ACTA UNIVERSITATIS LODZIENSIS
FOLIA PHILOSOPHICA 9, 1993

Patrice Bailhache

A LA RECHERCHE D'UN (IMPOSSIBLE) DÉMONSTRATEUR INTELLIGENT

Le développement de l'informatique a permis l'essor, ces dernières années, de nombreux logiciels d'aide à l'enseignement de la logique, sinon en France, du moins dans les pays anglo-saxons. La plupart de ces logiciels, dès qu'il s'agit du calcul des prédicats (du premier ordre, avec ou sans identité), fonctionnent en simples *vérificateurs*, c'est-à-dire que le rôle de la machine se trouve réduit à contrôler que les déductions proposées par l'utilisateur sont correctes. Qu'ils fonctionnent ainsi, et non pas comme des *démonstrateurs*, se comprend facilement: d'après le théorème d'indécidabilité, il ne peut exister d'algorithme général pour le calcul des prédicats, de sorte que l'écriture d'un programme est théoriquement impossible pour la démonstration.

Mais on sait tout de même que le problème de la décision est soluble pour certaines classes de formules. En outre, dans de nombreux cas plus ou moins simples, on constate qu'un logicien de force moyenne trouve aisément la «bonne déduction». Or ce qui est aisé informellement se révèle souvent difficile dès qu'on cherche à le formaliser, et plus encore dès qu'on veut le *programmer*.

Les travaux dans le domaine de la démonstration automatique sont nombreux. Avec l'apparition du langage *Prolog*, l'usage des clauses de Horn s'est étendu. Cependant, cela n'est pas satisfaisant pour un logiciel d'apprentissage de la logique, que l'on voudrait voir fonctionner en véritable *démonstrateur général intelligent*, qui ne se limite pas à une structure particulière de formules comme celle des clauses de Horn. C'est pourquoi, notamment, j'ai cherché à construire un tel démonstrateur.

1. «INTELLIGENCE» ET «NATURE»

Quelques examples simples – je vais en donner bientôt – font rapidement comprendre ce qu'il faut entendre par l'*intelligence* dans une preuve logique, du moins pour une première approche non détaillée (la définition précise du concept est certainement beaucoup plus délicate, voire même impossible, précisément d'après ce que nous a enseigné Gödel).

Prenons ainsi la formule:

$\exists x \forall y\ Fxy \supset \forall y \exists x\ Fxy$.

Sa preuve dans le système de *déduction naturelle* de Gentzen s'administre sans difficulté. Après avoir appliqué la règle d'analyse du foncteur *implication*, on utilise les règles concernant les quantificateurs:

Déduction «intelligente»

1. $\Rightarrow \exists x \forall y\ Fxy \supset \forall y \exists x\ Fxy$	$\Rightarrow \supset$
2. $\exists x \forall y\ Fxy \Rightarrow \forall y \exists x\ Fxy$	$\exists \Rightarrow x/a$ variable nouvelle
3. $\forall y\ Fay \Rightarrow \forall y \exists x\ Fxy$	$\Rightarrow \forall y/b$ variable nouvelle
4. $\forall y\ Fay \Rightarrow \exists x\ Fxb$	$\forall \Rightarrow y/b$ variable ancienne
5. $\forall y\ Fay,\ Fab \Rightarrow \exists x\ Fxb$	$\Rightarrow \exists x/a$ variable ancienne
6. $\forall y\ Fay,\ Fab \Rightarrow \exists x\ Fxb,\ Fab$	séquence valide

Bien entendu, dans cette déduction, le jeu du choix des variables libres qui apparaissent par l'application des règles d'analyse des quantificateurs est essentiel. Ainsi les règles $\Rightarrow \forall$ et $\exists \Rightarrow$ n'autorisent-elles que l'emploi de variables nouvelles, tandis que les deux autres permettent théoriquement celui de n'importe quelle variable[1], mais doivent en fait, pour des raisons d'élémentaire stratégie, être restreintes aux variables libres qui figurent déjà dans la déduction[2].

Mais au delà de ces restrictions, on constate que plusieurs choix *intelligents* ont été pratiqués dans cette preuve:

1. Les deux règles $\Rightarrow\forall$ et $\exists \Rightarrow$ ont été appliquées avant les deux autres $\Rightarrow\exists$ et $\forall\Rightarrow$, c'est-à-dire que les variables libres nouvelles ont été dégées avant qu'elles soient employées comme variables anciennes par les règles $\Rightarrow\exists$ et $\forall\Rightarrow$.

2. Le choix des variables anciennes, dans les deux derniers pas, n'a pas été quelconque, mais a été pratiqué de telle sorte que la formule *Fab* figure des deux côtés de la flèche de la séquence, pour que celle-ci achève la démonstration.

---

[1] On doit toutefois éviter les variables déjà quantifiées, au risque de commettre des «collisions» de quantificateurs, correspondant à des sophismes.

[2] La négligence de ce second impératif conduirait à des séquences non conclusives, sans rapport avec la nature de la formule. Par exemple, l'emploi des variables c et *d* dans les deux derniers pas de la preuve produirait la séquence: $\forall y\ Fay,\ Fac \Rightarrow \exists x\ Fxb,\ Fdb$.

Afin de mieux percevoir la portée de ces deux modes de la procédure, on peut comparer la déduction «intelligente» avec la déduction que j'appelerai «planifiée» ou «systématique», et qui consiste, en fait, en une analyse *en largeur d'abord* de l'arbre de déduction. Cette déduction procède par *étapes*, en chacune desquelles exactement une règle se trouve appliquée (si c'est possible) à chaque formule de la séquence du début de l'étape[3].

Déduction «planifiée»

ETAPE 1

chemin 1

1. $\Rightarrow \exists x \forall y\, Fxy \supset \forall y \exists x\, Fxy$ $\Rightarrow \supset$
2. $\exists x \forall y\, Fxy \Rightarrow \forall y \exists x\, Fxy$

ETAPE 2

chemin 1

2. $\exists x \forall y\, Fxy \Rightarrow \forall y \exists x\, Fxy$ $\exists \Rightarrow x/a$ variable nouvelle
3. $\forall y\, Fay \Rightarrow \forall y \exists x\, Fxy$ $\Rightarrow \forall\ y/b$ variable nouvelle
4. $\forall y\, Fay \Rightarrow \exists x\, Fxb$

ETAPE 3

chemin 1

4. $\forall y\, Fay \Rightarrow \exists x\, Fxb$ $\forall \Rightarrow y/a$ variable ancienne
5. $\forall y\, Fay,\ Faa \Rightarrow \exists x\, Fxb$ $\forall \Rightarrow y/b$ variable ancienne
6. $\forall y\, Fay,\ Fab,\ Faa \Rightarrow \exists x\, Fxb$ $\Rightarrow \exists\ x/a$ variable ancienne
7. $\forall y\, Fay,\ Fab,\ Faa \Rightarrow \exists x\, Fxb,\ Fab$ séquence valide

On constate un changement avec la déduction intelligente à la formule 5, dans laquelle la formule *Faa* est produite inutilement, mais régulièrement, car la planification exige notamment que soient tirées toutes les instances possibles, avec les variables libres disponibles, des formules universelles vraies (ou existentielles fausses). L'example choisi ici est du reste trop simple, car la *maladresse* provoquée par la planification reste très limitée. Une autre maladresse pourrait être que les règles $\Rightarrow \exists$ et $\forall \Rightarrow$ soient employées avant les règles $\Rightarrow \forall$ et $\exists \Rightarrow$, alors que l'inverse serait effectivement possible[4].

Notons qu'il est d'usage de qualifier de *naturelle* le style de déduction «à la Gentzen», illustré ici autant par la déduction intelligente que par la déduction

[3] Cette procédure planifiée est définie par S. C. Kleene, au chapitre 6 de sa *Logique mathématique* (A. Colin, Paris 1971, p. 289–310 de la traduction française).

[4] Il faut tenir compte du fait que des formules du genre $\exists x \forall y\, Fxy$ peuvent très bien figurer à la droite de la flèche des séquences, auquel cas la règle $\Rightarrow \exists$ se trouve nécessairement appliquée avant la règle $\Rightarrow \forall$ (du moins en ce qui concerne cette formule), c'est-à-dire qu'une variable libre nouvelle se trouve nécessairement dégagée après l'emploi de telles variables pour les *unifications* de formules. Ceci, on le sait, peut mener à des chemins infinis.

planifiée. Ce qualificatif provient du fait que, par contraste, les preuves axiomatiques qui utilisent la règle Modus Ponens (ou règle *de Détachement*) ont une structure entièrement indépendante de celle de la formule à prouver[5]. En quelque sorte, la combinaison de la règle de substitution et de la règle Modus Ponens annihile toute *mémoire* de la formule. Mais on découvre, pour ainsi dire, que la nature n'est nullement intelligente! Si effectivement la déduction naturelle fournit systématiquement, sans effort d'imagination, une preuve de n'importe quelle formule valide, en revanche elle ne donne, dès qu'il s'agit d'une expression tant soit peu complexe, qu'une *mauvaise* preuve, longue, remplie de pas intutiles. La déduction naturelle ne résout donc nullement le problème de la stratégie. Il faut se contenter de dire qu'elle le pose en certains termes, sans doute plus clairs qu'avec les autres styles de systèmes. L'ombre de Gödel se projette ici de toute sa grandeur: puisque le calcul des prédicats n'est pas décidable, bien que complet, il ne faut rien attendre de plus d'une déduction *naturelle* par essence incapable de tout pouvoir de décision.

2. A LA RECHERCHE DE L'(IMPOSSIBLE) INTELLIGENCE

Les deux critères énoncés ci-dessus pour la procédure intelligente doivent maintenant être développés.

1. En ce qui concerne la préséance de l'application des règles $\Rightarrow \forall$ et $\exists \Rightarrow$ sur les règles $\Rightarrow \exists$ et $\forall \Rightarrow$, seule une vision simplifiée des choses peut laisser penser que la situation est simple. En réalité, comme je l'ai mentionné en note, la structure des formules à traiter peut forcer à l'application dans l'ordre inverse, et il ne reste plus qu'à se soumettre au verdict des données. Mais même si l'on ne se trouve pas dans ce dernier cas, la situation peut être truffée d'ambiguïtés: il peut très bien y avoir concurrence entre plusieurs applications des règles du premier groupe et/ou concurrence entre plusieurs applications des règles du second groupe. On ne voit guère quelle (méta-)règle générale adopter pour trancher dans ces options.

En outre, si c'est le cas, c'est-à-dire si par exemple des formules du genre $\exists x \forall y\ Fxy$, $\exists x \exists y \forall z\ Gxyz$,... figurent à la droite de la flèche de la séquence, il faut adopter une ligne de conduite qui permette de dégager aussi vite que possible les variables libres nouvelles. Il est bien clair que cette ligne de conduite devre être de travailler sur de telles formules (sur leurs quantificateurs existentiels pour celles de droite) avant d'appliquer les mêmes règles à des formules «cul-de-sac» (comme $\exists x\ Fx$), qui ne conduisent pas à des expressions dégageant ensuite des variables nouvelles. Et dans cette voie, on pourra même

[5] Par exemple, dans les systèmes axiomatiques pour le calcul des propositions de Russell ou de Łukasiewicz.

établir une hiérarchie stratégique: traiter par ordre de priorité décroissante les formules de structure (à droite de la flèche);
$\exists x\, Fx$, $\exists x \forall y\, Fxy$, $\exists x \exists y \forall z\, Gxyz$, $\exists x \exists y \exists z \forall w\, Hxyzw$,...
Avouons cependant que, si d'une manière générale cette ligne de conduite paraît fondée, rien n'assure toutefois qu'elle se montrera parfaitement intelligente dans tous les cas d'espèce[6].

2. Le second critère pour la procédure intelligente est celui du choix des variables anciennes, autrement dit celui de l'*unification* des formules atomiques. Ici, un algorithme est connu: c'est celui de la détermination du *plus grand unificateur*. J'ai cherché, pour un logiciel d'enseignement de la déduction naturelle, à faire à peu près la même chose par une procédure simplifiée, mais moins systématique, où l'unification a lieu en deux temps. Dans un premier temps, l'unification est faite par comparaison des atomes de la formule à instantier avec ceux des autres formules de la séquence. Au cours de cette étape, les instantiations produisant des atomes sans aucune variable liée ont la priorité sur les autres[7]. Dans un second temps, le choix des variables anciennes est «affiné» par la comparaison, non plus de simples formules atomiques, mais de véritables sous-formules. Un exemple[8] fera comprendre la nécessité de cette seconde «passe».

Supposons que l'on veuille prouver l'incompatibilité logique des trois expressions:

1. $\exists x \forall y\, Fxy$
2. $\forall x \forall y \exists z\ (Fxy \supset ((\sim Fxz\ \&\ Fzy) \vee (Fyz\ \&\ Fzx)))$
3. $\forall x \exists y \forall z \sim (Fyz\ \&\ Fzx)$

Sans décrire le détail de la procédure de la déduction naturelle dans le style de Gentzen, nous pouvons abréger les écritures en tirant les instances suivantes:

4. $\forall y\, Fay$ (de 1 avec *a* variable nouvelle)
5. $\exists y \forall z \sim (Fyz\ \&\ Fza)$ (de 3 avec *a* variable ancienne)
6. $\forall z \sim (Fbz\ \&\ Fza)$, (de 5 avec *b* variable nouvelle)
7. $\forall y \exists z\, (Fay \supset ((\sim Faz\ \&\ Fzy) \vee (Fyz\ \&\ Fza)))$ (de 2 avec *a* var. ancienne)
8. $\exists z\, (Fab \supset ((\sim Faz\ \&\ Fzb) \vee (Fbz\ \&\ Fza)))$ (de 7 avec *b* var. ancienne)
9. $Fab \supset ((\sim Fac\ \&\ Fcb) \vee (Fbc\ \&\ Fca))$ (de 8 avec *c* variable ancienne)

---

[6] Il faut noter que ce genre de difficulté est éludé dans le calcul des prédicats avec *fonctions de termes*. Car alors, les formules prenexes avec des quantificateurs universels placés dans la portée de quantificateurs existentiels (pour une formule située à droite de la flèche), sont remplacées par des formes de Skolem, ce qui revient à les «pré-instantier; dans un ordre donné. Cf. par exemple, J. P. Delahaye, *Outils logiques pour l'intelligence artificielle*, Eyrolles, Paris 1987, p. 127 et suiv.

[7] C'est-à-dire les instantiations qui produisent des formules atomiques comme *Fab*, et non comme $\exists x\, Fxa$. J'appelle *instantiation* l'application de l'une des quatre règles concernant les quantificateurs. En l'occurrence, il s'agit des instantiations utilisant des variables anciennes.

[8] T. de Quine, *Méthodes de logique*, chap. 29, A. Collin, Paris 1973, (p. 175 de la trad. française).

10. *Fab* (de 4)
11. *Fac* (de 4)
12. $\sim(Fbc \ \& \ Fca)$ (de 6)

Les formules 9, 10, 11, 12 étant logiquement incompatibles, la preuve est achevée. L'essentiel qui nous intéresse ici est le passage de la formule 7 à la formule 8. Le choix de la variable ancienne *b* est fait par unification de la sous-formule *Fbz & Fza* de 8 avec cette même sous-formule dans 6. Evidemment, la gestion informatique de ces opérations devient passablement complexe. A titre indicatif, je donne en annexe, la partie de programme pascal qui réalise l'unification dans le logiciel que j'ai mentionné. Et je m'adresse à qui veut, pour callaborer dans le perfectionnement de l'algorithme informatique de ce genre d'unification.

En définitive, bien que la voie «intelligente» soit souvent manifeste pour la déduction des formules les plus courtes et les plus banales (encore qu'il arrive fréquemment qu'un étudiant de force moyenne ne les voie pas), à examiner la question de plus près, on s'aperçoit que les problèmes posés sont très difficiles, sans parler de la mise en oeuvre informatique, qui devient vite épineuse. Cela est, si l'on veut, une conséquence directe de cet état de choses logique, mis en lumière par Gödel: le calcul des prédicats bien que comples, n'est pas décisoire. De là résulte qu'un algorithme de déduction planifiée est relativement facile à écrire, mais qu'en revanche la déduction intelligente réclame imagination, essais et «bricolages», avec en prime – si j'ose me permettre cette vulgarité – la certitude qu'on ne l'atteindra jamais.

Université de Nantes
France

Patrice Bailhache

POSZUKIWANIE PROCEDURY INTELIGENTNEGO DOWODZENIA TWIERDZEŃ

W ostatnich latach wraz z rozwojem nauk komputerowych, pojawiło się wiele programów mających wspomagać nauczanie logiki. Większość z tych programów służy do sprawdzania rachunków, a nie do dowodzenia twierdzeń, ponieważ zgodnie ze znanym twierdzeniem Gödla, nie istnieje procedura rozstrzygalności twierdzeń rachunku pierwszego rzędu. Kilka metatwierdzeń, w których jednym jest twierdzenie Gödla, tłumaczy dlaczego dysponujemy dobrymi systemami dedukcji naturalnej, mimo że tak trudno jest (a faktycznie nie można) znaleźć „inteligentną" procedurę dowodzenia formuł. W artykule przedstawiono drogę uzyskania częściowego systemu dowodzenia twierdzeń za pomocą procedury unifikacji, dla której załączono program komputerowy w języku pascal.

ANNEXE

ESSAI DE PROGRAMMATION PASCAL DE L'UNIFICATION

```
procedure DetermineVarinst (E: moyexpression);
 label
  atcstsuiv,fin,sortie,sortieD,sortieG,
  cvsortie,varinstauiv;
 var
  i,k,kk,l,n,t,tt,u:byte;
  FF:formexpression;
  F:moyexpression;
  ce:microexpression;
  c,cc,ccv,ci:char;
  varinstpure:tableaudecar;
  varlibegal,existeatomeconstpur,varpresente:boolean;

begin [de DetermineVarinst]
 for t:=1 to noeudmax do ATI^[t]:='';for r:=1 to nbratomesconstants do ATC^[t]:='';
 n:=0;DetermineAtome(E,'i');for t:=1 to noeudmax do if AT^[t]<>'' then begin inc(n);ATI^[t]:=AT^[t] end;
 nbratinst:=n;
 n:=0;i:=1;while FD^[i]<>''do
 begin
  DetermineAtome(FD^[i],'c');for t:=1 to noeudmax do
  if AT^[t]<>'' then begin inc(n);ATC^[n]:=AT^[t] end;
  inc(i)
 end;
 i:=1;while FG^[i]<>'' do
 begin
  DetermineAtome(FG^[i],'c');for t:=1 to noeudmax do
  if AT^[t]<>'' then begin inc(n);ATC^[n]:=AT^[t] end;
  inc(i)
 end;
 nbratconst:=n;
```

```
(ATI sont les atomes de la formule E à instantier. ATC sont les atomes qui
contiennent au moins une constante, de toutes les formules de la séquences.
Pour simplifier, les variables libres anciennes sont appelées ici „constantes")

nbrvarinst:=0;nbrvarinstpure:=0;
for t:=1 to nbratinst do
begin
 l:=length(ATI^[t]);
 for u:=1 to nbratconst do
 begin
  if copy(ATC^[u],1,1)<>copy(ATI^[t],1,1) then goto atcstsuiv;          (si les prédicats sont différents dans)
  for k:=2 to l do                                                      (l'ATI et l'ATC considérés, alors passe)
  begin
   ce:=copy(ATI^[t],k,1);ci:=ce[1];ce:=copy(ATC^[u],k,1);ccv:=ce[1];    (si à la position k)
   if (ccv in ['a'..'r','0'..'9'])and(ci=vi) then                       (ATC contient une constante)
   (vi, var d'instantiation, est déterminée dans la)                    (et ATI contient la var. d'inst. vi)
   (procédure UniverselG ou ExistentielD)
    begin
     varlibegal:=true;                                                  (alors, si à toutes les autres positions,)
     for kk:=2 to l do                                                  (lorsque ATI et ATC ont une constante, ces)
     begin                                                              (deux constantes sont identiques,...)
      ce:=copy(ATI^[t],kk,1);ci:=ce[1];ce:=copy(ATC^[u],kk,1);cc:=ce[1];
      if (ci in ['a'..'r','0'..'9'])and(cc in ['a'..'r','0'..'9'])and(ci<>cc) then varlibegal:=false;
     end;
     if varlibegal then
     begin
      for tt:=1 to nbrvarinst do if ccv=varinst[tt] then goto sortie;   (alors si cette constante de la)
      for tt:=1 to nbrvarinstpure do if ccv=varinstpure[tt] then goto sortie;  (position k est nouvelle....)
                                                                        (alors, après avoir déterminé si oui ou non)
    if existeverliee(ATC^[u]) then                                      (l'ATC contient une varlable liée,)
    begin                                                               (si c'est le cas, choisir la constante de la)
     inc(nbrvarinst);                                                   (position k comme var. d'instantiation varinst)
      varinst[nbrvarinst]:=ccv;
     end
     else                                                               (si ce n'est pas le cas, la choisir comme var.)
     begin                                                              (d'instantiation varinstpure)
      inc(nbrvarinstpure);
      varinstpure[nbrvarinstpure]:=ccv;                                 (Les varinstpure correspondent à des ATC sans)
     end;                                                               (variable liée, par ex. à Fab et non à Fax)
     sortie:
    end;
   end;
  end;(de for k:=2 to l do)
  atcstsuiv:
 end; (de for u:=1 to nbratconst)
end;(de for t:=1 to nbratinst)

DetermineVarlib;

(ajoute 'a' comme varinst s'il n'y a pas du tout de var libre dans la séquence)
if nbrvarinst+nbrvarinstpure=0 then
begin
 if nbrvarlib=0 then begin nbrvarinst:=1;varinst[1]:='a' end;
end;

(ajoute des variables libres comme varinst, si nécessaire; par ex. cas de la formule ]x$yFxy)
if nbrvarinst+nbrvarinstpure<nbrvarlib then
begin
 (ajoute une varlib comme varinst seulement si au moins un ATI contient une variable
 liée différente de la variable d'instantiation vi. Sinon, en effet, c'est-à-dire
 si tous les ATI contenaient seulement des var libres à part vi, on aurait obtenu
 la varlib comme varinst dans la première détermination des varinst s'il avait
 existé au moins un ATC avec var libres uniquement. Si le programme en est arrivé
 là, cela veut dire qu'il n'y pas de tel ATC. On est donc en droit de procéder
 à une instantiation qui ne va pas faire apparaitre un atome sans var liée, qui serait
 différent d'un atome ATC sans var liée également; par ex. on ne va pas faire
```

```
    apparaitre Fab à partir de Fxb, alors qu'il n'y aurait comme ATC que Faa et
    Fbb; mais on va faire apparaitre Fby à partir de Fxy s'il y a les seuls ATC
    Fay et Fab (comme dans le cas de la formule ]x$yFxy.)
    for n:=1 to nbrvarlib do
    begin
     for t:=1 to nbratinst do
     begin
      k:=pos(vi,ATI^[t]);if k<>0 then
      begin
       FF:=ATI^[t];
       Supprimevar(FF,vi);
       if existevarliee(FF) then
       begin
        varpresente:=false;
        for m:=1 to nbrvarinst do if varinst[m]=varlib[n] then varpresente:=true;
        if not varpresente then begin inc(nbrvarinst);varinst[nbrvarinst]:=varlib[n] end;
        goto cvsortie;
       end;
      end;
     end;
     cvsortie:
    end;(de n:=1 to nbrvarlib)

    if plusderegle then
    begin
     plusderegle:=false;
     varpresente:=false;
     for m:=1 to nbrvarinst do if varinst[m]=varlib[n] then varpresente:=true;
     if not varpresente then begin inc(nbrvarinst);varinst[nbrvarinst]=varlib[n] end;
    end;(if plusderegle)
   end;(if nbrvarinst+nbrvarinstpure<nbrvarlib)
    (classe les variables trouvées: d'abord varinst, puis varinstpure)
    ClassTableauCar(varinst,nbrvarinst); [cf. DNOUTIL]
    ClassTableauCar(varinstpure,nbrvarinstpure);
```

```
(intègre les varinstpure au début de la liste des varinst)
for t:=nbrvarinst downto 1 do varinst[t+nbrvarinstpure]:=varinst[t];
fpr t:=1 to nbrvarinstpure do varinst[t]:=varinstpure[t];
nbrvarinst:=nbrvarinst+nbrvarinstpure;

(supprime les variables qui figurent plusieurs fois)
t:=2;while t<=nbrvarinst do
begin
 for u:=1 to t-1 do if varinst[t]=varinst[u] then
 begin
  for tt:=t to nbrvarinst-1 do varinst[tt]:=varinst[tt+1];
  doc(nbrvarinst);goto varinstsuiv;
 end;
 inc(t);
 varinstsuiv;
end;

(Second critère de détermination des constantes d'instantiation:
propose seulement un classement différent pour les varinst déjà trouvées ci-dessus)
u:=0;
k:=1;while FH^[k]<>'' do
begin
 F:=FG^[k]; if F<>E then for t:=1 to nbrvarinst do if pos(varinst[t],F)<>0 then
 begin
  while (copy(F,1,1)='$')or(copy(F,1,1)=']') do F:=copy(F,3,length(F)-2);
  inc(u);FC^[u]:=F;goto sortieG
 end;
 sortieG:
 inc(k)
end;
k:=1;while FD^[k]<>'' do
begin
 F:=FD^[k];if F<>E then for t:=1 to nbrvarinst do if pos(varinst[t],F)<>0 then
 begin
```

(fait la liste des formules autres que)
(E contenant au moins une constante,)
(en les appelant FC[ ])

```
    while (copy(F,1,1)='$')or(copy(F,1,1)=']') do F:=copy(F,3,length(F) -2);
    inc(u);FC^[u]:=F;goto sortieD
   end;
   sortieD:
   inc(k)
  end;

  F:='';for k:=1 to u do if length(FC^[k])>length(F) then F:=FC^[k];
  F:=SubstitutionVarliee (F,'%');

  F:=SupprimeParGlobale (SupprimeNegGlobale(F));

  for t:=1 to nbrvarinst do
  begin
   ccv:=varinst[t];FF:=E;FF:=Substitution(FF,vi,ccv);
   FF:=SubstitutionVarliee(FF,'%');


   if pos(F,FF)<>0 then
   begin
    for u:=t downto 2 do varinst[u]:=varinst[u-1];
    varinst[1]:=ccv;goto fin
   end;
  end;
  fin:
  (fin du Second critère de détermination des constantes d'instantiation)

end: (de procedure DetermineVarinst)
```

```
(choisit la FC de longueur maximale)
(et y substitue le caractère % à toutes)
(ses variables liées)
(cela donne la formule F)

(Pour chaque varinst déterminée par le premier)
(critère, la substitue à la variable concernée)
(dans l'expression E à instantier,...)
(en y substituant également % partout où elle)
(a une variable liée. D'où la formule FF)

(Si F est incluse dans FF pour telle)
(varinst, alors placer cette constante)
(au début de la liste des varinst.)
```