PROLOG II

MANUEL D'UTILISATION

Mars 1982

Michel VAN CANEGHEM

PROLOG II est un système qui a été développé par:

M. van Caneghem, A. Colmerauer, H. Kanoui

Groupe Intelligence Artificielle ERA CNRS 363.
Faculté des Sciences de Luminy Case 901
70, Route Léon Lachamp
13288 MARSEILLE Cedex 9
tel: (91) 41 01 40

AVANT PROPOS

C'est souvent celui qui vient de terminer une implantation qui doit écrire le manuel d'utilisation. Avant de commencer cette documentation je tiens à faire quelques remarques sur l'implantation elle même.

Le système PROLOG II est la suite directe de l'effort qui avait été entrepris en 1979 par A.Colmerauer, H.Kanoui et M.van Caneghem pour implanter un système Prolog sur micro- ordinateur (M6800). Ce travail nous avait permis de vérifier qu'un Prolog complet était [illegible]sable sur des machines très petites même si il y avait une mémoire virtuelle sur disque souple.

En commun avec A.Colmerauer j'ai établi ce que devait être ce nouveau Prolog. Nous avons très longuement réfléchi aux problèmes d'environement et de coroutinages.

Le modèle théorique de Prolog conçu par A. Colmerauer a été défini en même temps que l'implantation progressait, ce qui a été très bénéfique pour celle-ci. Si au début elle était en avance sur le modèle théorique, maintenant c'est le contraire et certains détails du modèle théorique ne sont pas encore implantés dans la version 1 de PROLOG II.

Avec H. Kanoui j'ai défini la machine virtuelle MI[illegible]MESAS ainsi que le langage CANDIDE qui sert à l'implantation de Prolog de manière portable. Il faut bien voir que l'effort consistant à implanter Prolog sur un Apple II est considérablement plus important que celui nécessaire à la même réalisation sur une grosse machine. C'est donc en partie grace à la grande qualité et à la fiabilité des logiciels écrits par H. Kanoui (l'interpréteur de Micromegas, la mémoire virtuelle et le compilateur Candide) que la version Apple de PROLOG II est une réussite.

Notre but avec cette nouvelle implantation est d'avoir un Prolog portable qui permette de passer des programmes plus gros et plus intelligents.

Des programmes plus gros grace à un nouvel environnement:
-les mondes pour la modularité et la gestion de mémoire.
-l'éditeur et le tampon pour l'interactivité.
-bloc pour la fiabilité.

Des programmes plus intelligents grace:
-aux arbres infinis pour avoir une structure de données plus riche tout en ayant un modèle plus juste de Prolog.
-dif et geler pour pouvoir retarder des évaluations.

Independamment de notre recherche nous avons regardé d'autres implantations de Prolog. Cela a été utile pour décider ce qui fallait ou ne fallait pas faire. Je tiens à remercier ici tous les auteurs:
M.Bruynooghe, F.McCabe, K.Clark, P.Donz, C.Mellish, F.Peirera, G.Roberts, P.Roussel, P.Szeredi, D.Warren.

La version actuelle (version 1) fonctionne depuis un an. Elle a passé avec succès le test des étudiants (maitrise et dea).

La documentation disponible est la suivante:
-Manuel d'utilisation
-Manuel de référence et modèle théorique
-Manuel d'exemples
-Implantation PROLOG APPLE II
-Manuel de référence CANDIDE.
-Manuel de référence MICROMEGAS.

Michel VAN CANEGHEM

Mars 1982

TABLE DES MATIERES

1 LANCEMENT DE PROLOG

1.1 LANCEMENT

Le système de base fonctionne avec deux disques:

1. l'un pour la mémoire virtuelle(MEM-VIR) -> drive 2
2. l'autre pour Prolog(PROLOG) -> drive 1.

Pour lancer le système il faut :

1. Mettre le courant ou faire RESET.
2. Puis faire CONTROL-E (pour passer en mode minuscule/majuscule)
3. Enfin X PROLOG et ca y est c'est parti, les deux disques doivent s'allumer un certain nombre de fois et un message doit apparaitre sur votre écran, puis le caractère ")" apparait signifiant que Prolog attend une entrée.

ATTENTION

NE PAS OUBLIER DE FAIRE CONTROL-E CAR TOUS LES NOMS DE REGLES PREDEFINIES SONT ECRITS EN MINUSCULE.

Pour sortir du système il faut écrire bonsoir; Il faut bien comprendre que puisque l'on travaille avec une mémoire virtuelle il n'existe pas d'état intermédiaire car certaines pages qui se trouvent en mémoire centrale n'ont pas forcément leur copie sur le disque. En particulier si l'on arrete le système par RESET l'état de la mémoire virtuelle est indeterminé. Si vous voulez garder un état de la mémoire virtuelle il faut le recopier sur un autre disque sous le système PASCAL. Il faut savoir également que quand on sort de l'éditeur, la mémoire virtuelle est sauvegardée.

TOUJOURS TERMINER PAR bonsoir. SI VOUS FAITES RESET OU SI VOUS COUPEZ LE COURANT VOUS RISQUEZ DE PERDRE VOTRE PROGRAMME.

1.2 ESCAPE

Cette touche sert à interrompre un programme PROLOG qui boucle, ou quelque chose qui est trop long. En réponse la machine imprime "VOUS VENEZ D'APPUYER SUR LA TOUCHE ESC". Il faut parfois appuyer plusieurs fois sur cette touche, car cette interruption est ineffective quand on est sous PASCAL.

1.3 LES DISQUES

1.3.1 PROLOG

Le disque PROLOG doit contenir les fichiers suivant:

1. PROLOG.CODE (la machine virtuelle)
2. P.KODE (l'interpreteur Prolog)
3. SYSTEM PASCAL.

La version livrée contient en plus les fichiers suivant (facultatifs):

1. SYSTEM.APPLE
2. SYSTEM.MISCINFO
3. SYSTEM.FILLER
4. SYSTEM.EDITOR

vous pouvez ajouter sur ce disque vos fichiers PROLOG source.

1.3.2 MEM-VIR

Ce disque contient 2 fichiers :

1. MV.MV support de la mémoire virtuelle(272 blocs)
2. MV.SV sauvegarde de l'état de la memoire virtuelle.

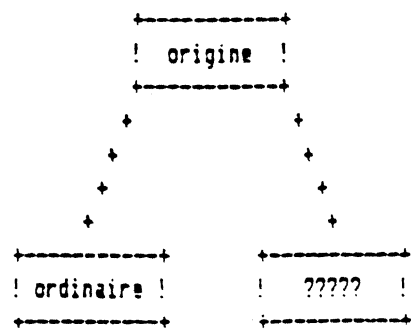
Le disque qui est livré contient une mémoire virtuelle qui est déjà initialisée avec le superviseur. C'est donc un état de départ, il donc IMPORTANT DE SAUVEGARDER CE DISQUE SUR UN AUTRE DISQUE. Chaque fois que vous voulez repartir à zéro, il faut refaire une copie de ce disque.

1.3.3 PRO-EX

Il s'agit d'un disque qui contient des exemples de programmes Prolog. Ce sont tous les programmes qui apparaissent dans le manuel d'exemples.

1.4 ORGANISATION DE LA MEMOIRE VIRTUELLE DE DEPART

La mémoire virtuelle fournie contient 3 mondes:



Au début le monde "ordinaire" est vierge. Le monde "origine" contient tous les accès aux règles prédéfinies du superviseur. Tous ces accès seront donc connus dans tous les descendants du monde "origine". Le monde "?????" qui est en fait le superviseur contient la définition des règles prédéfinies ainsi que certaines règles d'usage courant ecrites en Prolog.

Quand le système est lancé vous vous trouvez en haut du monde "ordinaire".

REMARQUE

Pour une utilisation simple il faut connaitre les commandes suivantes:

1. "neuf": Réinitialise le monde dans lequel on travaille. Commande similaire au new de Basic.
2. "tasser": Commande qui récupère la place disponible dans le monde courant.

On peut également sauver un monde sous la forme texte source (ce qui est utile car la mémoire virtuelle n'est pas d'une fiabilité parfaite) en se placant sous l'éditeur et en effectuant les commandes suivantes:

```
)editer;
+h
+i 10000"xxx.."
```

ou xxx.. représente le nom du fichier sur lequel on veut sauver le programme.

2 SYNTAXE DE PROLOG

Un programme Prolog consiste en une suite de règles ou commentaires. Chaque commentaire est une chaine. Chaque règle comprend un membre gauche (ou tete) et un membre droit (ou queue) éventuellement vide, reliés le connecteur "->".

La tete se réduit à un seul terme, la queue de clause est une suite de termes séparés par des blancs et terminé par le caractère ";".

On peut faire deux remarques:

1. La notation en n-uplet a une forme abrégée équivalente quand le premier argument est un identificateur.

   Ex: <plus,3,4> équivalent à plus(3,4).

2. La syntaxe variable/identificateur est nouvelle. Un identificateur commence par deux lettres au moins alors qu'une variable commence par une lettre au plus.

Exemple d'énoncés:

"Le célèbre conc"

```
conc(nil,y,y) ->;
conc(e.x,y,e.z) -> conc(x,y,z);
```

2.1 QUELQUES LIMITATIONS DE LA SYNTAXE

Les identificateurs doivent etre plus petits que la longueur de ligne maximale, c'est a dire 128 caractères.

Meme limitation pour les chaines de caractère.

Attention: en sortie, les identificateurs et les chaines seront tronqués à la longueur de la ligne courante.

Les entiers sont des nombres positifs plus petit que 2097151.

Le nombre de variables dans une règle doit etre inférieur à 42.

Le nombre de champs d'un n-uplet doit etre inférieur à 2097151.

Dans les règles de syntaxe hors contexte qui suivent :

Le signe de réécriture est ::= et le membre gauche d'une règle n'est pas répété lorsqu'il est identique à celui de la règle précédente.

Les terminaux sont des caractères et sont effectivement représentés par des caractères isolés, sauf le caractère d'espacement représenté par le mot espace.

Les non-terminaux sont des suites de mots entourées des signes ⟨ et ⟩. Nous prévenons le lecteur que les caractères ⟨ et ⟩ interviennent aussi en tant que symboles terminaux. Dans ce cas ils apparaissent isolement.

⟨caractère⟩
  ::= ⟨caractère spécial⟩
  ::= ⟨lettre⟩
  ::= ⟨chiffre⟩

⟨caractère spécial⟩
  ::= +
  ::= -
  ::= '
  ::= .
  ::= ,
  ::= ;
  ::= =
  ::= *
  ::= /
  ::= (
  ::=)
  ::= ⟨
  ::= ⟩
  ::= espace

⟨chaine⟩
  ::= " ⟨suite de caractères⟩ "

⟨chiffre⟩
  ::= 0
  ::= 1
  .....
  ::= 9

⟨commande⟩
  ::= ⟨suite de termes⟩ ;

⟨commentaire⟩
  ::= ⟨chaine⟩

⟨constante⟩
  ::= ⟨identificateur⟩
  ::= ⟨chaine⟩
  ::= ⟨entier⟩

⟨énonce⟩
  ::= ⟨commentaire⟩
  ::= ⟨règle⟩

⟨entier⟩
  ::= ⟨chiffre⟩ ⟨suite de chiffres⟩

⟨identificateur⟩
  ::= ⟨mot long⟩
  ::= ⟨identificateur⟩ - ⟨mot⟩

⟨lettre⟩
  ::= ⟨minuscule⟩
  ::= ⟨majuscule⟩

⟨minuscule⟩
  ::= a
  ::= b
  .....
  ::= z

⟨majuscule⟩
  ::= A
  ::= B
  .....
  ::= Z

⟨mot⟩
  ::= ⟨mot court⟩
  ::= ⟨mot long⟩

⟨mot court⟩
  ::= ⟨lettre⟩ ⟨suite de chiffres⟩
  ::= ⟨mot court⟩ '

⟨mot long⟩
  ::= ⟨lettre⟩ ⟨mot court⟩
  ::= ⟨lettre⟩ ⟨mot long⟩

⟨parasite⟩
  ::= /
  ::= ⟨syntaxe inconnue de l'utilisateur⟩

⟨programme⟩
  ::= ;
  ::= ⟨énonce⟩ ⟨programme⟩

⟨regle⟩
  ::= ⟨terme⟩ - ⟩ ⟨suite de termes⟩ ;

```
<suite de caractères>
   ::= <vide>
   ::= <caractère> <suite de caractères>

<suite de chiffres>
   ::= <vide>
   ::= <chiffre> <suite de chiffres>

<suite de termes>
   ::= <vide>
   ::= <terme> <espace> <suite de termes>
   ::= <parasite> <espace> <suite de termes>

<terme>
   ::= <terme simple>
   ::= <terme simple> . <terme>

<terme simple>
   ::= ( <terme> )
   ::= <variable>
   ::= <constante>
   ::= <>
   ::= < <terme> >
   ::= < <terme> , <terme> >
   ::= < <terme> , <terme> , <terme> >
   ..............................
   ::= <identificateur> ( <terme> )
   ::= <identificateur> ( <terme> , <terme> )
   ........................................

<variable>
   ::= <mot court>
   ::= <variable> - <mot>

<vide>
   ::=
```

Le caractère ' doit etre doublé à l'intérieur d'une chaine.

Sans altérer en quoi que ce soit le sens des choses écrites:

1. des espaces peuvent etre insérés à tout endroit, sauf à l'intérieur des constantes et des variables.

2. des espaces peuvent etre enlevés de tout endroit, sauf à l'intérieur d'une chaine et sauf si cela provoque la création de nouvelles constantes ou variables par agglutination d'anciennes.

3. REGLES PREDEFINIES

ATTENTION: TOUS LES APPELS A DES REGLES PREDEFINIES SONT ECRITS EN minuscule.

Pour s'executer chaque règle prédéfinie vérifie le type de ses arguments. Afin de faciliter la lecture on prend les conventions suivantes:

v -> variable
t -> terme
i -> identificateur
c -> chaine
e -> entier

3.1 LE CONTROLE

> /

Il s'agit du "/" qui est bien connu des utilisateurs de PROLOG. Il faut cependant remarquer que dans cette version le "/" doit etre présent dans la clause où il apparait. Il ne peut pas etre utilisé dynamiquement en remplacant une variable.

>BLOC(v,t),FIN-BLOC(t)

Il s'agit d'un moyen de terminer brutalement un programme. Ce mécanisme est principalement utilisé pour la récupération des erreurs:

Ex: bloc(u,pp(x))....

pp(x) -> ... fin-bloc(w) ...;

Bloc execute pp(x) normalement.
Si au cours de l'execution de pp(x) on rencontre fin-bloc(w) on remonte au bloc immédiatement supérieur, et si u et w sont unifiables on continue de manière déterministe après le bloc.
Sinon on remonte au bloc englobant et ainsi de suite.
Quand le processus ne se termine pas on a l'erreur "FIN-BLOC MAL TERMINE".

```
commande ->
   ligne
   exa("+")
   in-car'(k)
   bloc(x,faire(k))
   filtrer(x);

filtrer(x) -> libre(x) /;
filtrer(x) -> entier(x) / erreur(x);
filtrer(quit) -> /;
filtrer(x) -> fin-bloc(x);
```

Dans cet exemple extrait du superviseur (annexe A) on voit une utilisation de bloc. la règle "commande" execute une commande de l'éditeur. "filtre" verifie la terminaison de cette commande, quatre cas se présentent:

1. Il s'agit d'une sortie normale de bloc, la commande s'est déroulé normalement.

2. Une erreur a été rencontré au cours de l'execution de la commande, soit dans une règle prédéfinie, soit dans le système. Dans ce cas Prolog engendre fin-bloc(e), e correspondant à un entier qui indique le numéro de l'erreur.

3. Pour terminer la commande "changer" de l'éditeur on engendre fin-bloc(quit), ce qui permet de sortir d'un backtracking infini.

4. Ce dernier cas correspond a un fin-bloc apparu quelque part dans une commande (dans la commande execute par exemple) on repasse alors le controle dans le bloc englobant.

La liste des erreurs apparait dans l'annexe A. En particulier comme l'interruption ESCAPE correspond à l'erreur 16, elle peut etre reprise dans un programme. Les erreurs<=15 sont fatales. Elles ne peuvent pas etre récupérées, et elles remontent au niveau supérieur de Prolog.

3.2 GESTION DES ENONCES

En principe cette gestion est faite par l'éditeur de clauses, mais les règles prédéfinies concernées peuvent etre utilisées indépendamment. On rappelle qu'il y a un pointeur courant d'énoncés (règles ou commentaire) et que la plupart des fonctions suivantes se réfèrent à ce pointeur. On ne travaille que dans le sonde courant. On pourra trouver une description plus complète de ces commandes dans le chapitre 4.

>BAS

Le pointeur d'énoncés est positionné en bas du sonde.

>DESCENDRE(e)

On descend le pointeur courant de e énoncés, où en bas du sonde si e dépasse le nombre d'énoncé restant.

>HAUT

Le pointeur courant est positionné en haut du monde.

>MONTER(e)

On monte le pointeur courant de e énoncés, où en haut du monde si e dépasse le nombre d'enoncés restant.

>TETE(i)

On positionne le pointeur d'énoncés sur la première règle dont l'accès est l'identificateur i.

>INSERER

Insére avant le pointeur courant les énoncés lus sur l'unité active. Si il y a une erreur, on lit jusqu'au ";" et on termine l'exécution de "inserer". "inserer" se termine quand il trouve un énoncé vide, c'est à dire quand il rencontre ";;". Si l'on rencontre un EOF avant la fin de "inserer", l'insertion se termine normalement avec le message "EOF SUR LE FICHIER D'ENTREE".

>LISTER(e)

Liste sur l'unité active de sortie e énoncés à partir du pointeur courant. Cette règle ne modifie pas le pointeur courant. Il y a une présentation standard: si une règle tient sur une ligne, on l'écrit sur une ligne, sinon on écrit un terme par ligne.

>SUPPRIMER(e)

Supprime e énoncés à partir du pointeur courant. Ce pointeur est alors positionné sur le prochain énoncé non supprimé.

>RENOMMER(c1,c2)

Change le nom de l'identificateur correspondant à c1 par c2 à condition qu'il n'exite pas d'identificateur associé a c2.

3.3 LES ENTREES

Le système contient 4 unités d'entrée/sortie:

1. "tampon" entrée et sortie
2. "console" "
3. un fichier disque en entrée
4. un fichier disque en sortie

A chaque instant il y a une seule unité active en entrée et une seule unité active en sortie. Tous les transferts se font par rapport à ces deux unités actives.

La lecture se fait ligne par ligne. Dans le cas du passage "tampon" <-> "autre unité" le pointeur de caractère est conservé. Dans le cas du passage "disque" <-> "console" seul le pointeur de ligne est conservé; c'est à dire que ce qui reste sur la ligne est perdu.

Unité représente une unité d'entrée/sortie c'est à dire "tampon", "console" ou "xxx". xxx représentant un nom de fichier conforme à la syntaxe du système hote. Pour l'Apple il s'agit du nom du fichier sans le suffixe .text.

3.3.1 QUELQUES REMARQUES SUR TAMPON

L'unité "tampon" correspond à une pile de fichiers située en mémoire virtuelle. Ces fichiers "tampon" servent à toutes les manipulations qui utilisent les e/s comme ajout, univ ...

Comme à un moment donné "tampon" peut etre utilisé à la fois en entrée et en sortie, il faut faire très attention à la manière dont les ordres d'entrée/sortie se succèdent. Chaque fois que l'on passe d'un ordre d'entrée à un ordre de sortie, il faut changer de ligne (à l'aide de "ligne" par exemple).

Remarque: Certaines règles prédéfinies comme "sortie" provoquent un retour à la ligne.

Malgré toutes ces explications l'utilisation de "tampon" est assez délicate et il est conseillé de regarder dans le superviseur (annexe A) comment on se sert de "tampon".

"tampon" est utilisé dans les commandes suivantes: editer, ajout, tasser, liste-des.

Voici un programme qui utilise le tampon pour concaténer deux chaines.

```
-------------------------------------
conc(x,y,z) -> tampon-neuf(conc'(x,y,z));

conc'(x,y,z) ->
   sortie("tampon")
   exa("****")
   exa(x)
   exa(y)
   exa("****")
   ligne
   sortie("console")
   entree("tampon")
   in-chaine(z)
   entree("console");
-------------------------------------
```

3.3.2 LES REGLES PREDEFINIES POUR L'ENTREE

>ENTREE(c)

L'unité de nom c devient l'unité active d'entrée.

>TAMPON-NEUF(t)

Lance l'execution déterministe de t en créant un nouveau fichier "tampon". Quand t est effacé ou si l'on ne peut pas effacer t le fichier crée est détruit et on revient au précédent fichier "tampon".

Les règles prédéfinies suivantes lisent sur l'unité d'entrée active. Dans ce qui suit le terme caractère signifie une chaine de un caractère.

>IN-CAR(c)

Lit sur l'unité active le caractère c.

>IN-CAR'(c)

Lit sur l'unité active le prochain caractère c non blanc.

>CAR-APRES(c)

Observe le prochain caractère c sans le lire effectivement.

>CAR-APRES'(c)

Observe le prochain caractère c non blanc sans le lire effectivement. Dans l'implantation actuelle le pointeur de caractère est en fait modifié et pointe après l'execution de "car-apres'" sur le premier caractère non blanc qui suit le dernier caractère lu.

>IN(t)

Lit le plus grand terme possible. A cause de la syntaxe des termes il faut un caractère n'appartenant pas au terme pour terminer la lecture. Attention ce caractère n'est pas lu.

```
-------------------------------------
>in(t) in-car(";");
ceci.est.un.terme;
t=ceci.est.un.terme
-------------------------------------
```

>IN-ENTIER(v)

Essaye de lire sur l'unité courante un entier. Si l'objet à lire n'a pas la syntaxe d'un entier, alors rien n'est lu. Sinon le dernier caractère lu est le dernier caractère de l'entier.

>IN-IDENT(v)

Lecture d'un identificateur de la même manière que précédement.

>IN-CHAINE(v)

Lecture d'une chaine de la même manière que précédement.

>IN-PH(t)

Lit une phrase qui se termine par ".","?","!" et met la phrase sous la forme d'une séquence d'atomes terminée par nil. Tout mot, c'est à dire toute suite de lettres, est transformé dans l'identificateur at-xxx, ou xxx représente le mot lu. Tout entier, c'est à dire toute suite de chiffres, est transformé dans l'entier correspondant. Tout autre caractère est remplacé par la chaine correspondante.

```
-------------------------------------
>in-ph(x);
ceci est une phrase.
x=at-ceci.at-est.at-une.at-phrase.".".nil
>in-ph(x);
toto:=52345;.
x=at-toto.":".":=".52345.";".".".nil
-------------------------------------
```

>FIN-LIGNE(c)

ansforme le retour chariot en le caractère c. Par ;aut fin-ligne est le blanc. Dans l'éditeur de :lauses fin-ligne est le caractère ",".

3.4 SORTIES

3.4.1 SORTIES NORMALES

.es sorties se font par rapport à l'unité active de sortie.

>SORTIE(c)

L'unité de nom c devient l'unité active de sortie.

>FERMER-SORTIE

Dans le cas où l'unité active est le disque, "fermer-sortie" ferme le fichier sur disque et le rend permanent. Ceci est fait automatiquement quand on sort de l'éditeur ou de Prolog.

>EX(t)

Sort sur l'unité courante le terme t. "ex" ne coupe pas un atome en deux (si un atome est plus grand que la longueur de la ligne il est tronqué). "ex" essaye si possible de ne pas couper un terme en deux, si le terme est plus grand que la longueur de la ligne, un retour à la ligne est engendré avec une indentation de 3 blancs. Avec cette restriction tout terme écrit par "ex" peut etre lu par "in".

"ex" permet également de sortir des arbres infinis. "ti" représente un arbre égal à l'arbre ancêtre qui est à "i" niveau au dessus. Cette sortie n'est pas optimisée, c'est à dire que l'on N'a pas la forme minimale de l'arbre infini. Voici quelques exemples de sortie.

```
>eg(x,ff(x));
x=ff(t1)
>eg(x,ff(x,x));
x=ff(t1,t1)
>eg(x,x.x.x.x);
x=t1.t2.t3.t3
```

>EXM(c)

Sort sur l'unité courante la chaine c sans mettre les guillemets. La remarque précédente est toujours valable: on ne peut pas écrire de chaine plus longue que la longueur de ligne.

>LIGNE

Effectue un saut de ligne.

>PAGE

Effectue un saut de page. Sur l'unité "console", l'écran est effacé et le curseur se trouve en haut à gauche.

>LG-LIGNE(e)

L'entier e devient la nouvelle longueur de ligne. Par défaut la longueur de ligne est de 72 caractères. Dans le cas de l'APPLE il peut etre intéressant de mettre cette longueur à 40 caractères. La longueur maximale est de 128 caractères.

>EN-XY(e1,e2)

Positionne le curseur en e1,e2 sur l'écran, similaire au gotoxy de PASCAL. Pour mémoire (0,0) correspond au coin en haut et à gauche. On doit avoir: 0<= e1 <=79 et 0<= e2 <=23.

>POS(e)

Permet se positionner le pointeur courant de caractère à la position e de la ligne écrite. Il s'agit d'une espèce de tabulation. Ne marche pas en arrière du pointeur courant de caractère.

>ECHO

Permet d'obtenir l'écho sur "console" de ce qui est lu ou écrit sur un fichier disque ou "tampon". Attention quand l'unité de sortie est un fichier disque, l'écho ne permet pas de sortir ce qui est frappé sur le clavier.

>SOURD

Annule l'effet de "écho".

3.4.2 SORTIE SUR L'IMPRIMANTE

'aprisante est considérée comme un organe qui recopie tout ce qui apparait sur l'écran, à la fois les entrées et les sorties.

On peut également faire la recopie d'ecran sur un fichier. Il suffit pour cela de définir un fichier 'imprimante.text' sur votre disque prolog et dans ce cas la recopie se fera sur ce fichier qui en quelque sorte remplace l'imprimante.
ATTENTION:Quand vous utilisez cette possibilité vous ne pouvez plus vous servir du fichier en sortie sous prolog.

>PAPIER

L'imprimante imprimera tout ce qui apparait sur l'écran.

>SANS-PAPIER

Annule l'effet de "papier".

3.5 LES TESTS

Il s'agit de vérifier le type des atomes.

>LIBRE(t)

Vrai si t est une variable libre (c'est également vrai si terme est une variable libre gelée).

>PRIS(t)

Vrai si t n'est pas une variable libre.

>ENTIER(t)

Vrai si t est un entier.

>IDENT(t)

Vrai si t est un identificateur.

>CHAINE(t)

Vrai si t est une chaine.

3.6 ARITHMETIQUE

Toute l'arithmétique est faite par le biais de la règle prédéfinie "val" qui évalue une expression arithmétique. Les entiers sont des nombres compris entre 0 et 2097151 (2 exp 21 -1). Pour l'instant il n'y a que les entiers positifs, mais bientot il y aura les réels positifs et négatifs.

>VAL(t1,t2)

Evalue t1, le résultat doit etre de type atome et ajoute la contrainte valeur(t1)=t2.

Le terme évaluable est composé de fonctions évaluables, qui ne s'appliquent que sur des termes connus. Un entier, une chaine sont évalués par leur valeur. Le cas des identificateurs est plus complexe. Quand une fonction évaluable n'a pas le bon nombre d'arguments, ou quand un argument n'est pas du bon type, "val" engendre une erreur. Bien sur cette erreur est récupérable.

Un identificateur a comme valeur soit:

1. Un accès à un paquet de clause. Dans ce cas la valeur est l'identificateur lui meme.
   Ex: toto -> .....;
   valeur(toto) = toto

2. Un accès à une fonction évaluable definie par une primitive interne au superviseur.
   Ex: valeur(add) = erreur

3. Un atome. Dans ce cas la valeur a été définie par le prédicat évaluable affecter.
   Ex: affecter(toto,3);
   valeur(toto) = 3

On dispose donc d'autant de variables simples (au sens des langages de programmation classiques) que l'on veut.

Les fonctions évaluables définies actuellement sont:

1. add(t1,t2): valeur(add(t1,t2)) = valeur(t1) + valeur(t2)
2. sub(t1,t2): valeur(sub(t1,t2)) = valeur(t1) - valeur(t2)
3. mul(t1,t2): valeur(mul(t1,t2)) = valeur(t1) * valeur(t2)
4. div(t1,t2): valeur(div(t1,t2)) = valeur(t1) / valeur(t2)
5. mod(t1,t2): valeur(mod(t1,t2)) = valeur(t1) modulo valeur(t2)
6. inf(t1,t2): valeur(inf(t1,t2))= si valeur(t1) < valeur(t2) alors 1 sinon 0
7. eq(t1,t2): valeur(eq(t1,t2))= si valeur(t1) = valeur(t2) alors 1 sinon 0
8. si(t,t1,t2): valeur(si(t,t1,t2))= si valeur(t)=0 alors valeur(t1) sinon valeur(t2)

En général les arguments d'une fonction évaluable doivent etre de type entier. Cependant dans le cas de la fonction "inf" les arguments peuvent etre de type atome.
Dans le cas des entiers il s'agit il s'agit de la relation inf sur les nombres. Dans le cas des chaines il s'agit de l'ordre alphabétique, et dans le cas des identificateurs il s'agit de l'ordre alphabétique sur les chaines associées.

Pour les fonctions booléennes nous avons pris la meme convention que Basic: vrai=1, faux=0. Ceci permet de faire facilement des "et" et des "ou" sur les booléens.

>AFFECTER(i,t)

Affecte la valeur de t à l'identificateur i. A condition que:

1. La valeur du terme t soit de type atome.
2. L'identificateur ne corresponde ni à un accès à une règle, ni à une fonction évaluable.
3. Dans le cas ou t a le type chaine, il faut que cette constante ait été définie dans le meme sous-monde que l'identificateur. En particulier on ne peut pas affecter une chaine qui est lue sur le clavier à un identificateur.

Bien entendu cette affectation reste après le backtracking. Il s'agit de l'affectation traditionnelle en informatique. Il faut utiliser cette facilité avec modération car Prolog n'est pas Fortran.

>DEF-TAB(i,e)

Permet de définir un tableau d'identificateur de taille e. L'acces ou l'affectation se faisant comme dans le cas des tableaux traditionnels. A utiliser uniquement en cas de besoin vital.

```
"gestion d'une pile"

inc(i) -> val(add(i,1),x) affecter(i,x);

dec(i) -> val(sub(i,1),x) affecter(i,x);

initialiser -> affecter(pointeur,0) def-tab(pile,100);


empiler(v) ->
   inc(pointeur)
   val(inf(pointeur,100),1)
   /
   val(pointeur,p)
   affecter(pile(p),v);
empiler(v) -> exa("debordement") ligne impasse;

depiler(v) ->
   val(eq(pointeur,0),0)
   /
   val(pile(pointeur),v)
   dec(pointeur);
depiler(v) -> exa("debordement") ligne impasse;

>initialiser;

>empiler(12345);

>empiler(12346);

>depiler(x);
x=12346
>depiler(x);
x=12345
>depiler(x);
debordement
```

On peut trouver d'autres exemples de "val" et "affecter" dans l'annexe A, pour le calcul de la date par exemple.

3.7 LES MONDES

Il s'agit d'une structuration de l'espace des énoncés en une hiérarchie arborescente. Ce qu'il faut savoir c'est que d'un monde donné on a accès à tous les identificateur des mondes ancetres. Les mondes parallèles sont donc protégés les uns des autres. Dans le disque mémoire virtuelle fourni, le monde

courant a comme nom "ordinaire" et il n'a qu'un seul ancêtre le monde "origine". Ce dernier contient tous les accès aux règles définis dans le superviseur.

La définition précédente est une définition statique qui est valable pour la plupart des cas. D'un point de vue historique cela est un peu plus compliqué. Voyons cela sur un exemple:

```
>toto;
     (toto vient d'etre défini dans "ordinaire")
>descendre("mondel");
     (on se positionne dans le monde "mondel")
>inserer;
toto -> exa("toto");
     (l'identificateur toto est celui du monde
      "ordinaire". la règle toto est défini
      dans "mondel", mais accessible dans le
      monde "ordinaire")
lulu -> exa("lulu");
     (la règle lulu est uniquement accesible
      dans le monde "mondel")
;
>monter("ordinaire");
     (on revient au monde "ordinaire")
>lulu;
     (cet identificateur est différent de celui
      défini dans le monde précédent)
```

>DESCENDRE(c)

Le monde courant devient le monde de nom c.

1. Si il existe un fils du monde courant de nom c, alors ce monde devient le monde courant.

2. Si il n'existe pas de fils de nom c alors, on crée un monde fils du monde courant qui aura comme nom c et on se place dans ce monde qui devient le monde courant.

Le monde "?????" est protégé, on ne peut pas descendre dans ce monde.

>MONTER

On se place dans le monde père du monde courant. On ne peut pas monter au-dessus du monde "origine".

>TUER-MONDE(c)

Supprime le monde fils de nom c du monde courant. Il faut remarquer que c'est le seul moyen de récupère effectivement la mémoire. Quand on tue un monde on réinitialise les identificateurs qui sont des accès à des clause du monde supprimé. Actuellement on ne vérifie pas si il y a des accès à ce monde dans les piles du système.
ATTENTION:Cette règle ne peut etre utilisé que pour tuer un monde terminal, c'est a dire un monde qui n'a pas de fils. Si l'on veut détruire un ensemble de monde il faut utiliser la commande purger.

>MONDE(t)

La contrainte t="nom du monde courant" est ajoutée. Ce nom est un atome de type chaine.

>SOUS-MONDE(t)

La contrainte t="liste des sous-mondes du monde courant" est ajouté. Par exemple si l'on est dans le monde "origine":

```
>sous-monde(x)
x="?????"."ordinaire".nil
```

Pour mémoire les commandes neuf, tasser, purger, etat qui manipulent également les mondes sont définies dans le chapitre suivant.

3.8 LES COROUTINES

Il s'agit d'un moyen de retarder les évaluations sous certaines conditions. Ce mécanisme, nouveau dans Prolog, est simple à comprendre mais beaucoup plus délicat à utiliser. Nous vous conseillons de vous référer à la brochure d'exemples pour voir comment on peut l'utiliser.

>GELER(v,t)

Le but de cette règle est de retarder l'évaluation de t tant que v est inconnu. Plus précisement:

1. Si v est libre, geler est effacé et l'évaluation de t est retardé jusqu'au moment où v devient pris.

2. Si v est pris alors t s'efface normalement.

Voici l'exemple célèbre "meme-feuilles" qui vérifie si deux arbres ont les memes feuilles terminales.

L'interet de coroutiner un tel processus est de pouvoir vérifier que les deux arbres n'ont pas les memes feuilles sans avoir parcouru complètement le premier arbre.

```
-----------------------------------
meme-feuilles(a,b) ->
   feuilles(a,u)
   feuilles(b,u)
   liste(u);

feuilles(a,u) -> geler(u,feuilles'(a,u));

feuilles'(a,a.nil) -> terminal(a);
feuilles'(a.l,a.u) -> terminal(a) feuilles(l,u);
feuilles'((a.b).l,u) -> feuilles'(a.b.l,u);

liste(nil) ->;
liste(a.u) -> liste(u);

terminal(a) -> ident(a);
-----------------------------------
```

```
-----------------------------------
hors-de(x,nil) ->;
hors-de(x,a.l) -> dif(x,a) hors-de(x,l);

differents(nil) ->;
differents(x.l) -> hors-de(x,l) differents(l);

permutation(x) ->
   eg(x,x1.x2.x3.nil)
   differents(x)
   liste-de-chiffres(x);

liste-de-chiffres(nil) ->;
liste-de-chiffres(x.l) ->
   chiffre(x)
   liste-de-chiffres(l);

chiffre(1) ->;
chiffre(2) ->;
chiffre(3) ->;
-----------------------------------
```

>DIF(t1,t2)

Vérifie que t1≠t2. Si t1 et t2 ne contiennent pas de variables libres alors "dif" est évalué directement. Si l'évaluation de "dif" doit se faire en affectant des variables libres de t1 ou t2, alors cette évaluation est mise en attente jusqu'au moment ou l'une de ces variables libres sera affecté.

Cette primitive est très importante, et très utilisée dans les programmes Prolog. En particulier cela permet de se passer du "/" dans beaucoup de cas.

Dans l'exemple qui suit, on calcule des permutations en exprimant que les chiffres d'une permutation doivent etre tous différents. A noter les règles "differents" et "hors-de" qui sont d'un usage très fréquent.

3.9 DIVERS

>ARG(e,t1,t2)

Cette primitive calcule le terme ou le caractère de rang e d'un n-uplet ou d'une chaine.

1. Si t1 est une chaine alors si
   -e=0 t2:=longueur(t1)
   -e≠0 t2:="caractère de rang e de t1"

2. Si t1 est une séquence alors <entier>=1, <terme2>
   -e=1 t2:=tete(t1)
   -e=2 t2:=queue(t1)

3. Si t1 est un n-uplet alors si
   -e=0 t2:="nombre d'arguent du n-uplet"
   -e≠0 t2:="arguent de rang e du n-uplet"

>BONSOIR

Termine l'execution de PROLOG et sauve la memoire virtuelle. Il faut toujours terminer par bonsoir.

>BOUCLE

Option qui permet l'unification d'abres infinis. Cette information est sauvegardée dans la mémoire virtuelle. Par défaut boucle est faux.

>SANS-BOUCLE

Option standard: unification normale entre arbres finis. Prolog risque de boucler si jamais il a des arbres infinis à manipuler.

>TRACE

Il s'agit d'une fonction sommaire d'aide à la mise au point. Quand "trace" est activé, tous les appels à une règle sont imprimés, avec leurs arguments.

>SANS-TRACE

Supprime l'effet précedent.

>BOUM(i,c)

Fait correspondre à un identificateur la chaine qui le représente et réciproquement. Quand i est une variable on crée un nouvel identificateur. "boum" ne marche pas si les deux arguments sont des variables.

>NO-CAR(c,e)

Fait correspondre à une chaine d'un caractère c un entier e représentant son code ASCII, et réciproquement.

4. REGLES DU SUPERVISEUR

4.1 EDITER

Il s'agit d'un éditeur de texte, écrit en Prolog (voir annexe A), qui permet la manipulation des énoncés. Cet éditeur travaille en décompilant les énoncés, il ne peut etre aussi puissant que celui du système Pascal, il faudrait plutot le comparer à un éditeur Basic.

Il faut remarquer en particulier, que toute modification dans les énonces est faite par suppression, puis par recopie. Au bout d'un certain temps le monde sur lequel on travaille est rempli d'énoncés inutiles, il faut donc faire appel a la commande tasser(voir 4.3) pour nettoyer ce monde.

4.1.1 GENERALITES

L'éditeur travaille sur le monde courant. Ce monde possède un pointeur courant, qui indique sur quelle règle, ou sur quel commentaire on est positionné. Toutes les commandes se réferent à ce pointeur. Ce pointeur est toujours compris entre haut, qui est le début du monde et bas qui correspond a la fin du monde.

Chaque commande s'exprime sous la forme d'une lettre suivie éventuellement de plusieurs arguments. Si l'on veut écrire plusieurs commandes sur la meme ligne il faut les séparer par le caractère ",". Une faute de syntaxe dans une commande provoque l'écriture d'un message qui indique la syntaxe de la commande.

La console est l'unité d'entrée/sortie standard de l'éditeur. Le caractere "+" est écrit chaque fois que l'éditeur attend une commande.

La syntaxe des unités est la suivante:

```
<unité>::=          (console)
      ::= i         (imprimante)
      ::= t         (tampon)
      ::= <chaine>  (fichier de nom <chaine>)
```

4.1.2 MONTER

```
m
m <entier>
```

Cette commande déplace le pointeur courant de <entier> énoncés vers le haut, et écrit l'énoncé courant. Si il n'y a pas d'arguments la valeur 1 est prise par défaut.

```
m <chaine>
```

Cette commande change le monde courant et écrit le premièr énoncé du nouveau monde. Le nouveau monde courant est le père de l'ancien monde courant. La chaine <chaine> doit représenté le nom du monde père. Cette commande n'est pas exécuté, si le nom du monde père n'est pas le bon.

4.1.3 DESCENDRE

```
d
d <entier>
```

Cette commande déplace le pointeur courant de <entier> énoncés vers le bas, et ecrit l'énoncé courant. Si il n'y a pas d'arguments la valeur 1 est prise par défaut. Le message: ;FIN DU SOUS-MONDE xxxx, ou xxxx est le nom du monde courant, indique que le pointeur courant se trouve a la fin du monde.

```
d <chaine>
```

Cette commande change le monde courant et écrit le premièr énoncé du nouveau monde. Le nouveau monde courant devient le fils de nom <chaine> du précédent monde courant. Si ce monde existe alors on est positionné dessus, sinon un nouveau monde est crée. Le prédicat "état" donne la liste des mondes et la place disponible.

4.1.4 HAUT

```
h
```

Positionne le pointeur courant en haut du monde.

4.1.5 BAS

```
b
```

Positionne le pointeur courant en bas du monde.

4.1.6 E(N

e <identificateur>

Cette commande positionne le pointeur courant d'énoncé sur la premiere règle dont l'accès est <identificateur> et écrit la règle courante. Si il n'y a pas de règles on est positionné en bas du monde.

On rappelle que l'accès est l'identificateur le plus à gauche dans le terme.

Ex: ff(x,y) ff
<ff,x,y> ff
<ff.gg,x,y> ff
<ff(hh).gg,x,y> ff

4.1.7 L(ISTER

l <unité>
l <entier> <unité>

Cette commande écrit <entier> énoncés a partir de l'énoncé courant sur l'unité choisie. Le pointeur courant d'énoncés est positionné sur le dernier énoncé listé. Si <entier> est absent la valeur 1 est prise par défaut. Si <unité> est absent la console est prise par défaut.

Ex 1100: liste 100 énoncés sur la console.
110i: liste 10 énoncés sur l'imprimante.
1100"toto": liste 100 énoncés sur le fichier de nom "toto".

REMARQUE: Pour sauvegarder les clauses d'un monde, sur un fichier xxxxx il faut donc faire:

h
110000"xxxxx"

Il faut s'assurer avant qu'il y a assez de place sur le disque.

4.1.8 S(UPPRIMER

s
s <entier>

Cette commande supprime <entier> énoncés à partir de l'énoncé courant. Le pointeur d'énoncés courant pointe sur le premier énoncé non supprimé. Il faut quand meme remarquer que les clauses supprimées sont inaccessibles, mais pas supprimées de la mémoire.

4.1.9 C(HANGER

c
c <entier>

Cette commande est utilisée pour modifier <entier> énoncés à partir de l'énoncé courant. Ces règles vont dans un buffer et sont affichées sur l'écran, on passe alors dans le sous mode changer avec le caractère "-" comme prompt. Il faut remarquer que ce changement est fait par suppression insertion.

Dans ce sous mode les trois commandes possibles sont:c, f, q.

c <chaine1> <chaine2>
c <entier> <chaine1> <chaine2>

Remplace dans le buffer <entier> occurences de <chaine1> par <chaine2>.

f (e)

Termine le changement dans le buffer, supprime les règles modifiées et les réinsére à la meme place. ATTENTION: Si il y a eu une faute de syntaxe, la règle erroné est perdue il faudra la réecrire.

q

Termine le changement, mais ne touche pas aux règles d'origine. Cette commande est utile pour réinsérer un paquet de règles à un autre endroit, mais dans ce cas il faut faire supprimer et inserer "tampon".

4.1.10 R(ENOMMER

r <chaine1> <chaine2>

Change le nom de l'identificateur correspondant à <chaine1> par <chaine2> à condition qu'il n'existe pas d'identificateur associé à <chaine2>.

4.1.11 I(NSERER

i <unite>

Insére les énoncés lues sur l'unité d'entrée <unite> avant l'énonce courant. Si on rencontre une erreur de syntaxe, on lit jusqu'au ";" et l'insertion reprend à l'énoncé suivant. L'insertion se termine quand on rencontre un énoncé vide ce qui correspond à ";;". Si on rencontre la fin du fichier avant, l'insertion se

terwine quand aeae noraalesent.

Ex: +i "toto" insère les clauses qui se trouvent sur le fichier "toto".

4.1.12 X(EXECUTER

x (terae)

Perset de lancer l'execution de (terae) tout en restant sous le controle de l'éditeur. Mais attention les buts doivent avoir une syntaxe de terses, c'est à dire qu'ils sont séparés par des points et non par des blancs. Il faut se rappeler que sous l'éditeur l'unité d'entrée/sortie est la "console".

4.1.13 F(IN

f

Termine le mode éditeur et revient sous Prolog. A la sortie de l'éditeur la aémoire virtuelle est sauvegardée et le fichier de sortie est fermé.

4.2 AJOUT

AJOUT(<t1,t2>)

Perset d'ajouter la règle qui a t1 comme tete et t2 comme queue. La queue étant une suite de terses séparés par des points et terminée par nil

Ex: ajout(<conc(e.x,y,e.z),conc(x,y,z).nil>)

La règle est ajoutée au dessus du paquet de règles ayant le aeae nom si il existe, sinon la règle est ajoutée en bas du aonde. Cette règle prédéfinie est écrite en Prolog (voir annexe A) Il utilise l'unité "tampon" et par conséquent aodifie les affectations des unités d'entrée/sortie. "trace" ne fonctionne pas quand il y a des "ajout".

```
ajout(c) -> tampon-neuf(ajout'(c));

ajout'((t,q)) ->
   tete-bis(t)
   sortie("tampon")
   ex(t)
   exa(" -> ")
   ex(q)
   exa(";;")
   ligne
   sortie("console")
   entree("tampon")
   inserer
   entree("console");
```

4.3 MONDES

Il s'agit de commandes supplémentaires écrites en Prolog et permettant de manipuler les aondes.

>NEUF

Perset d'effacer le sous-aonde "ordinaire" ainsi que tous ses descendants et de repartir ainsi avec un disque propre. Cette commande a le aeae esprit que le new de Basic.

>TASSER

Perset de récuperer de la place dans le aonde courant. En effet quand on supprise ou quand on aodifie des énoncés, les supprassions ne sont pas réalisées effectivesent, au bout d'un certain tesps une place importante est occupée par ces ordures. Le "garbage collector" est réalisé en écrivant tous les énoncés du aonde courant sur l'unité "tampon", puis en supprisant ce aonde en le recréant vide puis enfin en relisant les énoncés sur l'unité tampon.

ATTENTION:"tasser" détruit tous les descendants du sous-aonde courant.

On peut avoir une idée de la aémoire utilisée en utilisant la commande etat.

>PURGER

Perset de supprimer le sous-aonde courant ainsi que tous ses descendants. La différence entre "purger" et "tuer-aonde" est que ce dernier détruit un aonde terminal, alors que purger détruit un arbre de aondes.

>ETAT

Donne une idée des tailles occupées par les différents mondes. Pour chaque monde de l'univers on a le nom, le nombre de granules disque utilisés, et le nombre d'octets utilisés. L'allocation étant faite par granule, un monde réserve plus de place qu'il occupe réellement. Les deux chiffres importants sont le nombre de granules restants et le nombre d'octets occupés par un monde. Le décalage du nom des mondes indique la hiérarchie.

```
-----------------------------------
TABLE DES MONDES

origine              2    3075
 ?????               8   25502
 ordinaire           2    1215

taille max d'un monde= 65536 octets

il vous reste 20 granules(4096 octets)
-----------------------------------
```

Remarque: au minimum un monde occupe 2 granules et 1215 octets(pour la table de hash-code).

>DICO

Donne la liste des identificateurs du monde courant dans l'ordre du hash-code. Attention actuellement cette commande est très lente.

4.4 DIVERS

>ES

Il s'agit tout simplement de la règle suivante:

```
eg(x,x) ->;
```

>POINTEURS

Donne l'état des différentes piles du système.

1. sumax: pile des substitutions.
2. ndmax: pile des noeuds(récursivité).
3. pcmax: pile des points de choix.
4. rtmax: pile des variables à restaurer.

```
-----------------------------------
>pointeurs;
sumax= 69
ndmax= 36
pcmax= 18
rtmax= 0
-----------------------------------
```

Remarque: sumax et ndmax sont un peu différents de la valeur réelle car étant écrit en Prolog, pointeurs utilise les piles. La différence est de quelques centaines.

>LISTE-DES(t1,l1,t2,l2)

Permet d'obtenir la liste des individus qui vérifient une certaine propriété. Cette liste est sans répétition et triée. Ce programme est écrit en Prolog et se trouve dans l'annexe A.

De manière plus précise, t1 spécifie l'individu, t2 la propriété et l2 le résultat. La liste l1 représente la liste des variables libres de t2. Quand t2 contient des variables libres, "liste-des" backtracke pour toutes les affectations possibles de ces variables.

```
-----------------------------------
homme(grand,michel) ->;
homme(grand,alain) ->;
homme(grand,henry) ->;

homme(petit,nicolas) ->;
homme(petit,julien)  ->;
homme(petit,gilles) ->;

>liste-des(x,t.nil,homme(t,x),l);

t=grand l=alain.henry.michel.nil

t=petit l=gilles.julien.nicolas.nil
-----------------------------------
```

>AUJOURD-HUI

Donne la date courante.

>DATE

Permet de rentrer une nouvelle date. Ce prédicat peut etre appelé de 4 manières:

1. date(jour): on garde l'ancien mois et l'ancienne année.
2. date(jour-de-la-semaine): calcule la date correspondant au prochain jour de la semaine.
3. date(jour,mois): on garde l'ancienne année.
4. date(jour,mois,annee).

"jour" et "annee" sont des entiers et "jour-de-la-semaine" et "mois" sont des identificateurs. Il faut remarquer que prédicat vérifie si les dates sont correctes et calcule le jour de la semaine meme pour les années bissextiles. Vous pouvez calculer de cette manière le jour de la semaine de votre date de naissance.

>date(18,juillet,1946);
jeudi 18 juillet 1946

9037

>INFORMATION

Donne des informations sur la version de Prolog II utilisée.

On affiche également l'ensemble des corrections apportées par rapport aux versions précédentes.

5 DIFFERENCE AVEC L'ANCIEN PROLOG

A la fois beaucoup et très peu: à vous de décider. Tout ce qui peut être écrit dans l'ancien Prolog peut être écrit dans le nouveau.

Du point de vue de la syntaxe, une clause qui s'écrivait avant:

```
   +a -b -c -d.
ou
   a: b.  c.  d..
```

s'écrit maintenant:

```
   a -> b c d;
```

Les variables ne sont plus précédées de cette horrible étoile:

avant CONC(*E.*X,*Y,*E.*Z)
apres conc(e.x,y,e.z)

Les opérateurs n'existent plus (sauf le point), on peut les remplacer par des n-uplets:

x op y par (x,y)

REMPLACEMENT DES PREDICATS EVALUABLES

LU	->	in-car
LUB	->	in-car'
ECRIT	->	ex
LIGNE	->	ligne
UNIV	->(1)	boum
ANCETRE	->(2)	
/	->	/
/()	->(2)	
AJOUT	->	ajout,inserer
SUPP	->	supprimer
LETTRE	->(3)	
CHIFFRE	->(3)	
PLUS	->	val.add
MOINS	->	val.sub
MULT	->	val,mul
DIV	->	val,div
RESTE	->	val,mod
INF	->	val,inf
EGALF	->(4)	
VAR	->	libre
ENT	->	in,in-entier,in-chaine,in-ident.
SORT	->	ex
SORM	->	exm
SORC	->	lister
SI	->	entree
SO	->	sortie
SAUVE	->	bonsoir

(1) Uniquement pour les identificateurs.

(2) N'existe pas.

(3) N'existe pas mais on dispose de "no-car". Quand on utilise in-ident ou in-entier on peut se passer de ces test.

(4) On peut le faire avec dif.

```
"SUPERVISEUR DU 1 JANVIER 1982"
"LANCEMENT DU SYSTEME"

go ->
   titre
   sourd
   entree("console")
   affecter(sj,vide)
   sortie("console")
   affecter(so,vide)
   monter
   descendre("ordinaire");

erreur(x) -> as-err(x,c) / exl(c);

titre ->
   en-xy(0,11)
   exa("aujourd'hui  ")
   aujourd-hui
   ligne;

"MESSAGES D'ERREUR"

as-err(2,"DEBORDEMENT DE LA PILE DES NOEUDS") ->;
as-err(3,"DEBORDEMENT DE LA PILE DES CHOIX") ->;
as-err(4,"DEBORDEMENT DES SUBSTITUTIONS") ->;
as-err(5,"DEBORDEMENT DE LA PILE DE RESTAURATION") ->;

as-err(6,"DEBORDEMENT DU TAMPON") ->;
as-err(7,"PLUS DE PLACE EN MEMOIRE VIRTUELLE") ->;
as-err(8,"ERREUR PHYSIQUE D'ENTREE/SORTIE") ->;
as-err(9,"DEBORDEMENT DES PILES DE MICROMEGAS") ->;
as-err(11,"DEBORDEMENT DU DICO DES VARIABLES") ->;
as-err(16,"VOUS VENEZ D'APPUYER SUR LA TOUCHE ESC") ->

   ;
as-err(17,"CE N'EST PAS UN ACCES A UNE CLAUSE") ->;
as-err(18,"ACCES INCORRECT") ->;
as-err(20,"NOMBRE TROP GRAND (> 2097151) ") ->;
as-err(21,"RC DANS UNE CHAINE") ->;
as-err(22,"( ) INTERDIT, OU MANQUE )") ->;
as-err(23,"PRIMAIRE INCORRECT") ->;
as-err(24,"MANQUE > ") ->;
as-err(25,"CE N'EST PAS UN SYMBOLE FONCTIONNEL") ->;
as-err(30,"COMMENTAIRE DANS UN PAQUET DE CLAUSES") ->;

as-err(31,"CLAUSE INCORRECTE") ->;
as-err(32,"CE PAQUET EXISTE AILLEURS") ->;
as-err(33,"ON NE PEUT PAS INSERER DANS CE PAQUET") ->;

as-err(34,"MAUVAISE DEFINITION DE P.E.V.") ->;
as-err(35,"PLUS DE PLACE DANS CE SOUS MONDE") ->;
as-err(36,"LITTERAL NUMERIQUE INTERDIT") ->;
as-err(37,"TERMINAL INCORRECT") ->;
as-err(38,"CONDITION INCORECTE") ->;
as-err(40,"ON NE PEUT PAS FERMER UN FICHIER ACTIF") ->

   ;
as-err(41,"ON LIT PLUS QUE PREVU DANS LE TAMPON") ->;

as-err(50,"EOF SUR LE FICHIER D'ENTREE") ->;
as-err(51,"FICHIER INEXISTANT") ->;
as-err(52,"PLUS DE PLACE SUR LE DISQUE") ->;
as-err(53,"DISQUE PROTEGE EN ECRITURE") ->;
as-err(54,"I/O ERROR") ->;
as-err(60,"FIN-BLOC MAL TERMINE") ->;
as-err(70,"TERME INCORRECT DANS EVALUER") ->;
as-err(71,"MAUVAIS NOMBRE D'ARGUMENTS") ->;
as-err(72,"FONCTION EVALUABLE INCORRECTE") ->;
as-err(73,"ERREUR ARITHMETIQUE") ->;
as-err(74,"AFFECTATION INTERDITE") ->;
as-err(75,"ON NE PEUT PAS AFFECTER CETTE CHAINE") ->;
as-err(80,"PLUS DE PLACE POUR ALLOUER LES S.M.") ->;
as-err(100,"UNITE INCONNUE") ->;
as-err(x,"ERREUR INCONNUE") ->;

eg(x,x) ->;

toujours(p) -> encore p impasse;

encore ->;
encore -> encore;

"EDITEUR PROLOG"

editer ->
   page
   initialiser
   echo
   fin-ligne(",")
   bloc(editeur,tampon-neuf(toujours(commande)))
   fin-ligne(" ")
   sauve
   sourd;

initialiser ->
   entree("console")
   sortie("console")
   exl("a(ont d(esc h(aut b(as a(n l(ster x(cut")
   exl("s(pprimer c(hnger r(nommer i(nsrer f(in");

"LECTURE ET EXECUTION DE CHAQUE COMMANDE"

commande ->
   ligne
   exa("+")
   in-car'(k)
   bloc(x,faire(k))
   /
   filtrer(x);

filtrer(x) -> libre(x) /;
filtrer(x) -> entier(x) / erreur(x) fin-ligne(",");
filtrer(quit) -> /;
filtrer(x) -> fin-bloc(x);

faire("a") ->
   in-chaine(x)
   in-car'(",")
```

9037

```
      monter(x)
      haut
      lister(1);
faire("s") -> in-combien(x) in-car'(",") monter(x)
      lister(1);
faire("d") ->
      in-chaine(x)
      in-car'(",")
      descendre(x)
      haut
      lister(1);
faire("d") -> in-combien(x) in-car'(",") descendre(x)
      lister(1);
faire("e") ->
      in-ident(x)
      in-car'(",")
      tete-bis(x)
      lister(1);
faire("l") ->
      in-combien(n)
      ex-peripherique(p)
      in-car'(",")
      lister(n)
      descendre(n)
      monter(1)
      fin-lister(p);
faire("h") -> in-car'(",") haut lister(1);
faire("b") -> in-car'(",") bas lister(1);
faire("s") ->
      in-combien(n)
      in-car'(",")
      supprimer(n)
      lister(1);
faire("c") ->
      in-combien(n)
      in-car'(",")
      sourd
      effacer-tampon
      sortie("tampon")
      lister(n)
      sortie("console")
      changement
      supprimer(n)
      echo
      inserer-continu
      fin-ligne(",")
      entree("console")
      monter(1);
faire("r") ->
      in-chaine(x)
      in-chaine(y)
      in-car'(",")
      renommer(x,y)
      page
      lister(1);
faire("l") ->
      in-peripherique(p)
      in-car'(",")
      entree(p)
      fin-ligne(" ")
      inserer-bis(p)
      fin-ligne(",")
      monter(1)
      entree("console");
faire("x") ->
      in(x)
      in-car'(k)
      ponctuation(k)
      execute(x);
faire("f") -> in-car'(",") fi-sortie("")
      fin-bloc(editeur);
faire(",") ->;
faire(k) -> encore in-car'(",") renseigner-sur(k);

execute(x) -> x ligne impasse;
execute(x) ->;

ponctuation(",") -> /;
ponctuation(";") ->;

tete-bis(c) -> tete(c) /;
tete-bis(c) -> bas;

inserer-bis("tampon") ->
      sortie("tampon")
      exm(";")
      sortie("console")
      /
      inserer-continu;
inserer-bis(x) -> inserer-continu;

inserer-continu ->
      encore
      bloc(x,inserer)
      fin-inserer(x)
      /;

fin-inserer(x) -> libre(x) /;
fin-inserer(50) -> entree("console") /;
fin-inserer(x) ->
      entier(x)
      val(add(inf(x,17),inf(36,x)),0)
      /
      as-err(x,c)
      ligne
      exm(c)
      ligne
      /
      impasse;
fin-inserer(x) ->
      entree("console")
      as-err(x,c)
      ligne
      exm(c)
      ligne;

in-combien(n) -> in-entier(n) /;
in-combien(1) ->;
```

```
in-peripherique("console") -> car-apres'(",") /;
   peripherique("tampon") -> car-apres'("t") /
   in-car'("t");
in-peripherique(y) -> in-chaine(y) car-apres'(",");

ex-peripherique("console") -> car-apres'(",") /
   sortie("console");
ex-peripherique("imprimante") ->
   car-apres'("i")
   in-car'("i")
   /
   papier;
ex-peripherique("tampon") ->
   car-apres'("t")
   /
   in-car'("t")
   ex-bis(0);
ex-peripherique("disque") -> in-chaine(y)
   car-apres'(",") sortie(y);

in-lister("imprimante") -> / ligne sans-papier;
fin-lister(x) -> sortie("console");

changement ->
   affichage
   bloc(changer,toujours(commande-bis))
   /;

affichage ->
   page
   exl("c(hange (n) ""x"" ""y"" ,f(in ,q(uit")
   ligne
   transfert-tampon
   en-xy(0,22);

commande-bis ->
   exm("-                              ")
   en-xy(1,22)
   in-car'(k)
   faire-bis(k)
   /;

faire-bis("c") ->
   in-combien(n)
   in-chaine(x)
   in-chaine(y)
   in-car'(",")
   /
   changer-dans-tampon(n,x,y)
   affichage;
faire-bis("c") ->
   exl("Commande incorrecte")
   encore
   in-car'(";")
   en-xy(0,22)
   commande-bis;
faire-bis("f") ->
   in-car'(",")
   entree("tampon")
   fin-ligne(" ")
   sortie("tampon")
   exm(";")
   sortie("console")
   fin-bloc(changer);
faire-bis("q") ->
   in-car'(",")
   echo
   fin-bloc(quit);
faire-bis(",") ->;
faire-bis(x) -> affichage;

"MESSAGES CORRESPONDANT A DES ERREURS"

renseigner-sur("a") ->
   exl("a(jouter de un enonce)")
   exl("a(jouter de) entier (enonces)")
   exm("a(jouter dans sur-monde) chaine");
renseigner-sur("d") ->
   exl("d(escendre de un enonce)")
   exl("d(escendre de) entier (enonces)")
   exm("d(escendre dans sous-monde ) chaine");
renseigner-sur("e") ->
   exm("e(n premier enonce de) identificateur");
renseigner-sur("l") ->
   exl("l(ister un enonce sur) unite")
   exl("l(ister) entier (enonces sur) unite")
   exl("unite: (console),i(mprimante),t(ampon),")
   exm("(disque) chaine");
renseigner-sur("s") ->
   exl("s(upprimer un enonce) ")
   exm("s(upprimer ) entier (enonces)");
renseigner-sur("c") ->
   exl("c(hanger un enonce) ")
   exm("c(hanger) entier (enonces)");
renseigner-sur("r") ->
   exl("r(enommer ident) chaine ")
   exm("(par ident tout-nouveau) chaine");
renseigner-sur("i") ->
   exl("i(nserer sur) unite")
   exl("unite: (console),i(mprimante),t(ampon),")
   exm("(disque) chaine");
renseigner-sur("x") ->
   exl("x(ecuter) terme")
   exm("exemple: x 11.12.13 (return)");
renseigner-sur(k) -> initialiser;

"PREDICATS DU SUPERVISEUR"
"AJOUT"

ajout(c) -> tampon-neuf(ajout-bis(c));

ajout-bis((t,q)) ->
   tete-bis(t)
   ex-bis(0)
   ex(t)
   exm(" -> ")
   ex(q)
   exl(";;")
```

```
    ex-bis(1)
    in-bis(0)
    inserer
    in-bis(1);

"GESTION DES MONDES"

neuf ->
    purger
    descendre("ordinaire")
    page
    exl("le sous monde ""ordinaire"" est vide")
    go;

purger ->
    sous-mondes(l)
    balaye(l)
    monde(x)
    monter
    tuer-monde(x);

balaye(nil) -> /;
balaye(a.b) ->
    descendre(a)
    purger
    balaye(b);

tasser ->
    monde(m)
    echo
    haut
    tampon-neuf(krunch-bis(m))
    sourd
    /;
tasser -> sourd;

krunch-bis(m) ->
    sortie("tampon")
    lister(10000)
    sortie("console")
    monter
    tuer-monde(m)
    descendre(m)
    entree("tampon")
    inserer
    entree("console");

"AUTRES DEFINITIONS"

liste-des(f,v,p,l) ->
    tampon-neuf(set-of(f,p,v,l1))
    arranger(l1,l2)
    /
    dans(<v,l3>,l2)
    lineariser(l3,l);

arranger(nil,l) -> / niler(l);
arranger(<a,u>.l,l') -> dans(<a,v>,l') insere(u,v)
    arranger(l,l');

set-of(f,p,l,l') ->
    p
    instancie(l)
    sortie("tampon")
    ex(<l,f>)
    exm(".")
    impasse;
set-of(f,p,l,l') ->
    exl("nil;")
    sortie("console")
    entree("tampon")
    in(l')
    entree("console");

instancie(nil) -> /;
instancie(a.l) -> pris(a) instancie(l);

dans(x,x.l) ->;
dans(x,y.l) -> dans(x,l);

insere(a,<a,g,d>) -> /;
insere(a,<x,g,d>) -> inferieur(a,x) / insere(a,g);
insere(a,<x,g,d>) -> insere(a,d);

inferieur(a.b,a.c) -> / inferieur(b,c);
inferieur(a.b,c.d) -> / inferieur(a,c);
inferieur(a,b) -> val(inf(a,b),1);

lineariser(l,l') -> lineariser'(l,nil,l'') eq(l',l'');


lineariser'(a,x,x) -> libre(a) /;
lineariser'(<a,g,d>,x1,x3) -> lineariser'(d,x1,x2)
    lineariser'(g,a.x2,x3);

niler(nil) -> /;
niler(a.l) -> niler(l);

"CALCUL DE LA DATE"

aujourd-hui ->
    val(jour',j)
    val(date',d)
    val(mois',i)
    val(annee',a)
    semaine(j,s)
    mois(i,a,p,d')
    /
    exm(s)
    exm(" ")
    ex(d)
    exm(" ")
    exm(m)
    exm(" ")
    ex(a)
    ligne;

date(s) ->
    boum(s,s')
```

```
   semaine(i,s')
   /
     (add(si(inf(jour',i),sub(i,jour'),sub(add(i,7),
   jour')),date'),d)
   demain(d)
   affecter(jour',i)
   aujourd-hui;
date(j) ->
   valide-date(j)
   /
   affecter(date',j)
   calculer-jour
   aujourd-hui;
date(j) -> exl("CE MOIS A MOINS DE JOUR");
date(j,a) ->
   bous(a,a')
   mois(i,a',p,d)
   /
   affecter(mois',i)
   date(j);
 te(j,a) -> exl("JE NE CONNAIS PAS CE MOIS");
date(j,a,a) ->
   val(add(inf(a,2000),inf(1900,a)),2)
   /
   affecter(annee',a)
   date(j,a);
date(j,a,a) -> exl("UNIQUEMENT LE XX SIECLE");

valide-date(j) ->
   val(mois',2)
   val(mod(annee',4),0)
   /
   val(add(eq(j,0),inf(29,j)),0);
valide-date(j) ->
   val(mois',i)
   mois(i,a,p,d)
   val(add(eq(j,0),inf(d,j)),0);

calculer-jour ->
   val(mois',i)
   mois(i,a,p,d)
   val(add(add(p,3),mul(eq(mod(annee',4),0),inf(2,i)))

   ,p')
   val(mod(add(add(annee',div(sub(annee',1),4)),add(p'

   ,date')),7),j)
   affecter(jour',j);

demain(d) -> valide-date(d) / affecter(date',d);
demain(d) ->
   val(mois',12)
   /
   val(add(annee',1),a)
   affecter(annee',a)
   affecter(mois',1)
   val(sub(d,31),d')
   affecter(date',d');
demain(d) -> valide-date(31) / demain-bis(d,31);
demain(d) -> valide-date(30) / demain-bis(d,30);
demain(d) -> valide-date(29) / demain-bis(d,29);
demain(d) -> valide-date(28) / demain-bis(d,28);

demain-bis(d,n) ->
   val(sub(d,n),d')
   affecter(date',d')
   val(sub(mois',1),i)
   affecter(mois',i);

semaine(1,"lundi") ->;
semaine(2,"mardi") ->;
semaine(3,"mercredi") ->;
semaine(4,"jeudi") ->;
semaine(5,"vendredi") ->;
semaine(6,"samedi") ->;
semaine(0,"dimanche") ->;

mois(1,"janvier",0,31) ->;
mois(2,"fevrier",31,28) ->;
mois(3,"mars",59,31) ->;
mois(4,"avril",90,30) ->;
mois(5,"mai",120,31) ->;
mois(6,"juin",151,30) ->;
mois(7,"juillet",181,31) ->;
mois(8,"aout",212,31) ->;
mois(9,"septembre",243,30) ->;
mois(10,"octobre",273,31) ->;
mois(11,"novembre",304,30) ->;
mois(12,"decembre",334,31) ->;

information ->
   page
   exl("       INFORMATION du  1/3/1982")
   ligne
   exl("-dans l'editeur: i""fichier"" marche")
   exl("                 11000t marche")
   exl("-dans date les mois et les jours sont")
   exl(" ecrits sans guillemets")
   exl("-neuf a ete modifie")
   exl("-les annees bissextiles ok!")
   exl("-esc dans inserer marche")
   exl("-renommer marche")
   exl("-in-oh a ete modifie: les mots")
   exl(" sont prefixes par at-")
   exl("-papier peut sortir sur fichier")
   exl(" imprimante.text")
   exl("-arg marche pour les chaines");
```

-A-	
add(e,e)	val(add(3,4),7);
affecter(i,a)	affecter(aa,3);
ajout(<t,t>)	ajout(<vrai(x),nil>);
arg(e,t,t)	arg(2,<aa,bb>,bb);
aujourd-hui	
-B-	
bas	
bloc(v,t)	bloc(x,editer);
bonsoir	
boucle	
boua(i,c)	boua(toto,"toto");
-C-	
car-apres(c)	car-apres(",",");
car-apres'(c)	car-apres'(",",");
chaine(t)	chaine("toto");
-D-	
date(e)	date(21);
(i)	date(lundi);
(e,i)	date(1,novembre);
(e,i,e)	date(1,janvier,1982);
def-tab(i,e)	def-tab(pile,100);
descendre(e)	descendre(27);
(c)	descendre("ordinaire");
dico	
dif(t,t)	dif(toto,lulu);
div(e,e)	val(div(8,2),4);
-E-	
echo	
editer	
eq(t,t)	eq(x,toto);
entier(t)	entier(2056489);
entree(c)	entree("console");
en-xy(e,e)	en-xy(5,5);
eq(e,e)	val(eq(3,3),1);
etat	
ex(t)	ex(a.b);
exa(c)	exa("bonjour");
-FSH-	
fermer-sortie	
fin-bloc(t)	fin-bloc(erreur);
fin-ligne(c)	fin-ligne(";");
geler(v,t)	geler(x,liste(x));
haut	
-I-	
ident(t)	ident(toto);
in(t)	in(x);
in-car(c)	in-car(c);
in-car'(c)	in-car'(c);
in-chaine(c)	in-chaine(c);
in-entier(e)	in-entier(e);
in-ident(i)	in-ident(i);
in-ph(l)	in-ph(l);
inf(a,a)	val(inf("to","zou"),1);
information	
inserer	
-L-	
lg-ligne(e)	lg-ligne(40);
libre(t)	libre(x);
ligne	
liste-des(v,l,t,l)	liste-des(x,nil,pp(x),1);
lister(e)	lister(10000);
-MN-	
mod(e,e)	val(mod(7,3),1);
monter(e)	monter(10);
(c)	monter("origine");
monde(c)	monde(x);
ms-err(e,c)	ms-err(27,x);
mul(e,e)	val(mul(2,3),6);
neuf	
no-car(c,e)	no-car("1",49);
-P-	
page	
papier	
pointeurs	
pos(e)	pos(27);
pris(t)	pris(x);
purger	
-RS-	
renommer(c,c)	renommer("to","lulu");
sans-boucle	
sans-papier	
sans-trace	
si(e,t,t)	val(si(eq(3,4),1,2),2);
sortie(c)	sortie("resultat");
sourd	
sous-mondes(l)	sous-mondes(1);
sub(e,e)	val(sub(8,6),2);
supprimer(e)	supprimer(1);
-TV-	
tampon-neuf(t)	tampon-neuf(pp(x));
tasser	
tete(i)	tete(toto);
trace	
tuer-monde(c)	tuer-monde("toto");
val(t,e)	val(3,3);