POUR UNE PREMIERE INITIATION A TURBO PROLOG

Maurice BOTTIN, Jean SEYVOZ

SOMMAIRE

1 INTRODUCTION

Pourquoi une initiation à Turbo Prolog ?

Les recherches dans le domaine de l'intelligence artificielle poursuivent deux objectifs principaux :

- Développer des modèles informatiques du comportement intelligent.
- Développer des systèmes informatiques permettant de résoudre des problèmes pour lesquels aucune méthode standard n'est connue.

Les enseignants sont évidemment intéressés par de telles recherches, et le langage Turbo Prolog nous semble bien adapté à un travail dans les écoles et les lycées qui permettrait aux élèves et aux professeurs de rester en liaison avec le domaine des langages d'intelligence artificielle.

D'autres points très importants justifient, à notre avis, ce travail :

- Ce langage permet aux élèves une approche facile de l'informatique : en effet, il est très facile de stocker et d'interroger une base de données, ce qui n'est pas le cas avec un langage traditionnel.
- Résoudre un problème, avec ce langage, revient à le poser correctement. Une étude concernant la compréhension des énoncés peut être menée de façon intéressante.
- Il est possible de fabriquer des "mini systèmes-experts" pouvant servir d'outils d'enseignement.

2 PROGRAMME PREVISIONNEL

Première journée

Accueil des stagiaires.

Programme du stage.

Généralités sur Turboprolog, langage déclaratif.

Travail en Turboprolog :

Entrer une base de faits et l'interroger.

- Exemples de "programmes" simples :

  relations : aime, habite, appartient_à_la_classe.

- Interroger en goal :

  interrogation simple.

  interrogation utilisant le "et" logique.

Deuxième journée

Introduction des règles :

- Programmes "classiques" : dîner, arbre généalogique.

  Opérations arithmétiques :

- résolution d'équations, exercices à "trous".
- problèmes simples : UNE+UNE=DEUX.

  SEND+MORE=MONEY.

Travail en Prolog nano réseau.

Troisième journée

Visite du CETE d'AIX-Les Milles (Centre des Etudes Technique et de l'Equipement)

Quatrième journée

Etude des listes.

- Tête et queue de liste.
- Fabrications d'outil

  membre, renverser, concatener, longueur, hors_de, supprimer, nième_élément.

Etude des bases de données dynamique.

Fabrication d'un système-expert de diagnostic.

Cinqième journée

Réflexion sur les applications possibles de ce langage dans l'enseignement primaire.

Projets de recherches.

3 RÉSUME SUCCINCT DES SÉANCES

Première séance : matin

Généralités sur le langage, constituer une base de faits et interroger cette base

1 Généralités

Cette séquence avait pour objectif de familiariser les stagiaires avec le langage, nous la détaillerons pour mettre en évidence les facilités offertes à des débutants en informatique.

Grâce à l'environnement intégré, multifenêtres, l'utilisateur sait, à tout moment, où il en est.

- L'éditeur est d'un emploi facile, tout en offrant des possibilités d'un bon traitement de texte.
- La compilation est rapide, elle renvoie à l'éditeur en cas d'erreur.
- L'exécution est, en général, très rapide puisque le "programme" est en langage machine.

Prolog est un système gestionnaire de bases de données : il permet d'entrer des faits que l'on pourra mettre en relation par des règles.

2 Constitution d'une base de faits

L'aspect déclaratif du langage rend possible le stockage d'une suite de relations. Nous allons prendre un exemple : nous désirons entrer une suite de faits mettant en relation les noms d'un certain nombre d'individus et leurs goûts.

La phrase « Durand aime le football » se traduit en Turbo Prolog par `aime(durand, football).` et s'appelle alors une clause ; "aime" est le nom du lien verbal que l'on vient de créer, il porte, ici, sur deux objets : durand et football.

Dans ce langage, il faut faire précéder l'ensemble des clauses par une partie appelée "predicates" dans laquelle on définit le nom des liens verbaux crées et le type des objets sur lesquels portent ces liens.

Ces types font référence à des types prédéfinis reconnus. (Voir annexe).

Ici durand et football sont de type "symbol".

- symbol : suite de lettres, nombres et caractères de soulignement, commençant par une lettre minuscule, ou toute suite de signes entre guillemets.

Les faits seront, ensuite, écrits dans la section "clauses".

Le programme aura la forme suivante :

```
predicates
clauses
  aime(durand, football).     <- ne pas oublier le point final
  aime(durand, tennis).
  aime(louis, ...).
  etc...
```

Voir, en annexe, les programmes écrits par les stagiaires.

Bien entendu le langage ne peut pas contrôler la pertinence des informations et accepte donc des clauses du type :

```
  aime ( xc23bn , azert).
```

3 Interrogation de la base de faits

Une fois les clauses écrites dans la fenêtre EDITEUR, il faut pour interroger cette base passer en mode RUN (option RUN du menu principal).

Le programme est compilé, c'est-à-dire traduit en langage machine.

Pendant cette phase, les erreurs de syntaxe sont signalées et la correction est attendue.

"Goal" s'affiche alors, dans la fenêtre réservée à l'exécution : Turbo Prolog est en attente d'un but à satisfaire.

Plusieurs possibilités s'offrent à l'utilisateur :

- S'il tape "`aime(durand, football)`", puis valide, ce fait se trouvant dans la base de données, la réponse va s'afficher : "`TRUE`".
- S'il tape un fait n'existant pas, la réponse est "`FALSE`".
- S'il tape `aime(durand, X)`, après validation, tous les sports qu'aime durand vont s'afficher. Exemple :

```
goal : aime(durand, X)
X=football
X=tennis
2 solutions
```

- Les questions peuvent prendre une forme plus complexe, en utilisant le "`et`" logique, qui s'écrit "`and`" ou "`,`".

  Exemple : si on désire connaître les sports communs qu'aiment caroline et durand, on tapera :

```
aime(caroline, X) and aime(durand, X).
```

Remarque : dans les cas 3 et 4, la variable X a été utilisée. Turbo Prolog reconnait une variable au fait qu'elle commence par une majuscule. On aurait pu écrire "aime(durand, Loisir)", et les résultats seraient apparus de manière plus explicites.

```
goal : aime(durand, Loisir)
Loisir=football
Loisir=tennis
2 solutions
```

Remarque : On peut, tout aussi bien, écrire "Animal" au lieu et place de "Loisir", la réponse est alors : Animal = footall !

4 Les règles

Dans une troisième partie, les règles ont été introduites.

Si Dubois a les mêmes goûts que Durand, il n'est pas utile de tout réécrire, sous la même forme. Il suffit d'entrer une règle spécifiant que Dubois aime ce qu'aime Durand.Les sports seront figurés par une variable pouvant, par exemple, être appelée Sport (la majuscule du début est indispensable) et on écrira :

```
aime(dubois, Sport) if aime(durand, Sport).
```

On voit donc qu'une règle comporte 3 parties :

- La tête de règle (prédicat unique précédant le if)
- La queue de règle (ce qui suit le if)
- Le point final.

Remarque : un "programme" TURBOPROLOG peut contenir plusieurs prédicats, correspondant à des problèmes distincts, exemples :

```
aime ( symbol , symbol )
solution
identité( symbol , symbol , integer , symbol , symbol )
  ... etc...
```

Exercice proposé : Écrire le prédicat "grouper" qui donne les paires de personnes ayant des goûts communs.

Deuxième séance : après-midi

Amélioration de la lisibilité :

Le prédicat "identité", écrit ci-dessus est, tel quel, assez sybilin ; on peut le rendre plus explicite en le rédigeant ainsi :

```
identité ( prénom , sexe , âge , tel , ville )
```

Cela revient à utiliser de nouveaux types, mais, dans ce cas, il faut les créer : la section "predicates" doit alors, être obligatoirement précédée par une section appelée "domains" dans laquelle on indique la correspondance entre les mots utilisés (pour désigner les types des arguments des prédicats) et les types prédéfinis :

```
domains
  prénom , sexe , tel , ville = symbol
  âge = integer
predicates
  identité ( prénom , sexe , âge , tel , ville )
```

```
clauses
  identité ( raoul , m , 34 , "42.21.10.99", marseille )
  identité ( "Alexandra", f , 25 , poste_28 , "AIX-en-Pce" )
         etc ...
```

Exercices proposés :

- entrer quelques clauses de ce type
- essayer quelques interrogations.

Remarque : Ce prédicat permet de répondre à de nombreuses questions : « Quelles sont les femmes de cette base ? »

L'interrogation peut être de la forme :

```
goal : identité(Prénom, f, _, _, _)
```

dans laquelle le souligné "_" représente ce que l'on appelle une variable anonyme. Le langage fournira tous les prénoms pour lesquels le deuxième paramètre est "f".

Citer les prénoms de toutes les personnes de la base.

Où habite une personne donnée ?, ...

- pour éviter de taper, à chaque interrogation, tous les paramètres de ce prédicat, (y compris les variables anonymes), il est commode d'en créer d'autres avec les seuls paramètres dont on a besoin.

Exercice proposé : Créer les prédicats suivants :

```
personne(P)  femme(F)     habite( Pers , Lieu )
  majeur(Pers) est_dun_âge_supérieur_à (Pers1 , Age , Pers2)
  doyen( D )
```

Autre remarque : à ce stade là on constate que Turbo Prolog est bien un SGBD (système gestionnaire de bases de données) plutôt plus simple à utiliser qu'un autre.

Avec dBase, par exemple, il faut :

- 1. créer le fichier, en utilisant la commande `create`, répondre à toutes les questions, y compris fixer la longueur maximum des champs
- 2. saisir les fiches
- 3. l'interrogation se fait ensuite en posant des questions en utilisant la commande `list` :

```
list prénom,sexe,age  for prénom="louis"
```

En TURBOPROLOG :

1. correspond à "predicates", 2. à "clauses et 3. à goal :

```
personne( louis , Sexe , Age , _ , _ )
```

On peut ensuite écrire un programme donnant tous les dîners possibles, des hors-d'œuvre, des plats de résistance, des desserts étant entrés dans une base de faits.(Voir programme dîner).

Troisième séance matin

Cette séance portait sur l'écriture de quelques prédicats arithmétiques :

1 Générer la suite des nombres entiers

les nombres entiers à un seul chiffre

générer les nombres entiers à deux chiffres

2 résoudre les problèmes d'"opérations à trous", exemple :

```
   9 . 8
 + . 4 .
----------
   . 5 . 3
```

ou ceux du type U N E + U N E = D E U X dans laquelle chaque lettre représente un chiffre, deux lettres différentes représentant deux chiffres différents. Voir les solutions en annexe.

Remarque : Pour résoudre ces problèmes, il suffit de déclarer l'énoncé. TURBOPROLOG recherche lui-même, toutes les solutions, de manière systématique, méthodique, en essayant tous les chiffres un à un!

Quatrième séance après midi

Prolog FIL sur NR MO5

- arbre généalogique	- description
	- liens de parenté
- les listes	- écriture (a,b,c, ... n) ou (*T ; *Q)
	- gestion
- quelques utilitaires	- APPartient à
	- CONCaténer
	- INVerser
	- COMPTer ...

Cinquième séance

Visite du C.E.T.E. conférence et démonstration par des concepteurs et des utilisateurs de systèmes-experts

Sixième séance

Principes et écriture d'un mini-système-expert

1 - Qu'est-ce qu'un système expert ?

c'est un système informatisé

a) permettant d'aider un spécialiste dans un certain domaine

- à raisonner et à résoudre ses problèmes
- à prendre des décisions

b) conçu et construit par une équipe constituée d'informaticiens et d'experts, non informaticiens, ayant un savoir-faire et une bonne connaissance du domaine.

2 - De quoi se compose un système-expert ?

a) La connaissance, sous deux formes :

- des faits
- des règles de raisonnement sur ces faits

Cette connaissance n'est pas figée définitivement dans le programme, elle est accessible à l'expert qui peut ajouter ou soustraire des faits ou des règles, à tout moment

b) Le moteur d'inférence qui permet de faire les déductions et surtout, d'explorer toutes les solutions possibles

c) des logiciels interactifs permettant un dialogue simple entre l'homme et la machine.

Les stagiaires, sous forme de travail dirigé, ont constitué un pseudo système-expert d'aide au diagnostic médical

la base de faits est du type :

```
maladie ( nom , liste des symptômes de cette maladie )
```

Septième séance

Cette séquence avait pour but de montrer comment ce langage permet de résoudre un problème apparemment complexe, en représentant convenablement la connaissance et les buts à atteindre.

Il est très important de remarquer que, celui qui écrit le programme n'a pas à connaître la méthode de résolution du problème, si elle existe. Ce n'est pas le cas avec les langages algorithmiques.

Ici, la méthode employée consiste à mettre des listes en correspondance.

Voici le texte de l'exercice proposé :

Problème du zèbre

Dans ce problème, on travaille sur :

5 maisons, 5 hommes, 5 nationalités, 5 couleurs, 5 animaux, 5 boissons, 5 marques de cigarettes.

Les assertions suivantes sont énoncées :

- L'Anglais habite la maison rouge.
- Le chien appartient à l'Espagnol.
- On boit du café dans la maison verte.
- L'Ukrainien boit du thé.
- La maison verte est à côté de la blanche, à droite.
- Le fumeur de Old Gold élève des escargots.
- On fume des Kools dans la maison jaune.
- On boit du lait dans la maison du milieu.
- Le Norvégien habite la première maison à gauche.
- Le fumeur de Chesterfield habite à côté du propriétaire du renard.
- Le fumeur de Gitanes boit du vin.
- Le Japonais fume des Cravens.
- Le Norvégien habite à côté de la maison bleue.
- Le fumeur de Kools habite à côté du propriétaire du cheval.

Qui boit de l'eau ?

A qui appartient le zèbre ?

Autre exercice proposé : les métiers. (Un cas plus simple d'énigme.) Voir en annexe les solutions des stagiaires.

4 MISE EN ROUTE DE PROLOG SUR PC

Algorithme impératif

1 - Premières opérations

1.1 Mettre l'appareil sous tension.

1.2 Charger le système d'exploitation

Attendre ... de voir apparaître : A>

1.3 Charger Turbo Prolog

Taper (à côté de A>) prolog puis ↵

Attendre... de voir apparaître le premier écran prolog.

2 - Utiliser Turbo Prolog

2.1 appuyer sur une touche pour obtenir l'écran à quatre fenêtres : éditeur, dialogue, messages, trace.

2.2 entrer dans le mode éditeur : ESC puis E

2.3 écrire le programme...

... domains predicates clauses...

2.4 sortir du mode éditeur pour :

- compiler le programme : ESC puis C
- obtenir l'exécution : ESC puis R

etc. revenir au point 2.2 autant de fois que nécessaire.

2.5 sauver le programme sur disquette :

- placer une disquette de travail dans le lecteur A :
- taper ESC puis F puis S
- taper (à côté de file name :) nom-du-programme ↵ (attention : 8 caractères au plus pour le nom!)

2.6 effacer le contenu de l'éditeur : ESC puis F puis Z. On vous demande ensuite le nom du nouveau programme.

- taper (à côté de file name :) nom-du-programme ↵ (attention : 8 caractères au plus pour le nom!)

2.7 charger un programme de la disquette :

- ESC puis F puis L
- taper nom-du-programme puis ↵.

5 LES TYPES PRÉDÉFINIS EN TURBO PROLOG

char : (caractère) tout élément unique du clavier figurant entre apostrophes(') : 'c' par exemple .

integer : tout nombre entier de -32768 à +32767

real : tout nombre entier ou décimal de ±1E-307 à ±1E+308. Les entiers sont automatiquement convertis en réels si nécessaire.

string : (chaîne) toute suite de caractères entre guillemets.

Exemples : `"Programmation en Turbo Prolog"` .

symbol : (symbole) Relèvent de ce type :

(1) toute suite de lettres, chiffres et caractères de soulignement, à la seule condition de commencer par une lettre minuscule.

(2) toute suite de signes entre guillemets (")

exemples :

(1) `numéro_de_téléphone` `james_bond_007`

(2) `"Agent de service"` `"24 h / 24"`

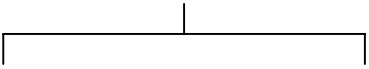
file : nom symbolique d'un fichier

N.B. Les autres types, définis par le programmeur, sont à déclarer dans la section `DOMAINS` .

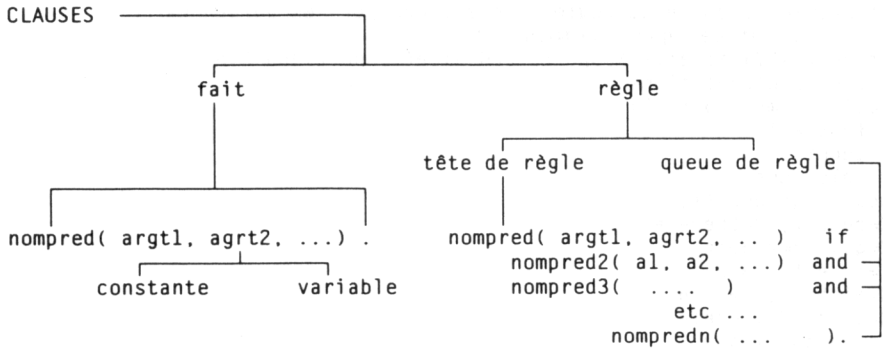
6 STRUCTURE GÉNÉRALE D'UN PROGRAMME EN TURBO PROLOG

```
DOMAINS       nom-du-type-à-définir = type-prédéfini
 quelques exemples :
         nom, prénom, marque, ville = symbol
         quantité, âge, nombre = integer
         prix, kilométrage = real

PREDICATES   nom-de-prédicat ( typeargt1 , typeargt2 , ...  )
```



```
          type prédéfini     ou    défini dans section
        exemples :
        solution ( integer, integer, integer )
        voiture ( marque, âge, kilométrage, prix )
        truc ( symbol, prénom, integer, poids )
```



variable :

1 lettre MAJUSCULE suivie, éventuellement, d'autres lettres ou signes, exemples : X Prénom JBOO7 Km Ville

constante :

1 nombre 34

ou

1 lettre minuscule suivie, éventuellement, d'autres lettres ou signes, exemples : marseille toto louisXI

ou

une chaîne de caractères entre guillemets, exemples : "Aix-en-Provence" "1989" "HENRI IV" .

7 EXEMPLES DE SOLUTIONS

```
/*  gérer une base simple */
predicates
        aime(symbol,symbol)
        grouper(symbol,symbol)
clauses
        aime(robert,football).
        aime(guy,tennis).
        aime(robert,tennis).
        aime(michele,basket).
        aime(jean,basket).
        aime(joseph, golf).
        aime(guy,golf).
        aime(therese,football).
        aime(toto,informatique).
```

```
        aime(turbo,prolog).

/* premier exemple de règle simple : grouper les personnes
        ayant des goûts communs  */
grouper(X,Y) if aime(X,Sport) and aime(Y,Sport)and X<>Y.
Exemples d'exécutions
goal : aime(robert,S)
S=football
S=tennis
2 solutions
goal : grouper(X,Y)
X=robert, Y=therese
X=guy, Y=robert
X=robert, Y=guy
X=michele, Y=jean
X=jean, Y=michele
X=joseph, Y=guy
X=guy, Y=joseph
X=therese, Y=robert
8 solutions
/*  construire des prédicats de gestion d'une base  */
/* amélioration de la lisibilité en définissant des types */
domains
        prenom,sexe,tel,ville=symbol
        âge=integer
predicates
        identité(prenom,sexe,âge,tel,ville)
        personne(prenom)
        femme(prenom)
        habite(prenom,ville)
        majeur(prenom)
        est_âgé_de(prenom,âge)
        person
clauses
/*  la base de données  */
        identité(louis,m,38,"42.38.92.71",aix_en_provence).
        identité(jean,m,39,"41.38.92.71",nice).
        identité(jojo,m,39,"51.38.92.71",orleans).
        identité(josette,f,39,"38.38.92.71",orleans).
```

```
/* les predicats de gestion   */
       personne(P) if identité(P,_,_,_,_).
       femme(P)if identité(P,Q,_,_,_)and Q=f.
       habite(X,Y)if identité(X,_,_,_,Y).
       majeur(X) if identité(X,_,Z,_,_) and Z>17.
       est_âgé_de(P,A) if identité(P,_,A,_,_).
/*     le diner   */
predicates
       entrée(symbol,real)
       plat(symbol,real)
       dessert(symbol,real)
       diner(symbol,symbol,symbol,real)
clauses
       entrée(jambon,12.50). entrée( foie_gras,80).
              entrée(oeuf,7.80).
       plat(roti,25.10).plat(poisson,25).plat(poulet,18).
       dessert(tarte,12).dessert(glace,18).dessert(fromage,16).

       diner(X,Y,Z,Prix) if
       entrée(X,A) and plat(Y,B) and dessert(Z,C) and Prix=A+B+C.
Interrogations :
goal : diner(H, poisson, Prix)
H=jambon, D=tarte, Prix=49.5
H=jambon, D=glace, Prix=55.5
H=jambon, D=fromage, Prix=53.5
H=foie_gras, D=tarte, Prix=117
H=foie_gras, D=glace, Prix=123
H=foie_gras, D=fromage, Prix=121
H=oeuf, D=tarte, Prix=44.8
H=oeuf, D=glace, Prix=50.8
H=oeuf, D=fromage, Prix=48.8
9 solutions
goal : diner(H, poisson, Prix) and Prix < 50
H=jambon, D=tarte, Prix=49.5
H=oeuf, D=tarte, Prix=44.8
H=oeuf, D=fromage, Prix=48.8
3 solutions
goal : diner(H, poisson, Prix) and Prix > 120
```

```
H=foie_gras, D=glace, Prix=123
H=foie_gras, D=fromage, Prix=121
2 solutions
/*  arithmétique    */
/*  ch = nombre d'un chiffre   nb2ch = nombre à deux chiffres */
predicates
       ch(integer)  nb2ch(integer)
       solution(integer,integer,integer,integer,integer)
clauses
       ch(0). ch(1). ch(2). ch(3). ch(4). ch(5). ch(6). ch(7).
        ch(8). ch(9).

       /* générer les nombres entiers à 2 chiffres  */
       nb2ch(N) if ch(X) and X >< 0  and  ch(Y)  and N = 10 * X + Y
        .
       /* rechercher toutes les solutions pour une opération à
        trous
              9 . 3
           +  . 4 .
           --------
           .  3 . 8          solution sans présentation   */
solution(X,Y,Z,T,U) if
             ch(X),ch(Y),ch(Z),ch(T),ch(U),
             900+10*X+3 + 100*Y+40+Z = 1000*T+300+10*U+8.
goal : nb2ch(45)   -->     True
goal : nb2ch(6)    -->     False
goal : nb2ch(N)
N=11
N=12
N=13
N=14
N=15
N=16
N=17
N=18
N=19
N=20
N=21
N=22
```

```
N=23
N=24
N=25
N=26
N=27      etc...  jusqu'au débordement de pile après N=746
goal : solution(D1,C2,U2,M,D)
D1=0, C2=4, U2=5, M=1, D=4
D1=1, C2=4, U2=5, M=1, D=5
D1=2, C2=4, U2=5, M=1, D=6
D1=3, C2=4, U2=5, M=1, D=7
D1=4, C2=4, U2=5, M=1, D=8
D1=5, C2=3, U2=5, M=1, D=9
D1=6, C2=3, U2=5, M=1, D=0
D1=7, C2=3, U2=5, M=1, D=1
D1=8, C2=3, U2=5, M=1, D=2
D1=9, C2=3, U2=5, M=1, D=3
10 Solutions
/* casse-tête arithmétique   UNE + UNE = DEUX */
predicates
       ch(integer)
       solution(integer,integer,integer,integer,integer)
       sol  sol3
       clauses
       ch(0). ch(1). ch(2). ch(3). ch(4). ch(5). ch(6). ch(7).
        ch(8). ch(9).
/* première version avec paramètres  ...   peu  efficcace  */
solution(U,N,E,X,D) if
       ch(U),ch(N),ch(E),ch(X),ch(D),
       D<>0,U<>N,U<>E,U<>X,U<>D,N<>E,N<>X,N<>D,E<>X,E<>D,X<>D,
       2*(100*U+10*N+E)=1000*D+100*E+10*U+X.

/* deuxième version sans paramètres  ... même efficcacité ! */
sol if
       ch(U),ch(N),ch(E),ch(X),ch(D),
       D<>0,U<>N,U<>E,U<>X,U<>D,N<>E,N<>X,N<>D,E<>X,E<>D,X<>D,
       2*(100*U+10*N+E)=1000*D+100*E+10*U+X,
       write("  ",U,N,E),nl, write("+ ",U,N,E),nl,write("------
        "),nl,
       write("=",D,E,U,X) ,nl, fail.
```

```
sol.
/*  troisième version    plus rapide  */
sol3 if
       ch(U),
       ch(N),N<>U,
       ch(E),E<>U, E<>N,
       ch(X),X<>U, X<>N, X<>E,
       ch(D),D<>0,D<>U, D<>N, D<>E, D<>X,
       2*(100*U+10*N+E)=1000*D+100*E+10*U+X, nl,
       write("  ",U,N,E),nl, write("+ ",U,N,E),nl,write("------
        "),nl,
       write("=",D,E,U,X) ,nl, fail.
sol3.
goal : sol3
  632
+ 632
-----
=1264
  785
+ 785
-----
=1570
  948
+ 948
-----
=1896
True
/* Principe d'un  mini-système-expert   */
domains
       symptome=symbol
       liste_de_symptomes=symptome*
database
       oui(symptome) non(symptome)        mal(symbol)
predicates
       aide   maladie(symbol,liste_de_symptomes)
       question(liste_de_symptomes)
       tester(symptome)     analyser(char,symptome)  viderbase
clauses   /*  la  micro_base  de données */
       maladie(rougeole,[fièvre,toux,conjonctivite,éruption]).
```

```
            maladie(grippe,[fièvre,céphalées,conjonctivite,frisso
        ns,toux, mal_de_gorge]).
       maladie(rhume,[céphalées,éternuements,douleurs_corporelles,
        frissons, mal_de_gorge]).
       maladie(varicelle,[douleurs_corporelles,frissons,fièvre,
        éruption]).
       maladie(coqueluche,[toux_rauque,éternuements]).
/*  les règles d'aide au diagnostic  */
       aide if viderbase and maladie (Nom,L) and
              question(L) and
              write("   ...   Le malade a peut-être, ", Nom), nl ,
              asserta( mal(Nom)), nl,  fail.
       aide if not(mal(_)), write("je donne ma langue au chat"),
        nl.
         aide.
       question ([]).
        question ([T|Q])if tester (T) and
                     oui(T) and question (Q).
       tester (X) if oui(X) ,!.
        tester (X) if not(oui(X)), not(non(X)),
               write ("Le malade a-t-il ",X, " ? (o/n) : "),
               readchar (R), write (R) , analyser (R,X),nl.
       analyser ('o',S) if assertz(oui(S)),!.
        analyser ('O',S) if assertz(oui(S)),!.
         analyser ('n',S) if assertz(non(S)).
          analyser ('N',S) if assertz(non(S)).
           analyser (_,S) if tester(S).
       viderbase if retract(oui(_)), fail.
        viderbase if retract(non(_)), fail.
         viderbase if retract(mal(_)), fail.
          viderbase.
goal : aide
le malade a-t-il fièvre ?(o/n) : o
le malade a-t-il toux ?(o/n) : o
le malade a-t-il conjonctivite ?(o/n) : o
le malade a-t-il éruption ?(o/n) : o
...   Le malade a peut être, rougeole
le malade a-t-il céphalées ?(o/n) : n
```

```
le malade a-t-il douleurs_corporelles ?(o/n) : n
le malade a-t-il toux_rauques ?(o/n) : n
True
  /*  Problème du zèbre */
domains
       s = symbol    l = s*  /* l est donc une liste de symboles */
predicates
       elem (s,l)    mpos (s,s,l,l)
       voisindr (s,s,l) voisin (s,s,l)    voisin2(s,s,l,l)
       lm(l) lc(l) lna(l) lci(l) lb(l) la(l)
       sol c(s) n(s) b(s) ci(s) a(s)
clauses
       c(verte).c(jaune).c(rouge).c(blanche).c(bleue).
       n(norv).n(anglais).n(ukr).n(jap).n(espagnol).
       b(eau).b(the).b(cafe).b(vin).b(lait).
       ci(old).ci(chest).ci(gitanes).ci(craven).ci(kool).
       a(chien).a(zebre).a(escargots).a(renard).a(cheval).
       lm([a,b,c,d,e]).
       lna([X,Y,Z,T,W]) if
              n(Y),n(Z), Y<>Z,n(T),Y<>T,n(W),
              Y<>W,Z<>T,Z<>W,T<>W,n(X),X<>Y,X<>Z,X<>T,X<>W.
       lc([X,Y,Z,T,W]) if
              c(Y),c(Z), Y<>Z,c(T),Y<>T,c(W),
              Y<>W,Z<>T,Z<>W,T<>W,c(X),X<>Y,X<>Z,X<>T,X<>W.
       lb([X,Y,Z,T,W]) if
              b(Y),b(Z), Y<>Z,b(T),Y<>T,b(W),
              Y<>W,Z<>T,Z<>W,T<>W,b(X),X<>Y,X<>Z,X<>T,X<>W.
       lci([X,Y,Z,T,W]) if
              ci(Y),ci(Z), Y<>Z,ci(T),Y<>T,ci(W),
              Y<>W,Z<>T,Z<>W,T<>W,ci(X),X<>Y,X<>Z,X<>T,X<>W.
       la([X,Y,Z,T,W]) if
              a(Y),a(Z), Y<>Z,a(T),Y<>T,a(W),
              Y<>W,Z<>T,Z<>W,T<>W,a(X),X<>Y,X<>Z,X<>T,X<>W.
       elem (X,[X|_]) if !.
       elem (X,[_|Q]) if elem (X,Q).

       mpos (X,Y,[X|_],[Y|_]).
       mpos (X,Y,[_|Q1],[_|Q2]) if mpos (X,Y,Q1,Q2).
```

```
        voisindr (X,Y,[T|Q]) if mpos(Y,X,[T|Q],Q).

        voisin (X,Y,L) if voisindr (X,Y,L).
        voisin (X,Y,L) if voisindr (Y,X,L).

        voisin2 (X,Y,L,M) if
               voisin(X,Z,L),mpos(Y,Z,M,L).

sol if
lc(C),voisindr (verte,blanche,C),
lm(M),lna(N),mpos(a,norv,M,N),mpos(anglais,rouge,N,C),
voisin2(norv,bleue,N,C),la(A),mpos(chien,espagnol,A,N),
lci(CI),mpos(old,escargots,CI,A),mpos(kool,jaune,CI,C),
voisin2(chest,renard,CI,A),mpos(jap,craven,N,CI),
       voisin2(kool,cheval,CI,A),
lb(B),mpos(lait,c,B,M),mpos(cafe,verte,B,C),
mpos(ukr,the,N,B),mpos(gitanes,vin,CI,B),
write(M),nl,write(C),nl,write(N),nl,write(B),nl,
write(CI),nl,write(A),nl.
/*  casse-tête  énigme       les trois métiers
       énoncé : Trois amis  André, Bernard et Camille  exercent
       chacun un métier différent. Trouvez le métier de chacun
         sachant :
       - si André est caissier alors Camille est comptable
       - si André est comptable alors Camille est secrétaire
       - si Camille n'est pas caissier alors Bernard est comptable
       - si Bernard est secrétaire alors André est comptable     */
predicates
       metier(symbol)
       c1(symbol,symbol) c2(symbol,symbol)
       c3(symbol,symbol) c4(symbol,symbol)
       solution
clauses
       /*  les métiers  */
       metier(comptable). metier(caissier). metier(secretaire).
/* soient  X le métier de André, Y celui de Bernard et Z celui de
        Camille */
       /*   les conditions
```

```
       ( si A  alors  B )  se traduit en logique par : ( non(A) ou
        B )
             ou encore par  ( B ou non(A) )                    */
       c1(X,Z) if Z=comptable, !  ; not (X=caissier) .
       c2(X,Z) if Z=secretaire, ! ; not(X=comptable).
       c3(Z,Y) if Y=comptable, !  ; Z=caissier.
       c4(Y,X) if X=comptable, !  ;not(Y=secretaire).
solution if
       metier(X),
       metier(Y), X<>Y,
       metier(Z),Y<>Z,X<>Z,
             c1(X,Z),c2(X,Z),c3(Z,Y),c4(Y,X),
             write("André est : ",X),nl,
             write("Bernard est : ",Y),nl,
             write("Camille est : ",Z),nl, fail.
solution .

goal : solution
André est secrétaire
Bernard est comptable
Camille est caissier
```

8 LA PROGRAMMATION EN TURBO PROLOG

Avec les langages algorithmiques traditionnels, les programmeurs passent beaucoup de temps à écrire laborieusement des programmes que la machine exécute ensuite, en peu de temps, lorsqu'ils sont au point.

Le but des auteurs de Prolog était de produire un langage de programmation qui permettrait d'écrire facilement et rapidement les programmes, même si l'ordinateur mettait ensuite beaucoup de temps pour donner les réponses.

Pour une certaine classe de problèmes la seule déclaration correcte de l'énoncé permet au langage de trouver les réponses. Il trouve même toutes les réponses possibles

Ce qu'on appelle programmer dans un langage déclaratif consiste à écrire correctement les clauses traduisant les données du problème.(Voir n.b. ci-dessous). Lors de l'exécution, il faut alors poser les questions convenablement.

En effet, il existe dans ces langages un "moteur d'inférences" qui permet la résolution du problème.

A titre d'illustration, étudions de nouveau le "programme" traité lors de la première séance, pour montrer comment fonctionne ce langage.

Le programme s'écrivait :

```
predicates
        aime(symbol,symbol)
clauses
        aime(durand,football).
        aime(durand,tennis).
```

Supposons qu'à l'exécution soit posée la question :

```
        aime(durand,X)
```

Le langage essaie de mettre en correspondance la demande écrite et les clauses de la base de données.

La première mise en correspondance est effectuée avec la clause numéro 1, la solution X="football" est trouvée.

Même chose avec la clause numéro 2, ce qui permet de découvrir la deuxième solution : X="tennis".

Si la question posée est maintenant :

```
        aime(X,tennis)
```

La comparaison se fait avec la première clause :

La valeur "durand" est donc attribuée à X, mais la correspondance "tennis" "football" est impossible : il y a donc échec.

Par contre, il y aura succés en mettant en correspondance avec la clause numéro 2. La solution X="durand" sera affichée.

Remarque : Cet exposé est simplifié, en réalité la ou les variable(s) utilisée(s) est(sont) renommée(s) avant la mise en correspondance.

Nota bene :

L'ordre dans lequel on écrit les clauses influe beaucoup sur le temps de recherche.

On peut donner comme exemple le problème de la résolution du casse-tête : "UNE + UNE = DEUX".

Version numéro 1 qui vient naturellement à l'esprit

- écrire que les variables : U, N... sont des chiffres

ch(U), ch(N), ch(E)...

- écrire les conditions portant sur ces variables :
  D non nul, tous les chiffres différents 2 à 2
  D<>0, N<>U, E<>U, E<>N...
- écrire enfin la relation liant les nombres
  1000*D+100*E+10*U+X = 2*(100*U+10*N+E)

Cette version mettra beaucoup de temps pour donner les réponses ...

Version numéro 2

- écrire les conditions portant sur une variable, dès le choix de celle-ci
  ch(U),
  ch(N), N<>U,
  ch(E), E<>U, E<>N, etc.

Cette version est nettement plus rapide que la précédente.

CONCLUSION

Nous ne donnons, ici, qu'un bref aperçu des possibilités offertes et de la manière d'écrire des "programmes" dans ce langage.

Les premières expériences faites avec des élèves de l'école primaire nous semblaient prometteuses. Des publications suivront si cette tendance se confirme.

BOTTIN Maurice, Physique
SEYVOZ Jean, Mathématiques
École Normale Mixte
2 Avenue J. ISAAC
13626 AIX-EN-PROVENCE CEDEX