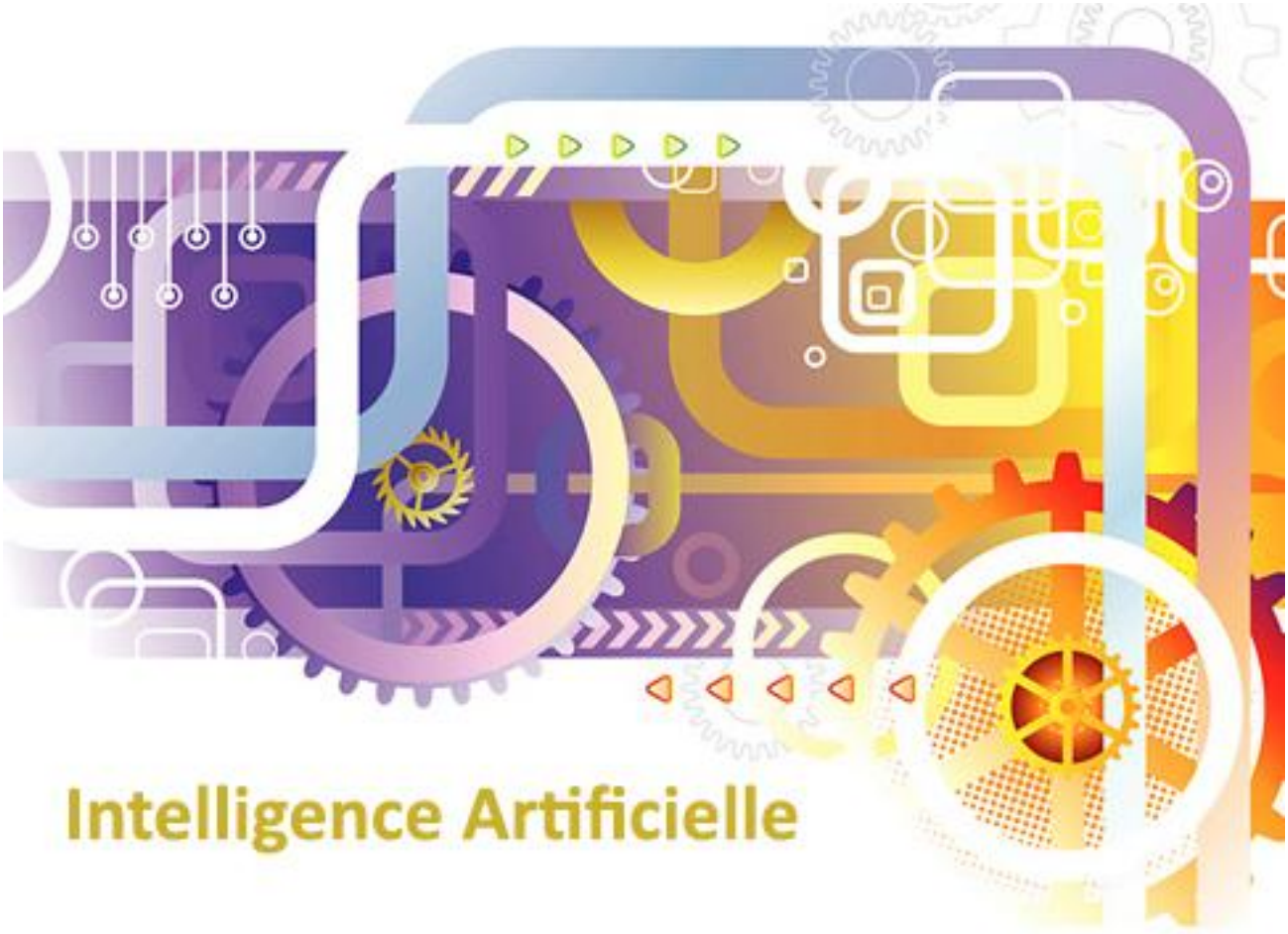
DAUPHINE
UNIVERSITÉ PARIS



Moteur d'inférence en LISP

Ordre 0 et 0+

Table des matières

I. Présentation des bases de règles

Afin de tester le fonctionnement de notre moteur d'inférence nous avons utilisé trois bases de règles d'ordre 0 et 0+.

a) Base de Winston

La base de règles de Winston est une base regroupant les caractéristiques des animaux permettant, à partir de spécificités, de déduire l'animal recherché. Cette base est la base par défaut et nous a été donné avec le sujet du projet. Nous utiliserons cette base pour le moteur d'inférence en ordre 0.

b) Base de règle cantine

Cette base de règles permet de suggérer la composition d'un plat en fonction des envies de l'utilisateur. Nous utiliserons cette base pour le moteur d'inférence en ordre 0.

c) Base de règle d'achat de voiture (ordre 0+)

Cette base de règle décrit le type de voiture optimale en fonction de la situation familiale et professionnelle de l'utilisateur. Nous utiliserons cette base pour le moteur d'inférence en ordre 0+.

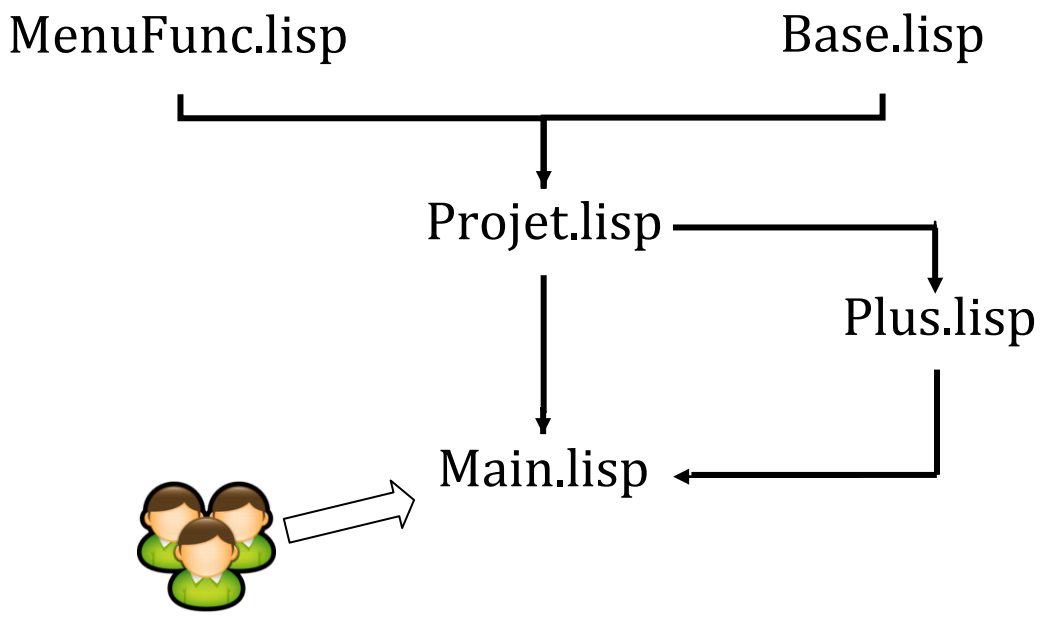
II. Travail effectué

Dans cette partie nous allons vous présenter le travail réalisé.

a) Architecture générale

Notre projet est réparti en cinq fichiers :

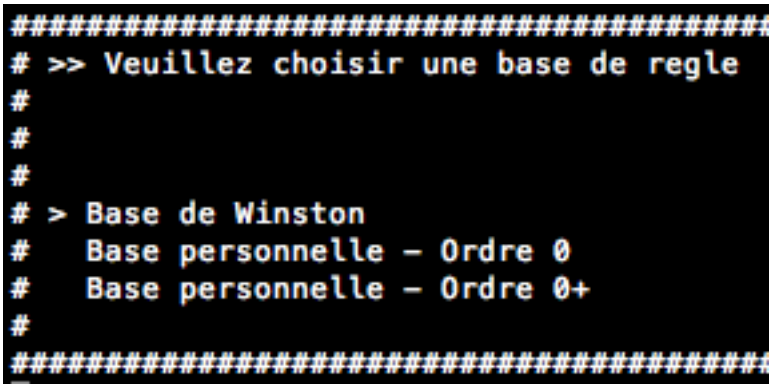
- **base.lisp** : regroupe les différentes bases de règles.
- **menuFunc.lisp** : regroupe les différentes fonctions relatives à l'interface (affichage et gestion du clavier).
- **projet.lisp** : contient toutes les fonctions nécessaires au moteur d'inférence.
- **main.lisp** : démarrage de l'application. Ce fichier a pour rôle d'initialiser, paramétrer et lancer le moteur d'inférence.
- **plus.lisp** : fonctions relatives au moteur d'inférence 0+.



b) Déroulement de l'application

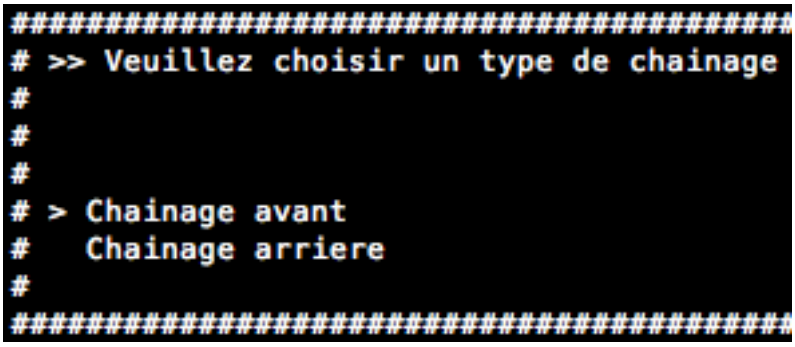
L'utilisation de l'application s'étend sur 6 étapes :

1. Choix de la base de règle



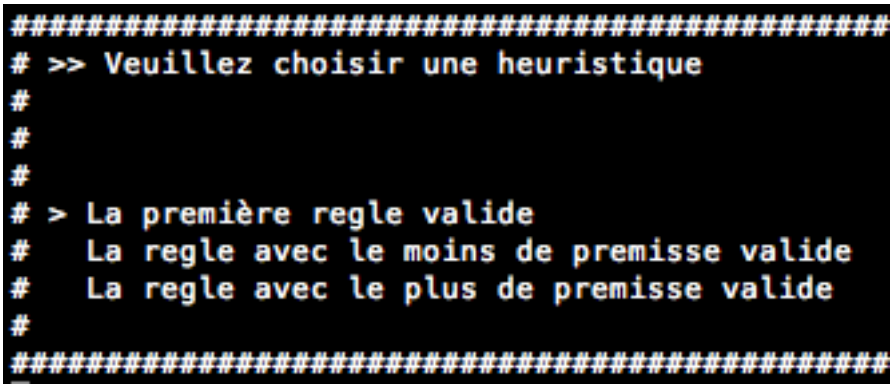
   L'utilisateur choisi la base de règles sur laquelle il veut inférer.

2. Choix du type de chainage



   L'utilisateur choisi le type de chainage (avant ou arrière).

3. Choix de l'heuristique (chainage avant)

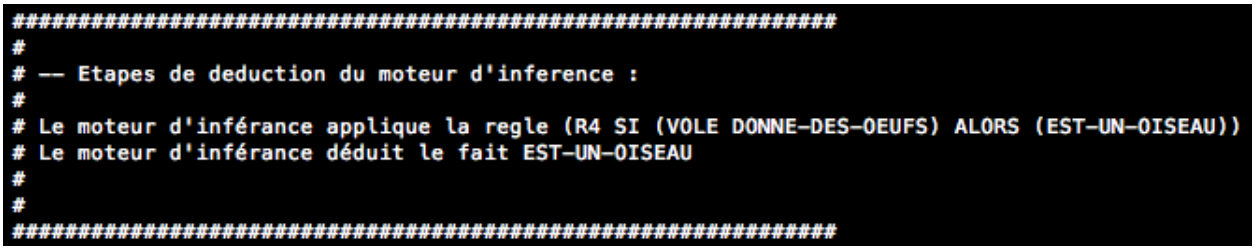


Si l'utilisateur a choisi un chainage avant alors il choisi également le type d'heuristique qu'il veut appliquer. Soit le moteur donne la priorité à la première règle qu'il trouve, soit il donne la priorité à la règle qui a le moins de prémisses parmi les règles qui sont applicables, soit il donne la priorité à la règle qui a le plus de prémisses parmi les règles qui sont applicables.

4. Choix des faits

Si l'utilisateur a choisi un chainage avant alors il choisi un ou plusieurs faits sinon pour un chainage arrière il ne choisi qu'un seul fait.

5. Trace du moteur d'inférence



Exemple d'une trace après choix des fait « donne-des-œufs » et « vole » en chainage avant.

6. Affichage de la déduction et message de fin

Exemple d'une conclusion après choix des fait « donne-des-œufs » et « vole » en chainage avant.

A noter que lors de chacune des 5 étapes l'interface a un header qui prend la forme ci-dessous :

Ainsi à tout moment l'utilisateur peut quitter le programme en appuyant sur la touche « Echap » de son clavier. Par ailleurs, les touches de navigation, de sélection et de validation lui sont continuellement indiquées.

c) Interface

L'interface permet à l'utilisateur d'interagir avec la machine. Elle est l'identité visuelle du programme, c'est la seule chose que verra l'utilisateur. C'est pour cela qu'elle doit être la plus agréable possible et facile d'utilisation. C'est dans cette optique que nous avons développé une interface fonctionnant avec les flèches du clavier.

1. Architecture

L'interface homme-machine est séparée en deux grandes parties :

- La gestion du clavier
- L'affichage

La gestion du clavier est gérée principalement dans le fichier *main.lisp*. L'affichage lui est présent tout au long des étapes du moteur. Nous allons vous présenter ces deux aspects plus en détails dans les parties suivantes.

2. Capture et gestion du clavier

La capture par défaut du clavier se fait par (**setq** touche (read)).

Cette fonction a pour inconvénient de ne prendre que les touches alphanumériques. C'est-à-dire que toutes les touches d'interaction (Entrée, supprimer, echap, flèches…) ne fonctionnent pas avec cette méthode.

Pour régler ce problème nous avons utilisé une macro intégrée à l'interpréteur Common Lisp :

```
(ext:with-keyboard
    ;code)
```

Tous les appels qui seront effectués au sein de cette macro auront accès aux touches spéciales du clavier. Cette macro nous permet ainsi de créer notre propre fonction de saisie :

```
(defun lire-clavier ()
        (ext:with-keyboard
                (setq input (read-char ext:*keyboard-input*))
                (setq key (OR (ext:char-key input) (character input)))
        )
        (return-from lire-clavier key)
)
```

Le code (read-char ext:*keyboard-input*) permet de récupérer la touche entrée comme le ferait (read) mais avec le support des touches spéciales en plus. Cependant la touche n'est pas directement utilisable, il faut avant tout la traduire dans un format plus simple et utilisable dans notre code.

Nous avons réalisé cette opération avec le code (OR (ext:char-key input) (character input))

Voici un petit tableau des valeurs retournés par la fonction lire-clavier() :

Nom touche	Valeur
Entrée	#\Return
Echap	#\Escape
Espace	#\Space
Flèche du haut	:UP
Flèche du bas	:DOWN
Flèche de gauche	:LEFT
Flèche de droite	:RIGHT

Nous avons donc utilisé ces fonctions pour créer des menus comportant un curseur. Il existe deux types de menu : le menu avec une seule colonne et celui avec plusieurs colonnes.

Voici un extrait de la boucle principale qui gère ces différents menus :

```
(loop while ( /= continuer 0) do
  (setq toucheClavier (lire-clavier))

  ;Test quelle touche du clavier a ÈtÈ pressÈe
  (cond
    ((equal toucheClavier :LEFT)
      (if (equal modeSaisie 1)
        (setq curseur (mod (- curseur 1) (length listeFaits)))
      )
    )
    ((equal toucheClavier :RIGHT)
      (if (equal modeSaisie 1)
```

```
            (setq curseur (mod (+ curseur 1) (length listeFaits)))
        )
    )
  )
)
```

3. L'affichage avec la fonction format

La fonction *format* permet d'afficher du texte formaté à l'écran. Contrairement à la famille des fonctions *print*, *format* gère les conditions, les paramètres, les sauts à la ligne et bien d'autres fonctionnalités. Nous avons choisis d'utiliser cette fonction pour sa puissance d'action et sa facilité d'implémentation.

Voici un exemple d'utilisation de conditions avec *format* :

```
(format t "# ~:[ ~;>~] ~A~%" (eq curseur i) (nth i liste))
```

Zoom sur la condition :

```
~:[ ~;>~]
```

La condition est formulée de la manière suivante : Si (curseur = i) alors afficher « > » sinon afficher un espace. Ce qui a pour résultat d'afficher ou non le curseur en fonction des touches pressées par l'utilisateur.

4. Tampon et présentation des résultats

Dans un souci d'optimisation nous avons décidé de ne pas polluer l'interaction de l'utilisateur avec le moteur par les détails de déduction de celui-ci. Pour ce faire nous avons mis en place un tampon qui regroupe l'ensemble des détails générés au fur et à mesure de l'inférence pour les afficher à la fin du raisonnement. Cela nous permet également de garder une trace claire et concise de tout le chainage. Dans l'hypothétique cas ou un technicien devrait intervenir pour régler un problème sur le moteur d'inférence son travail s'en trouverait facilité.

Voici le code qui permet d'ajouter une entrée au tampon :

```
(setq buffer_inference (make-string-output-stream))
…
(format buffer_inference "# Le moteur d'infÈrance essaye la regle ~S~%" r)
…
(format t (get-output-stream-string buffer_inference))
```

(**setq** buffer_inference (make-string-output-stream)) permet de définir la variable buffer_inference comme un flux de sortie.

(format buffer_inference "# Le moteur d'infÈrance essaye la regle ~S~%" r) permet d'écrire dans le tampon précédemment définit au lieu de la sortie standard.

format t (get-output-stream-string buffer_inference)) permet d'afficher sur la sortie standard le tampon.

d) Modifications apportées au code initial

1. Nom des fonctions

Les fonctions fournies initialement ont été renommées en supprimant d'abord les points d'interrogations et les deux-points mais aussi en standardisant leur nom.

En effet :

- toute fonction qui retourne une donnée autre que de type booléenne commencera par « get_ » suivi du nom de la propriété qu'elle renvoie
- toute fonction qui fait appel à la fonction « putprop (p) » commencera par « set_ » suivi du nom de la propriété qu'elle modifie
- toute fonction qui retourne un résultat de type booléen (« t » ou « nil ») commencera par « is_ »

D'autres fonctions ont également été renommées quand il a été jugé que cela aide à la compréhension du fonctionnement de la fonction.

2. Contenu des fonctions

Dans le contenue des fonctions existantes, certaines parcelles de codes ont été modifiées structurellement et parfois certains tests ont également été ajoutés.

- Structure des conditions : les conditions de types « cond » contenant deux tests uniquement er dont le deuxième test est le cas « t » ont été remplacées par des conditions de type « if ».

- Ajout de tests supplémentaires : dans la fonction compile_regle (r) l'instruction « if (member p propositions) » a été ajoutée afin de n'ajouter une règles dans la liste des propositions que si cette dernière ne s'y trouve pas déjà. Ainsi les doublons sont évités et donc les inférences sont plus rapides.

```
(if (member p propositions) propositions
        (setq propositions (cons p propositions))) ; else
```

e) Fonctions ajoutées

Plusieurs fonctions ont été ajoutées dans le fichier « *projet.lisp* » aux foncions déjà fournies afin de faciliter la gestion du chainage avant / arrière :

- putprop (p) *:* permet d'affecter une valeur à une propriété d'un atome en utilisant la fonction native CLisp « setf »
- rafraichir_moteur_inference () : permet de complètement rafraichir le moteur d'inférence en appelant la fonction « rafraichir_prop (p) » sur chaque proposition de la liste des propositions et la fonction rafraichir_regle (r) sur chaque règle de la base de règles
- prioriteMoinsDePremisses (regles) : permet de trier la base de règles en classant les règles par nombre de prémisses croissant, de la règle contenant le moins de prémisse à la règle contenant le plus de prémisses
- prioritePlusDePremisses (regles) : permet de trier la base de règles en classant les règles par nombre de prémisses décroissant, de la règle contenant le plus de prémisse à la règle contenant le moins de prémisses

```
; donner la priorite aux regles ayant le plus de premisses
(defun prioritePlusDePremisses (regles)
  (sort regles
    (lambda (a b)
      (> (get_nombre_de_premisses a) (get_nombre_de_premisses b))
    )
  )
)
```

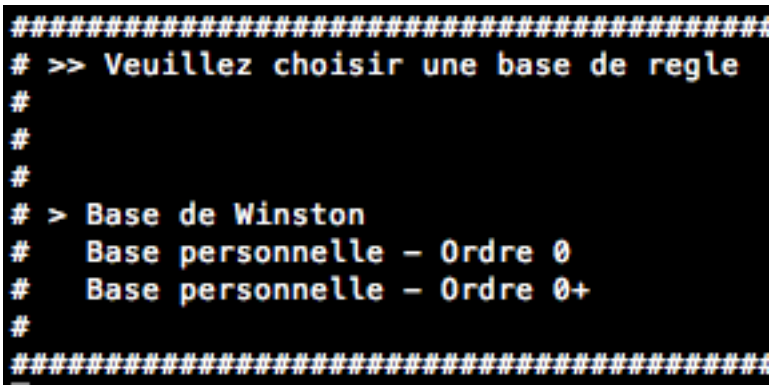
Les fonctions présentées ci-dessus sont relatives au moteur d'inférence d'ordre 0. Nous avons également ajouté une grande quantité de fonctions pour la gestion du menu et le moteur d'ordre 0+. Le code de ces fonctions est disponible en annexe.

III. Tests du moteur d'inférence

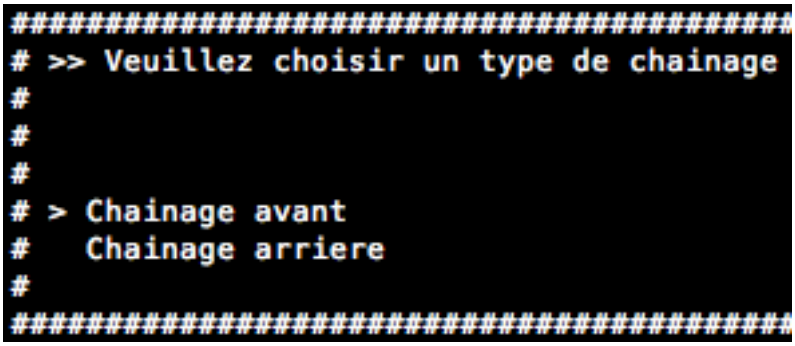
a) Tests sur la base de Winston

1) Exemple d'un chainage avant

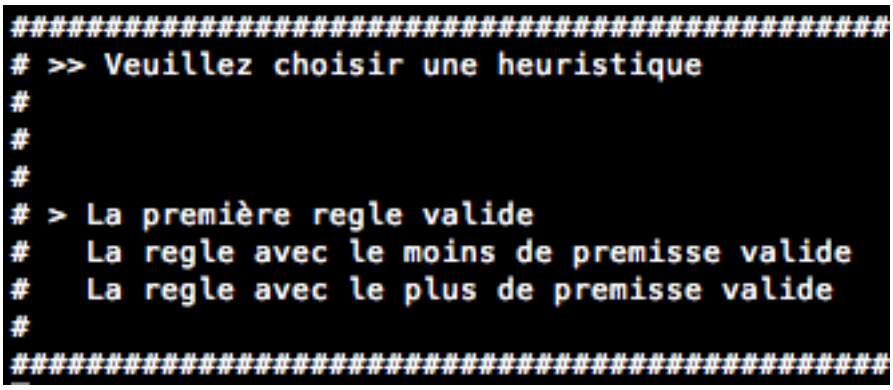
On choisi la base de Winston :



On choisi le chainage avant :

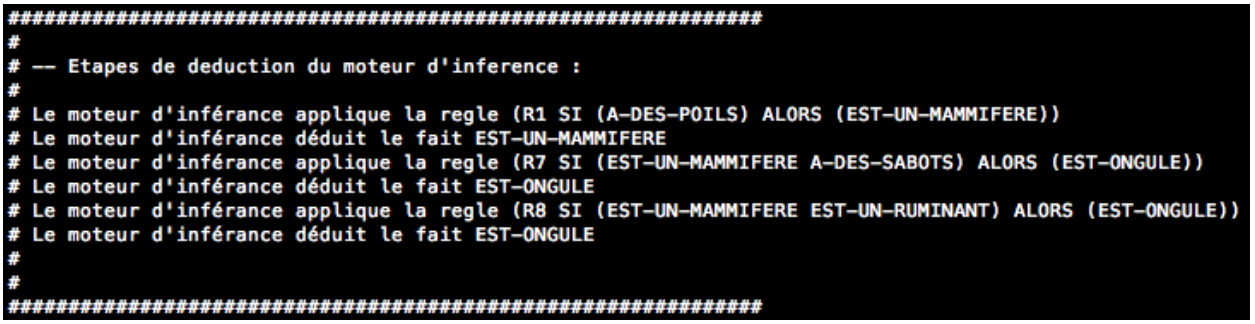


On choisi l'heuristique « la première règle valide » :



On choisi les faits connus : ici on choisi les faits « est-un-ruminant », « a-des-sabots » et « a-des-poils »

Le moteur d'inférence affiche la trace de ses déductions par l'application des règles disponibles dans la base de règle de Winston :



Important : pour déduire le fait « est-ongule » le moteur d'inférence avait le choix entre l'application de la règle R7 et la règle R8. Cependant, puisque nous

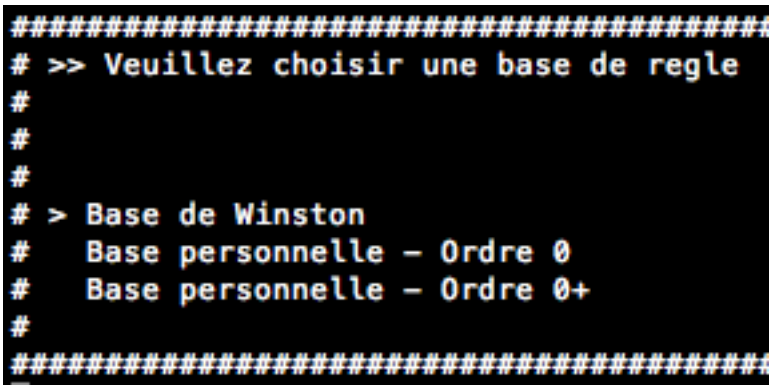
avons choisi l'heuristique « La première règle valide », la règle R7 est donc d'abord appliquée sans autre réflexion.

Pour finir le moteur d'inférence affiche sa conclusion qui contient tous les faits qui sont « vrai » :

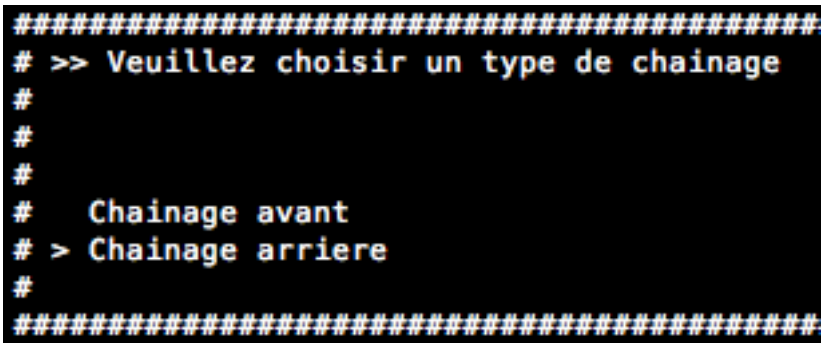
Ainsi le moteur d'inférence a déduis à partir des faits initiaux et des faits demandées, les faits « est-un-mammifere » (application de la règle R1) et « est-ongule » (application de la règle R7).

2) Exemple d'un chainage arrière

On choisi la base de Winston :



On choisi le chainage arrière :



On choisi le fait que l'on veut déduire en chinage arrière : ici on choisi le fait « est-un-albatros »

On répond aux faits demandables par « oui », « non » ou « ne sais pas » :

```
#################################################################
#
# La proposition VOLE est-elle vraie ? > Oui    Non    Ne sais pas
```

On décide que le fait « vole » est vrai.

```
#################################################################
#
# La proposition DONNE-DES-OEUFS est-elle vraie ? > Oui    Non    Ne sais pas
```

On décide que le fait « donne-des-œufs » est également vrai.

A ce stade le moteur d'inférence déduit le fait « est-un-oiseau ».

```
#################################################################
#
# La proposition VOLE-BIEN est-elle vraie ? > Oui    Non    Ne sais pas
```

On décide que le fait « vole-bien » est vrai.

Le moteur d'inférence affiche la trace de ses déductions par l'application des règles disponibles dans la base de règle de Winston :

```
#################################################################
#
# -- Etapes de deduction du moteur d'inference :
#
# Le moteur d'inférance essaye la regle (R15 SI (EST-UN-OISEAU VOLE-BIEN) ALORS (EST-UN-ALBATROS))
# Le moteur d'inférance essaye la regle (R4 SI (VOLE DONNE-DES-OEUFS) ALORS (EST-UN-OISEAU))
# La proposition VOLE est-elle vraie ? Oui
# La proposition DONNE-DES-OEUFS est-elle vraie ? Oui
# Le moteur d'inférance déduit le fait EST-UN-OISEAU
# La proposition VOLE-BIEN est-elle vraie ? Oui
# Le moteur d'inférance déduit le fait EST-UN-ALBATROS
#
#
#################################################################
```

Pour finir le moteur d'inférence affiche sa conclusion qui contient tous les faits qui sont « vrai » :

```
##############################################################
#
# -- La conclusion du moteur d'inférance est
#
# VOLE-BIEN est vraie
# EST-UN-ALBATROS est vraie
# DONNE-DES-OEUFS est vraie
# VOLE est vraie
# EST-UN-OISEAU est vraie
#
##############################################################
```

Le fait « est-un-albatros » est donc vrai dans ce cas.

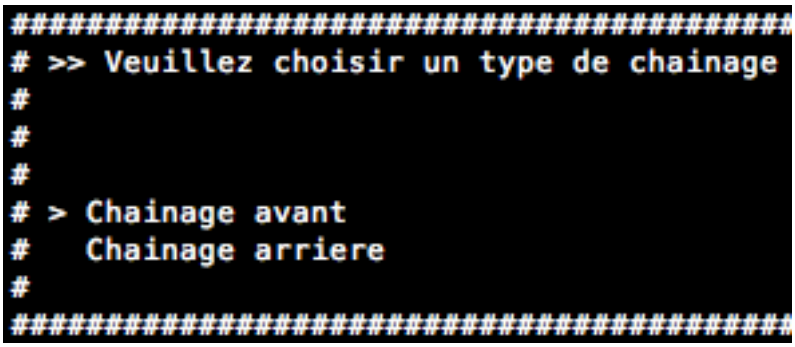
b) Tests sur la base de règles personnelle

1) Exemple d'un chainage avant

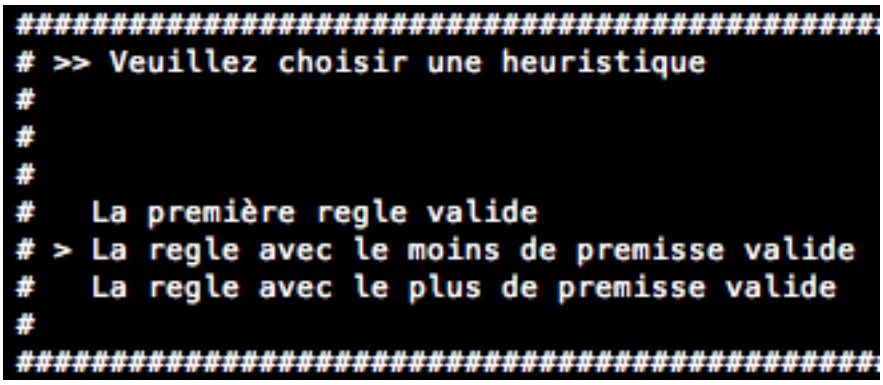
On choisi la base personnelle d'ordre 0 (cantine) :

```
##########################################
# >> Veuillez choisir une base de regle
#
#
#
#   Base de Winston
# > Base personnelle - Ordre 0
#   Base personnelle - Ordre 0+
#
##########################################
```

On choisi le chainage avant :



On choisi l'heuristique « la règle avec le moins de prémisse valide » :



On choisi les faits connus : ici on choisi les faits « a-tres-faim », « il-y-a-une-place-assise » et « a-une-pause-de-2h »

Le moteur d'inférence affiche la trace de ses déductions par l'application des règles disponibles dans la base de règle cantine :

```
############################################################
#
# -- Etapes de deduction du moteur d'inference :
#
# Le moteur d'inférance applique la regle (R12 SI (A-UNE-PAUSE-DE-2H) ALORS (A-LE-TEMPS))
# Le moteur d'inférance déduit le fait A-LE-TEMPS
# Le moteur d'inférance applique la regle (R4 SI (A-TRES-FAIM IL-Y-A-UNE-PLACE-ASSISE) ALORS (MANGE-ASSIS))
# Le moteur d'inférance déduit le fait MANGE-ASSIS
# Le moteur d'inférance applique la regle (R3 SI (A-TRES-FAIM IL-Y-A-UNE-PLACE-ASSISE A-LE-TEMPS) ALORS (PREND-PLAT-DU-JOUR))
# Le moteur d'inférance déduit le fait PREND-PLAT-DU-JOUR
# Le moteur d'inférance applique la regle (R9 SI (PREND-PLAT-DU-JOUR) ALORS (RECOIT-SEL RECOIT-POIVRE))
# Le moteur d'inférance déduit le fait RECOIT-SEL
# Le moteur d'inférance déduit le fait RECOIT-POIVRE
#
#
############################################################
```

Important : au départ il y a conflit entre la règle R4 et la règle R12, toutefois le moteur choisi d'appliquer la règle R12 car nous avons choisi l'heuristique qui donne la priorité à la règle qui a le moins de prémisses. C'est donc R12 qui est appliquée car celle-ci n'a qu'une prémisse alors que R4 a deux prémisses. Ensuite il y a encore ensemble de conflit avec la règle R3 et la règle R4, encore une fois c'est la règle qui a le moins de prémisses qui est d'abord appliquée. Donc R4 puisque celle-ci a deux prémisses alors que R3 a trois prémisses.

Pour finir le moteur d'inférence affiche sa conclusion qui contient tous les faits qui sont « vrai » :

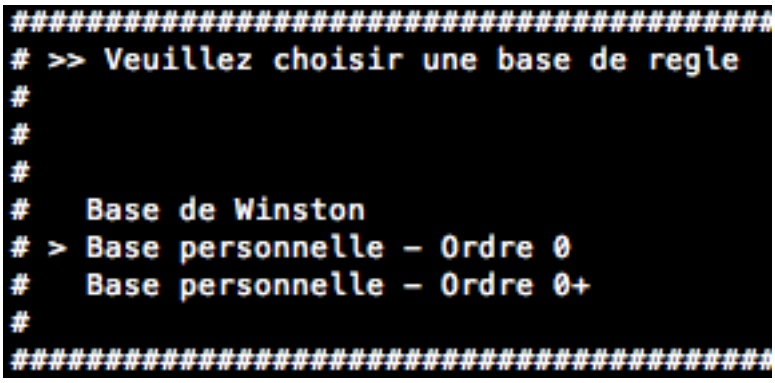
```
################################################
#
# -- La conclusion du moteur d'inférance est
#
# A-UNE-PAUSE-DE-2H est vraie
# RECOIT-POIVRE est vraie
# RECOIT-SEL est vraie
# MANGE-ASSIS est vraie
# A-LE-TEMPS est vraie
# IL-Y-A-UNE-PLACE-ASSISE est vraie
# A-TRES-FAIM est vraie
# PREND-PLAT-DU-JOUR est vraie
#
################################################
```

Ainsi le moteur d'inférence a déduis à partir des faits initiaux et des faits demandées, les faits « a-le-temps » (application de la règle R12), « mange-assis »

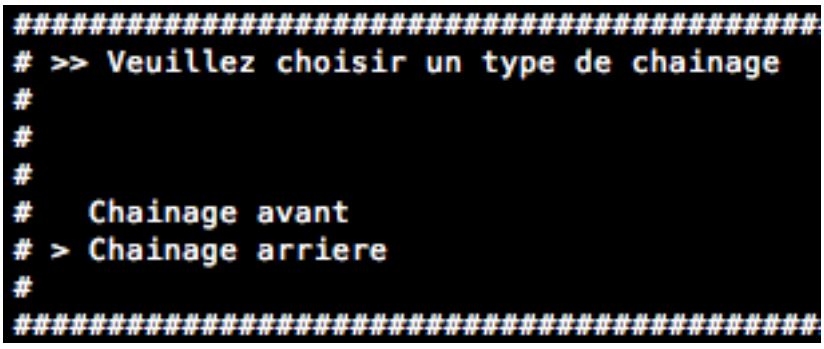
(application de la règle R4), « prend-plat-du-jour » (application de la règle R3) « recoit-poivre» et « recoit-sel » (application de la règle R9).

2) Exemple d'un chainage arrière

On choisi la base personnelle d'ordre 0 (cantine) :



On choisi le chainage arrière :



On choisi le fait que l'on veut déduire en chinage arrière : ici on choisi le fait « prend-menu »

On répond aux faits demandables par « oui », « non » ou « ne sais pas » :

```
#################################################################
#
# La proposition A-SOIF est-elle vraie ? > Oui    Non    Ne sais pas
```

On décide que le fait « a-soif » est vrai.

```
#################################################################
#
# La proposition N-A-PAS-BCP-DE-TEMPS est-elle vraie ?   Oui  > Non    Ne sais pas
```

On décide que le fait « n-a-pas-bcp-de-temps » est faux.

```
#################################################################
#
# La proposition A-TRES-FAIM est-elle vraie ? > Oui    Non    Ne sais pas
```

On décide que le fait « a-tres-faim» est vrai.

A ce stade le moteur d'inférence déduit le fait « est-un-oiseau ».

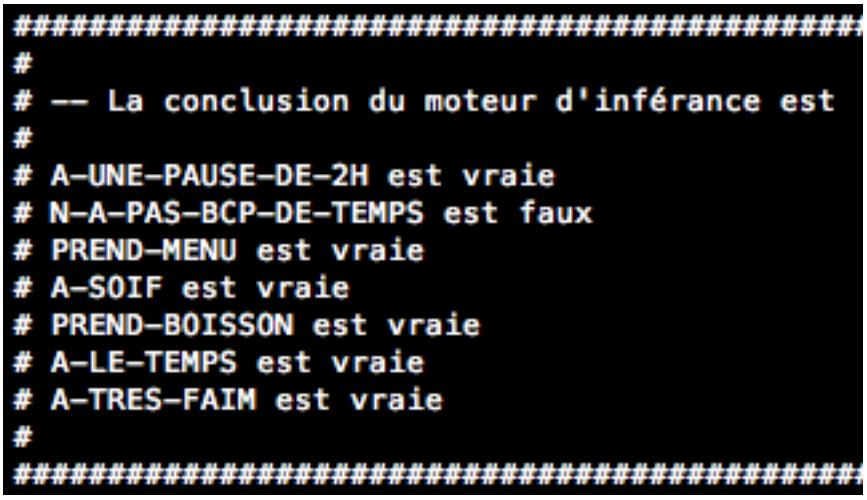
```
#################################################################
#
# La proposition A-UNE-PAUSE-DE-2H est-elle vraie ? > Oui    Non    Ne sais pas
```

On décide que le fait « a-une-pause-de-2h » est vrai.

Le moteur d'inférence affiche la trace de ses déductions par l'application des règles disponibles dans la base de règle cantine :

```
#############################################################
#
# -- Etapes de deduction du moteur d'inference :
#
# Le moteur d'inférance essaye la regle (R6 SI (PREND-BOISSON A-TRES-FAIM A-LE-TEMPS) ALORS (PREND-MENU))
# Le moteur d'inférance essaye la regle (R7 SI (A-SOIF N-A-PAS-BCP-DE-TEMPS) ALORS (PREND-SANDWICH PREND-BOISSON))
# La proposition A-SOIF est-elle vraie ? Oui
# La proposition N-A-PAS-BCP-DE-TEMPS est-elle vraie ? Non
# Le moteur d'inférance essaye la regle (R5 SI (A-SOIF) ALORS (PREND-BOISSON))
# Le moteur d'inférance déduit le fait PREND-BOISSON
# La proposition A-TRES-FAIM est-elle vraie ? Oui
# Le moteur d'inférance essaye la regle (R12 SI (A-UNE-PAUSE-DE-2H) ALORS (A-LE-TEMPS))
# La proposition A-UNE-PAUSE-DE-2H est-elle vraie ? Oui
# Le moteur d'inférance déduit le fait A-LE-TEMPS
# Le moteur d'inférance déduit le fait PREND-MENU
#
#
#############################################################
```

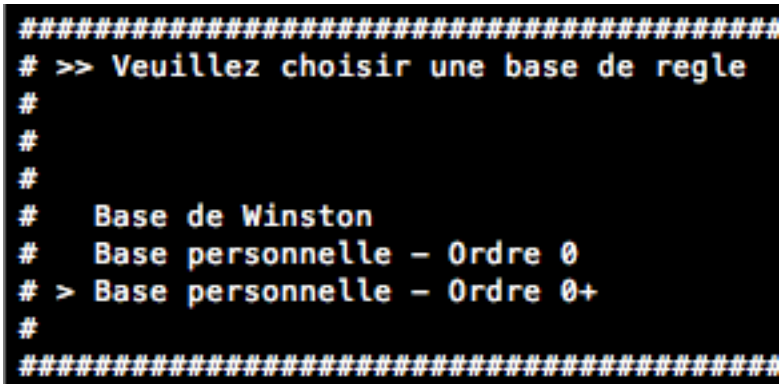
Pour finir le moteur d'inférence affiche sa conclusion qui contient tous les faits qui sont « vrai » :



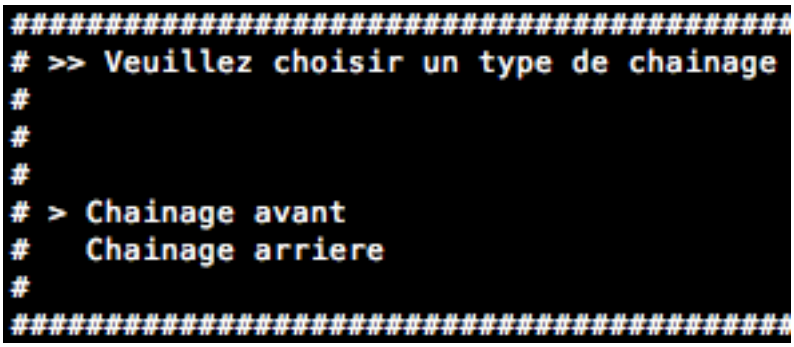
Le fait « prend-menu » est donc vrai dans ce cas.

c) Test d'un chainage avant sur la base de règles d'ordre 0+

On choisi la base personnelle d'ordre 0+ (achat voiture) :

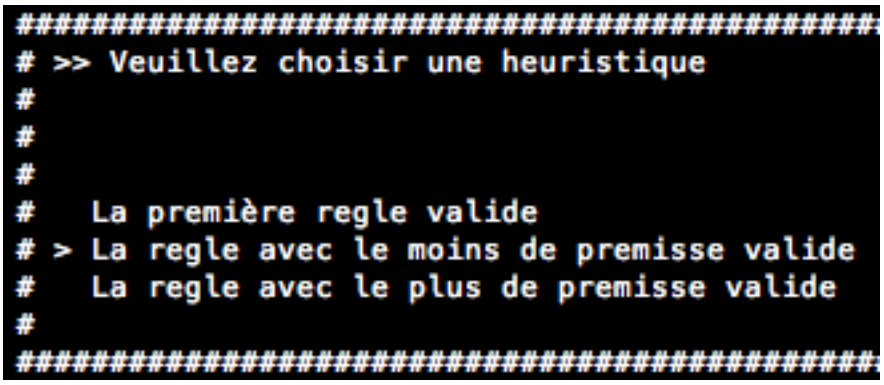


On choisi le chainage avant :



Pour ce test les faits connus sont « compagnon » avec pour valeur « vrai », « enfants » avec pour valeur « 0 » et « revenus » avec pour valeur 500000.
C'est à dire que la personne n'est pas célibataire mais n'a pas d'enfants et a un revenu égal à 500000€.

Cas 1 : choisi l'heuristique « la règle avec le moins de prémisse valide » :



Le moteur d'inférence affiche la trace de ses déductions par l'application des règles disponibles dans la base de règle d'ordre 0+ « achat voiture » :

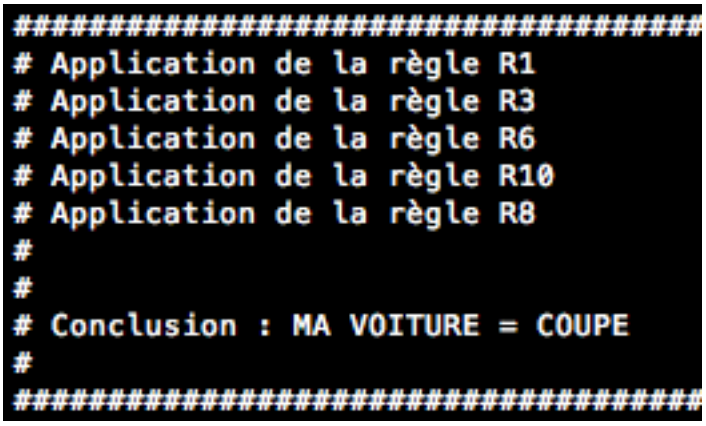
```
########################################
# Application de la règle R1
# Application de la règle R3
# Application de la règle R6
# Application de la règle R8
# Application de la règle R10
#
#
# Conclusion : MA VOITURE = COUPE
#
########################################
```

Important : après application des règles R1, R3 et R6 il y a conflit entre les règles R8 et R10, toutefois le moteur choisi d'appliquer d'abord la règle R8 car nous avons choisi l'heuristique qui donne la priorité à la règle qui a le moins de prémisses. C'est donc R8 qui est appliquée car celle-ci n'a qu'une prémisse alors que R10 a trois prémisses.

Cas 2 : choisi l'heuristique « la règle avec le plus de prémisse valide » :

```
################################################
# >> Veuillez choisir une heuristique
#
#
#
#   La première regle valide
#   La regle avec le moins de premisse valide
# > La regle avec le plus de premisse valide
#
################################################
```

Le moteur d'inférence affiche la trace de ses déductions par l'application des règles disponibles dans la base de règle d'ordre 0+ « achat voiture » :



Important : après application des règles R1, R3 et R6 il y a conflit entre les règles R8 et R10, cette fois le moteur choisi d'appliquer d'abord la règle R10 car nous avons choisi l'heuristique qui donne la priorité à la règle qui a le plus de prémisses. C'est donc R10 qui est appliquée car celle-ci a trois prémisses alors que R8 n'a qu'une prémisse.

Conclusion : dans les deux cas le moteur d'inférence a déduis à partir des faits connus, les faits « célibataire » égal à « faux » (application de la règle R1), « famille » égal à « sans enfants » (application de la règle R3), « fortune » égal à « riche » (application de la règle R6) et donc « voiture » égal à « coupe » (application de la règle R8 ou de la règle R10).