \neg A, D)$
5			
5.1	$C \equiv A \rightarrow D$		IMPLY(B, C)
5.2	$C \equiv A \wedge D$		IMPLY(B, C)
6	$B \equiv A \wedge D$	put $\theta :=$ HOA(A, C)	
6.1	$\theta \not\equiv nil$		θ
6.2	$\theta \equiv nil$		HOA(D, C)
7	(back chaining) $B \equiv (A \rightarrow D)$	put $\theta :=$ ANDS(D,C)	
7.1	$\theta \equiv nil$	goto 7E	
7.2	$\theta \not\equiv nil$	put $\lambda :=$ IMPLY$(H, A\theta)$	
7.3	$\lambda \equiv nil$	goto 8	
7.4	$\lambda \not\equiv nil$		$\theta o \lambda$
7E	$B \equiv (A \rightarrow a = b)$	put $\theta :=$ HOA$(a = b, C)$	
7E.1	$\theta \equiv nil$		nil
7E.2	$\theta \not\equiv nil$	put $\lambda :=$ IMPLY$(H, A\theta)$	
7E.3	$\lambda \equiv nil$	goto 8	
7E.4	$\lambda \not\equiv nil$		$\theta o \lambda$
8	$B \equiv A \leftrightarrow D$		HOA$((A \rightarrow D) \wedge (D \rightarrow A), C)$
9	$B \equiv (a = b)$	put Z $:=$ MINUS-ON(a, b)	
9.1	$Z \equiv 0$		nil
9.2	Z is a number		$true$
9.3	Z is not a number	put $a' :=$ CHOOSE(a, b), $b' :=$ OTHER(a, b) put $H' := H[a'\|b']$, $C' := C[a'\|b']$	IMPLY(H', C')
10	$B \equiv A \vee D$		IMPLY(B, C)
11	$B \equiv \neg A$		IMPLY$(H, A \vee C)$
12	else		nil

TAB. 4 – Définitions

	Formula being defined	Defined form
1	$A = B$	$A \subset B \wedge B \subset A$
2	$A \subset B$	$\forall x(x \in A \rightarrow x \in B)$ Skolem form $(x_o \in A \rightarrow x_o \in B)$ in Conclusion $(x \in A \rightarrow x \in B)$ in Hypothesis
3	$A \cup B$	$\{x : x \in A \vee x \in B\}$
4	$A \cap B$	$\{x : x \in A \wedge x \in B\}$
5	$\bigcup_{t \in S} A(t)$	$\{x : \exists t(t \in S \wedge x \in A(t))\}$
6	$\bigcap_{t \in S} A(t)$	$\{x : \forall t(t \in S \rightarrow x \in A(t))\}$
7	$\mathcal{P}(A)$	$\{x : x \subset A\}$
8	range f	$\{y : \exists x(y = f(x))\}$
9	$Oc\ F$	$Open\ F \wedge Cover\ F$

Exemple2
Théorème à démontrer

$$A \subset B \wedge B \subset C \rightarrow A \subset C$$

transformé en

$$A_o \subset B_o \wedge B_o \subset C_o \rightarrow A_o \subset C_o$$

$nil \Rightarrow A_o \subset B_o \wedge B_o \subset C_o \rightarrow A_o \subset C_o$
par I7 $A_o \subset B_o \wedge B_o \subset C_o \Rightarrow A_o \subset C_o$
par I13 $A_o \subset B_o \wedge B_o \subset C_o \Rightarrow (t_o \in A_o \rightarrow t_o \in C_o)$
par I7 $A_o \subset B_o \wedge B_o \subset C_o \wedge t_o \in A_o \Rightarrow t_o \in C_o$
par I7.2 $A_o \subset B_o \wedge B_o \subset C_o \wedge t_o \in A_o \wedge t_o \in B_o \wedge t_o \in C_o \Rightarrow t_o \in C_o$
(PEEK forward chaining, 2 fois)
par H6 et H2 renvoie *true*

7 Applications

7.1 Théorie des ensembles

Le programme décrit dans [5] était dédié à la théorie des ensembles.
On a vu dans dans la section 5.6 deux exemples de théorèmes et leurs démonstrations. On trouvera également dans [5] les démonstrations détaillées de deux autres théorèmes

$$\omega \subset On$$

où On est la classe des ordinaux,
et

$$\omega = \omega \cap \mathcal{P}(\omega)$$

L'étude des ordinaux fait partie de la théorie axiomatique des ensembles, dans laquelle on introduit la classe de tous les ensembles, souvent appelé univers $\mathcal{U}$, et qui doit apparaitre dans certaines des réécritures de REDUCE, et par définition $0 = \Phi$.

$0 \in \mathcal{U}$ devient *true*
$\mathcal{U} \subset C$ " $\mathcal{U} = C$
$s \in \mathcal{P}(A)$ " $S \subset A \wedge S \in \mathcal{U}$
$s \in \{A\}$ " $(S = A) \wedge S \in \mathcal{U}$
$s \in \sigma F$ " $\exists y\ (y \in F \wedge x \in y) \wedge S \in \mathcal{U}$

J'ai omis ces références, par souci de simplification et lisibilité, dans la section 5, ce qui est correct

en théorie naïve des ensembles, mais elles apparaissent bien dans [5] [7].

D'autres exemples sont données dans [10].

7.2 Théorèmes sur les limites

Bledsoe a utilisé le programme décrit dans la section 6.1 pour démontrer des théorèmes sur les limites [6]. Pour cela il a introduit des types et leur manipulation, des subroutines de résolution d'égalités et d'inégalités, des méthodes de simplification algébrique, et surtout une heuristique spéciale spécifique pour les limites.

Types et inégalités

Le type A est attribué à x si l'on sait que $x \in A$.
$< a, b >$ désigne l'intervalle ouvert entre a et b $(]a, b[)$. Les types utilisés sont les types *intervalle*, incluant $< -\infty, \infty >$, $< 0, \infty >$, et $< -\infty, 0 >$.

Pour montrer

$$0 < x \rightarrow Q(x)$$

on attribue le type $< 0, \infty >$ à x et on essaie de montrer $Q(x)$.

Par exemple, pour montrer

$$0 < b \rightarrow \exists x(0 < x \wedge x < b) \qquad (1)$$

le programme décrit précédemment essaierait d'abord de démontrer

$$0 < b \rightarrow 0 < x$$

qui donnerait la substitution $[b|x]$, puis essaierait de démontrer

$$0 < b \rightarrow b < b$$

ce qui est impossible.

Pour démontrer (1) il faudrait rajouter des axiomes. A la place, Bledsoe utilise les types.
Le programme attribue le type $< 0, \infty >$ à b et il démontre d'abord

$$0 < x$$

en attribuant le type $< 0, \infty >$ à x puis il démontre

$$x < b$$

en attribuant à x le type $< 0, b >$, qui est l'intersection, non vide, des types $< 0, \infty >$ et $< -\infty, b >$.
Remarquer qu'une variable, à qui un type a été attribué, reste une variable, et peut donc voir son type modifié (intervalle diminué).

Voici quelques exemples de résolution d'inégalités.

inégalité à résoudre	types	nouveau type de x
$x < 1$		$< -\infty, 1 >$
$x < 1$	x de type $< 0, \infty >$	$< 0, 1 >$
$0 < 1$		toujours vrai
$x \cdot a + c < -x + d$	a de type $< 0, \infty >$	$< -\infty, (\frac{d}{1+a} - \frac{c}{1+a} >$
$x < D_1$	x de type $< 0, D_2 >$ D_1 et D_2 de type $< 0, \infty >$	$< 0, D_1 > \cap < 0, D_2 >$
$\frac{a}{x} < b$	x, a et b de type $< 0, \infty >$	$< \frac{a}{b}, \infty >$

Heuristique limite

C'est un heuristique spéciale qui permet de découper les ϵ en plusieurs morceaux.
Si on a (parmi d'autres) l'hypothèse

$$|A| < \epsilon'$$

[7] Et $\mathcal{U}$ est utilisé pour *true* et 0 pour *false*.

et la conclusion à démontrer

$$|B| < \epsilon$$

essayer d'abord de trouver une substitution σ telle que

$$(B = K \cdot A + L)_\sigma$$

et essayer de démontrer les trois nouvelles conclusions

$$\begin{array}{ll} (|K| < \mu)_\sigma & \text{pour un certain } \mu \\ (|A| < \epsilon/2\mu)_\sigma & \\ (|L| < \epsilon/2)_\sigma & \end{array}$$

On aura alors

$$|B|_\sigma = |K \cdot A + L|_\sigma \leq (|K| \cdot |A| + |L|)_\sigma < \mu \cdot \epsilon/2\mu + \epsilon/2 = \epsilon$$

Par exemple, pour montrer que la limite du produit de deux fonctions de variables réelles est le produit de leurs limites, on doit montrer [8]

$$|f(x) \cdot g(x) - L_1 \cdot L_2| < \epsilon$$

avec l'hypothèse

$$|f(x') - L_1| < \epsilon'$$

on trouve, avec $\sigma = [x|x']$, que

$$f(x) \cdot g(x) - L_1 \cdot L_2 = g(x) \cdot (f(x) - L_1) + L_1 \cdot (g(x) - L_2)$$

soit $K = g(x)$, $A = f(x) - L_1$, $L = L_1 \cdot (g(x) - L_2)$.
On a donc trois sous-buts à démontrer

$$\begin{array}{ll} |g(x)| < \mu & \text{pour un certain } \mu \\ |f(x) - L_1| < \epsilon/2\mu & \\ |L_1 \cdot (g(x) - L_2) < \epsilon/2 & \end{array}$$

qui le seront facilement en utilisant l'hypothèse

$$|g(x") - L_2| < \epsilon"$$

Cette heuristique, spécifique pour la recherche de limite, est un cas particulier d'un principe intéressant, plus général :

Pour montrer C à partir de plusieurs hypothèses dont H, *forcer* H à contribuer autant qu'elle le peut et laisser le *reste* à démontrer avec les *autres* hypothèses.

L'intérêt est double :
- H n'est plus nécessaire, donc cela réduit le nombre d'hypothèses à considérer pour la suite ;
- il est implicite dans la notion de *forcer* que certaines hypothèses sont utilisées pour faire quelque chose d'opératoire, calculer quelque chose plutôt que faire des inférences au hasard.

Ceci est fait par une fonction EXTRACT qui recoit deux expressions A et B et renvoie K, L et σ. Elle fonctionne de la façon suivante :

Supposons qu'il y a une substitution σ et une expression x telles que A_σ et B_σ sont des polynômes en x et A est linéaire en x. Alors il y a des expressions a, c, b et d telles que x n'apparait pas dans c, a ou d (ne peut apparaitre que dans b) et A_σ et B_σ peuvent être réexprimées comme

$$\begin{array}{l} A_\sigma = a \cdot x + c \\ B_\sigma = b \cdot x + d \end{array}$$

EXTRACT(A, B) retourne alors les valeurs

$$\left| \begin{array}{l} K = \frac{b}{a} \\ L = d - \frac{b \cdot c}{a} \\ \sigma \end{array} \right.$$

On trouvera ci-après (Exemple2) un aperçu de la démonstration complète du théorème sur la limite d'un produit de fonctions, après une démonstration plus détaillée d'un théorème plus simple (Exemple1).

[8] Un aperçu de la démonstration complète est donné plus loin (Exemple2)

Exemple1

Théorème à démontrer $\lim_{x \to a} x^2 = a^2$

soit

$$\forall \epsilon (0 < \epsilon \rightarrow \exists \delta (0 < \delta \wedge \forall x (x \in \mathbb{R} \wedge x \neq a \wedge |x - a| < \delta \rightarrow |x \cdot x - a \cdot a| < \epsilon)))$$

Elimination des quantificateurs et introduction de ϵ_o et $x(\delta)$ respectivement constante et fonction de Skolem.

$$0 < \epsilon_o \rightarrow (0 < \delta \wedge (x(\delta) \in \mathbb{R} \wedge x(\delta) \neq a \wedge |x(\delta) - a| < \delta \rightarrow |x(\delta) \cdot x(\delta) - a \cdot a| < \epsilon_o))$$

Pour plus de simplicité et de lisibilité, on écrira ϵ et x à la place de ϵ_o et $x(\delta)$, mais le programme garde et travaille avec ces notations de Skolem.
Soit $0 < \epsilon \rightarrow (0 < \delta \wedge (x \in \mathbb{R} \wedge x \neq a \wedge |x - a| < \delta \rightarrow |x \cdot x - a \cdot a| < \epsilon))$
Les inégalités sont supprimées et remplacées par des types. ϵ puis δ sont de type $< 0, \infty >$, x de type $< -\infty, \infty >$.

L'heuristique limite est utilisée pour démontrer $|x \cdot x - a \cdot a|$ à partir de $|x - a| < \delta$.
Avec $A = x - a$ et $B = x \cdot x - a \cdot a$, EXTRACT$(A, B)$ donne
$K = x + a$, $L = 0$, $\sigma = true$ (substitution vide)
soit $x \cdot x - a \cdot a = (x + a) \cdot (x - a) + 0$
et on doit alors montrer les deux sous-buts suivants :
(1) $|x + a| < \mu$ pour un certain μ
(2) $|x - a| < \epsilon / 2\mu$
le troisième étant trivial.

L'heuristique limite est de nouveau utilisée pour montrer (1) avec $A = x - a$ et $B = x + a$
EXTRACT(A, B) donne
$K = 1$, $L = 2 \cdot a$, $\sigma = true$ (substitution vide)
et on doit alors montrer
(11) $1 < \mu'$ pour un certain μ'
(12) $|x - a| < \mu / 2\mu'$
(13) $|2 \cdot a| < \mu / 2$
Avec l'hypothèse $|x - a| < \delta$ le programme trouve que δ doit être de type $< 0, \mu / 2\mu' >$ et μ de type $< |4.a|, \infty >$

Pour (2) δ doit être de type $< -\infty,\ \epsilon / 2\mu >$, donc finalement de type
$< 0,\ \mu / 2\mu' > \cap < -\infty, \epsilon / 2\mu >$ c'est-à-dire compris entre 0 et $\min(\mu / 2\mu', \epsilon / 2\mu)$

Le programme a donc finalement "choisi" μ, μ' et δ tels que

$$\begin{cases} \mu' > 1 \\ \delta < \mu / 2\mu' \\ \delta < \epsilon / 2\mu \\ \mu > |4a| \end{cases}$$

et on a bien
$|x \cdot x - a \cdot a| = |(x - a) \cdot (x - a + 2a)| \leq |x - a| \cdot (|x - a| + 2|a|) < (\epsilon / 2\mu) \cdot (\mu / 2\mu' + \mu / 2)$
$< (\epsilon / 2\mu) \cdot \mu = \epsilon / 2 < \epsilon$

Exemple2 - Limite d'un produit de fonctions

Théorème à démontrer

$$\lim(f, a, L_1) \wedge \lim(g, a, L_2) \rightarrow \lim(f \cdot g,\ a,\ L_1 \cdot L_2)$$

Notations :
$\lim(f, a, L)$ est une notation prédicative pour $L = \lim_{x \to a} f(x)$

$\mathbb{R}$ est l'ensemble des réels (intervalle $< -\infty, \infty >$)
$\mathcal{F}(\mathbb{R}, \mathbb{R})$ est l'ensembles des applications de $\mathbb{R}$ dans $\mathbb{R}$.

Remplacement du prédicat lim par sa définition
$a \in \mathbb{R} \wedge L_1 \in \mathbb{R} \wedge f \in \mathcal{F}(\mathbb{R}, \mathbb{R})$
$\wedge \forall \epsilon_1 (0 < \epsilon_1 \rightarrow \exists \delta_1 (0 < \delta_1 \wedge \forall x_1 (x_1 \in \mathbb{R} \wedge x_1 \neq a \wedge |x_1 - a| < \delta_1 \rightarrow |f(x_1) - L_1| < \epsilon_1))$
$\wedge\ a \in \mathbb{R} \wedge L_2 \in \mathbb{R} \wedge g \in \mathcal{F}(\mathbb{R}, \mathbb{R})$
$\wedge \forall \epsilon_2 (0 < \epsilon_2 \rightarrow \exists \delta_2 (0 < \delta_2 \wedge \forall x_2 (x_2 \in \mathbb{R} \wedge x_2 \neq a \wedge |x_2 - a| < \delta_2 \rightarrow |g(x_2) - L_2| < \epsilon_2))$
$\rightarrow$
$a \in \mathbb{R} \wedge L_1 \cdot L_2 \in \mathbb{R} \wedge f \cdot g \in \mathcal{F}(\mathbb{R}, \mathbb{R})$
$\wedge \forall \epsilon (0 < \epsilon \rightarrow \exists \delta (0 < \delta \wedge \forall x (x \in \mathbb{R} \wedge x \neq a \wedge |x - a| < \delta \rightarrow |(f \cdot g)(x) - L_1 \cdot L_2| < \epsilon))$

Les trois premières parties de la conclusion sont démontrées en utilisant les hypothèses et les théorèmes connus sur $\mathbb{R}$ et $\mathcal{F}(\mathbb{R}, \mathbb{R})$
Puis remplacement de $(f \cdot g)(x)$ par sa définition $f(x) \cdot g(x)$, élimination des quantificateurs et introduction de constante et fonctions de Skolem $\delta_1(\epsilon_1)$, $\delta_2(\epsilon_2)$, ϵ_o et $x(\delta)$ que l'on écrira dans la suite δ_1, δ_2, ϵ, x pour plus de simplicité et de lisibilité mais, comme précédemment, le programme garde et travaille avec ces notations de Skolem.

$a \in \mathbb{R} \wedge L_1 \in \mathbb{R} \wedge f \in \mathcal{F}(\mathbb{R}, \mathbb{R})$
$\wedge (0 < \epsilon_1 \rightarrow (0 < \delta_1 \wedge (x_1 \in \mathbb{R} \wedge x_1 \neq a \wedge |x_1 - a| < \delta_1 \rightarrow |f(x_1) - L_1| < \epsilon_1))$
$\wedge\ a \in \mathbb{R} \wedge L_2 \in \mathbb{R} \wedge g \in \mathcal{F}(\mathbb{R}, \mathbb{R})$
$\wedge (0 < \epsilon_2 \rightarrow (0 < \delta_2 \wedge (x_2 \in \mathbb{R} \wedge x_2 \neq a \wedge |x_2 - a| < \delta_2 \rightarrow |g(x_2) - L_2| < \epsilon_2))$
$\rightarrow$
$(0 < \epsilon \rightarrow (0 < \delta \wedge (x \in \mathbb{R} \wedge x \neq a \wedge |x - a| < \delta \rightarrow |f(x) \cdot g(x) - L_1 \cdot L_2| < \epsilon))$

Les inégalités sont supprimées et remplacées par des types. E puis D sont de type $< 0, \infty >$, x de type $< -\infty, \infty >$, etc ....

L'heuristique limite est appliquée trois fois.
Avec $A = f(x_1) - L_1$, $B = f(x) \cdot g(x) - L_1 \cdot L_2$, EXTRACT$(A, B)$ donne
$K = g(x)$, $L = g(x) \cdot L_1 - L_1 \cdot L_2)$, $\sigma = [x|x_1]$
soit
$f(x) \cdot g(x) - L_1 \cdot L_2 = g(x) \cdot (f(x1) - L_1) + g(x) \cdot L_1 - L_1 \cdot L_2)$
et on doit alors montrer les trois sous-buts suivants :
(1) $|g(x)| < \mu$
(2) $|f(x) - L_1| < \epsilon/2\mu$
(3) $|g(x) \cdot L_1 - L_1 \cdot L_2| < \epsilon/2$

Pour démontrer(1), l'heuristique limite est de nouveau appliquée avec $A = g(x_2) - L_2$, $B = g(x)$, EXTRACT(A, B) donne
$K = 1$, $L = L_2$, $\sigma = [x|x_2]$
et on doit alors montrer
(11) $1 < \mu'$
(12) $|g(x) - L_2| < \mu/2\mu'$
(13) $|L_2| < \mu/2$
Avec l'hypothèse $|g(x_2) - L_2| < \epsilon_2$ le progamme trouve que ϵ_2 doit être de type $< -\infty, \mu/2\mu' >$ et μ doit être de type $< |2L_2|, \infty >$

Le sous-but (2) est facilement établi, à partir de l'hypothèse $|f(x_1) - L_1| < \epsilon_1$ en attribuant le type $< -\infty, \epsilon/2\mu >$ à ϵ_1.

Pour démontrer(3), l'heuristique limite est de nouveau appliquée avec
$A = g(x) - L_2$, $B = g(x) \cdot L_1 - L_1 \cdot L_2$, EXTRACT$(A, B)$ donne

$K = L_1, L = 0, \sigma = true$ (substitution vide)
et on doit alors montrer
(31) $|L_1| < \mu'$
(32) $|g(x) - L_2| < (\epsilon/2)/2\mu'$
(33) $0 < \epsilon/4$
Le programme trouve que μ' doit être de type $< |L_1|, \infty >$ et, avec l'hypothèse $|g(x_2) - L_2| < \epsilon_2$, que ϵ_2 doit être de type $< -\infty, \epsilon/4\mu' >$. Comme ϵ_2 était déjà de type $< 0, \mu/2\mu' >$, il doit maintenant être de type $< 0, \mu/2\mu' > \cap < -\infty, \epsilon/4\mu' >$

Le programme a donc finalement "choisi" μ, μ', ϵ_1 et ϵ_2 tels que

$$\begin{cases} \mu > 2|L_2| \\ \mu' > |L_1| \\ \epsilon_1 < \epsilon/2\mu > \\ \epsilon_2 < \min(\mu/2\mu', \epsilon/4\mu' > \end{cases}$$

pour établir, à partir de
$|f(x_1) - L_1| < \epsilon_1$
$|g(x_2) - L_2| < \epsilon_2$
que
$|f(x) \cdot g(x) - L_1 \cdot L_2| = |g(x) \cdot (f(x) - L_1) + g(x) \cdot L_1 - L_1 \cdot L_2|$
$\leq |g(x) \cdot (f(x) - L_1)| + |L_1 \cdot (g(x) - L_2)$
$= |g(x)| \cdot |f(x) - L_1| + |L_1| \cdot |g(x) - L_2|$
$< \mu \cdot \epsilon/2\mu + \mu' \cdot \min(\mu/2\mu', \epsilon/4\mu')$
$< \epsilon$

Autres exemples

On trouvera dans [6] deux autres exemples de preuves de théorèmes sur les limites :

• continuité de la composée de deux fonctions continues
$\lim(g, a, g(a)) \wedge lim(f, g(a), f(g(a))) \rightarrow lim(f_o g, a, f(g(a)))$

• continuité des fonctions dérivables
si $\lim_{h \to 0} \frac{f(a+h)-f(a)}{h} = F'$ alors $\lim_{x \to a} f(x) = f(a)$

et l'analyse d'un échec :

• le programme ne peut pas montrer que la limite de l'inverse d'une fonction est égale à l'inverse de la limite de la fonction.

7.3 Un système interactif en topologie

Le système décrit par Bledsoe dans [7] est une version améliorée et interactive des précédents programmes. Il a été expérimenté en topologie.

Le programme vérifie un certain nombre de propriétés, et peut demander de l'aide à l'utilisateur s'il en a besoin (par exemple s'il n'a pas réussi à démontrer un sous-théorème dans le temps limite qui lui est donné).

Le but n'est pas que l'être humain aide la machine mais qu'il puisse tester certaines idées et, si cela aboutit, que la machine se charge du travail fastidieux.

L'utilisateur peut demander à la machine
- des détails (DETAIL) ;
- d'instancier une variable (PUT) ;
- de remplacer les occurrences d'un prédicat par sa définition ;
- d'ajouter un théorème connu comme nouvelle hypothèse du théorème courant ;

- de prouver d'abord un lemme, puis de l'utiliser ;
- d'augmenter le temps limite pour le sous-but courant ;
- d'admettre le sous-but courant et de continuer ;
- de revenir en arrière au précédent sous-but ;
- d'ajouter des règles à REDUCE ;
- d'ajouter des définitions ;
- ... .

Quand la machine s'arrête car elle a échoué (FAILED) à démontrer le (sous)-théorème courant (au début le théorème initial) dans le temps limite imposé l'utilisateur peut taper
- soit DETAIL, et la machine commence ou continue la preuve jusqu'au premier sous-théorème suivant qu'elle ne peut pas démontrer ;
- soit une autre commande pour aider la machine.

A chaque fois que la machine a démontré un sous-théorème (PROVED), elle donne la main à l'utilisateur qui doit taper
- soit GO pour qu'elle continue ;
- soit une autre commande.

Exemple de dialogue

Théorème à démontrer :
Un ensemble est ouvert si et seulement si il contient un voisinage [9] de chacun de ses points.

U désigne l'utilisateur, M la machine, C est un commentaire.

U	ouvert$(A) \leftrightarrow \forall x(x \in A \rightarrow \exists V(\text{voisinage}(V,x) \wedge V \subset A)$
M (1)	ouvert$(A) \leftrightarrow \forall x(x \in A \rightarrow \exists V(\text{voisinage}(V,x) \wedge V \subset A)$
	FAILED
U	DETAIL
M (11)	ouvert$(A_o) \rightarrow (x_o \in A_o \rightarrow (\text{voisinage}(V,x_o) \wedge V \subset A_o)$
	FAILED
C	premier sous-théorème skolémisé, V est une variable à instancier
U	DETAIL
M (111)	ouvert$(A_o) \wedge x_o \in A_o \rightarrow \text{voisinage}(V,x_o)$
	FAILED
C	premier sous-théorème de (11)
U	DETAIL
M (111)	ouvert$(A_o) \wedge x_o \in A_o \rightarrow \text{ouvert}(A) \wedge A \subset V \wedge x_o \in A$
C	définition de voisinage et skolémisation, A est une variable à instancier
M (1111)	ouvert$(A_o) \wedge x_o \in A_o \rightarrow \text{ouvert}(A)$
	PROVED $[A_o\|A]$
U	GO
M (1112)	ouvert$(A_o) \wedge x_o \in A_o \rightarrow A_o \subset V)$
	ouvert$(A_o) \wedge x_o \in A_o \rightarrow (t_o \in A_o \rightarrow t_o \in V)$
	ouvert$(A_o) \wedge x_o \in A_o \wedge t_o \in A_o \rightarrow t_o \in V$
	PROVED $[A_o\|V]$
U	GO
M (1113)	ouvert$(A_o) \wedge x_o \in A_o \rightarrow x_o \in A_o$
	PROVED
U	GO

[9] Définition d'un voisinage V de x : $voisinage(V,x) \leftrightarrow \exists A(ouvert(A) \wedge A \subset V \wedge x \in A)$

M (112) ouvert$(A_o) \rightarrow (x_o \in A_o \rightarrow (A_o \subset A_o)$
PROVED
C par REDUCE, $A \subset A$ est réécrit *true*
M (12) $(x \in A_o \rightarrow$ voisinage$(V(x), x) \wedge V(x) \subset A_o) \rightarrow$ ouvert(A_o)
FAILED
C on cherche x (remarquer que la skolémisation est différente de celle de (11), voir section 6.1)
U DETAIL
M (12) $(x \in A_o \rightarrow$ voisinage$(V(x), x) \wedge V(x) \subset A_o)$
$\rightarrow (F \subset \mathcal{T} \wedge A_o = \sigma F) \vee$ (ouvert$(B) \wedge$ ouvert$(D) \wedge A_o = B \cap D)$
C définition de ouvert et skolémisation
on cherche, pour la topologie $\mathcal{T}$, soit une union d'ouverts σF,
soit une intersection de deux ouverts B et D
M(121) $(...) \rightarrow (F \subset \mathcal{T} \wedge A_o = \sigma F)$
FAILED
U DETAIL
(1211) $(...) \rightarrow (F \subset \mathcal{T}$
FAILED
C Ici l'utilisateur doit aider la machine en lui donnant comme valeur pour F la famille
des sous-ensembles ouverts de A_o
U PUT $F = \mathcal{T} \cap \mathcal{P}(A_o)$
M (1211) $(...) \rightarrow (\mathcal{T} \cap \mathcal{P}(A_o) \subset \mathcal{T}$
PROVED
U GO
M (1212) $(...) \rightarrow A_o = \sigma(\mathcal{T} \cap \mathcal{P}(A_o))$
FAILED
U DETAIL
M (1212) $(...) \rightarrow A_o \subset \sigma(\mathcal{T} \cap \mathcal{P}(A_o)) \wedge \sigma(\mathcal{T} \cap \mathcal{P}(A_o)) \subset A_o$
(12121) $(...) \rightarrow A_o \subset \sigma(\mathcal{T} \cap \mathcal{P}(A_o))$
$(...) \rightarrow (t_o \in A_o \rightarrow t_o \in \sigma(\mathcal{T} \cap \mathcal{P}(A_o))$
C définition de $\subset$
M $(...) \wedge t_o \in A_o \rightarrow t_o \in \sigma(\mathcal{T} \cap \mathcal{P}(A_o))$
$(...) \wedge$ voisinage$(V(t_o), t_o) \wedge V(t_o) \subset A_o \wedge t_o \in A_o \rightarrow t_o \in \sigma(\mathcal{T} \cap \mathcal{P}(A_o))$
C forward chaining (voir section 6.2) de $t_o \in A_o$) avec l'hypothèse
$(...) =$(x \in A_o \rightarrow$ voisinage$(V(x), x) \wedge V(x) \subset A_o)$
M $(...) \wedge ... \rightarrow$ ouvert$(A) \wedge A \subset A_o \wedge t_o \in A$
C définition de σ et $\subset$
on cherche A tel que $t_0 \in A$ et $A \in \mathcal{T} \cap \mathcal{P}(A_o)$ c'est-à-dire A ouvert $\subset A_0$
M (121211) $... \rightarrow$ ouvert(A)
$... \wedge$ ouvert$(A_1) \wedge A_1 \subset V(t_o) \wedge t_o \in A_1) \rightarrow$ ouvert(A)
C PEEK forward chaining avec l'hypothèse voisinage$(V(t_0), t_0)$ et la définition de voisinage
M PROVED $[A_1|A]$
U GO
M (121212) $... \wedge A_1 \subset V(t_o) \wedge ... \wedge V(t_o) \subset A_o) \rightarrow A_1 \subset A_o$

Ce sous-théorème (121212) est facilement démontré en remplaçant l'inclusion de la conclusion par sa définition et par deux "forward chaining" (Exemple2 de la section 6.2).
La machine peut aussi utiliser directement la transitivité de l'inclusion $A \subset B \wedge B \subset C \rightarrow A \subset C$ qu'on a pu lui donner comme théorème connu sur les ensembles.
Sinon l'utilisateur peut aussi lui donner et lui dire d'appliquer cette propriété.
L'utilisateur peut aussi dire à la machine d'appliquer la transitivité de l'inclusion.

Il reste à démontrer le sous-théorème (12122) de conclusion $\sigma(\mathcal{T} \cap \mathcal{P}(A_o)) \subset A_o$ facilement démontré.

Deux autres exemples de théorèmes de topologie sont donnés dans [7].

7.4 Autres applications et autres méthodes naturelles

D'autres systèmes ont été développés par Bledsoe et son équipe en topologie [1, 2, 4], analyse [4], analyse non standard [3] et arithmétique de Presburger (l'arithmétique de Peano sans la multiplication) [8].

Ces réalisations utilisent des enrichisssements des programmes précédemment décrits et aussi d'autres méthodes, elles aussi naturelles, en particulier divers sortes de représentations graphiques.

Ballantyne et Bennett [1] ont utilisé des méthodes graphiques pour démontrer automatiquement des théorèmes de topologie. Les ensembles sont représentés par les noeuds d'un graphe. Les relations entre ensembles, en particulier les relations d'inclusion sont représentées par des arcs du graphe. Le programme raisonne sur le graphe pour démontrer des propriétés sur les unions, intersections, complémentaires d'adhérences, intérieurs et frontières d'ensembles.

Cette méthode a également été utilisée par Ballantyne pour la recherche de contrexemples [2, 4], et par Ballantyne et Bledsoe [4] pour accélérer la découverte de preuve en analyse et pour aider le mathématicien dans sa recherche.

Ballantyne et Bledsoe [3] ont utilisé des techniques d'analyse non standard pour démontrer automatiquement des théorèmes d'analyse, après avoir (automatiquement) traduit leur énoncé standard en énoncé non standard. Ils utilisent les types *intervalle* et ont de plus défini une méthode nommée "sup-inf" qui, étant donné un ensemble d'inégalités entre expressions, encadrent les variables dans un intervalle <inf, sup>.

Enfin, dans [9], Bledsoe fait en 1977 une sorte d'inventaire des démonstrateurs utilisant des techniques autres que la *Résolution*, avec de nombreux exemples. Sans prétendre que ces techniques soient les seules possibles pour progresser, il termine par ce souhait :

> "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. This will allow us to eliminate some of our "good" ideas.
>
> It is *not* the time to give up on automatic theorem proving. How can that be advisable at a time when so little has been done to develop and apply the ideas we already have? For example, why doesn't someone else use analogy in automatic proofs?
>
> 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 develop truly powerful provers, and not just theories".

Bibliographie

[1] M. Ballantyne and W. Bennet. Graphing methods for topological proofs. Memo ATP 7, University of Texas at Austin, math dept, 1973.

[2] M. Ballantyne. Computer generation of counterexamples in topology. Memo ATP 24, University of Texas at Austin, math dept, 1975.

[3] A.M. Ballantyne and W.W. Bledsoe. Automatic proofs of theorems in analysis using nonstandard techniques. *J. ACM*, 24, 353–374, 1977.

[4] A.M. Ballantyne and W.W. Bledsoe. On generating and using examples in proof discovery. *Machine Intelligence*, 10, 3–39, 1982.

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

[6] W.W. Bledsoe, R.S. Boyer, and W.H. Henneman. Computer proofs of limit theorems. *Artificial Intelligence Journal*, 3, 27–60, 1972.

[7] W.W. Bledsoe and P. Bruell. A man-machine theorem-proving system. *Artificial Intelligence Journal*, 5, 51–72, 1974.

[8] W.W. Bledsoe. A new method for proving certain presburger formulas. *Proc. 4th IJCAI*, pages 15–21, 1975.

[9] W.W. Bledsoe. Non-resolution theorem proving. *Artificial Intelligence Journal*, 9, 1–35, 1977.

[10] W.W. Bledsoe and M. Tyson. The UT interactive prover. Memo ATP 17A, University of Texas at Austin, math dept, 1978.

[11] H. Gelertner. Realization of a geometry-theorem proving machine. *Proc. International Conf on Information Processing*, 1959 ; In : Feigenbaum and Feldman, editors, *Computers and thought*, 134–152, McGraw-Hill, 1963.

[12] A Newel, J.C. Shaw, and H.A. Simon. Empirical explorations with the logic theory machine : a case study in heuristics. *Proc. Western Joint Computer Conf.*, 1957 ; In : Feigenbaum and Feldman, editors, *Computers and thought*, 109–133, McGraw-Hill, 1963.

[13] A Newel, J.C. Shaw, and H.A. Simon. Chess playing programs and the problem of complexity. *IBM Journal of Research and Development*, 1958 ; In : dans Feigenbaum and Feldman, editors, *Computers and thought*, 39–70, McGraw-Hill, 1963.

[14] A Newel and H.A. Simon. GPS, a program that simulates human thought. *Proc. Conf. on Learning Automata*, 1961 ; In : Feigenbaum and J. Feldman, editors, *Computers and thought*, 279–293, McGraw-Hill, 1963.

[15] D. Pastre. Logique et principe de résolution. cours, Université René Descartes - Paris 5, http ://www.math-info.univ-paris5.fr/∼pastre/logique, 2000.

[16] J.A. Robinson. A machine oriented logic based on the resolution principle. *J. ACM*, 12, 23–41, 1965.

[17] H. Wang. Towards mechanical mathematics. *IBM J.Res.Develop*, 4, 2–22, 1960.