Projet Intelligence Artificielle

Université Paris-Dauphine

Michael GOLETTO
Salim GUENNOUNI
Victor ARFI
Enseignant : Pr. S. Pinson

Département MIDO
Master 1 Informatique des Organisations
Parcours MIAGE 2011/2012
UE : Intelligence Artificielle et Raisonnement

SOMMAIRE

Présentation du projet

Dans le cadre d'un projet académique d'Intelligence Artificielle, nous avons réalisé un moteur d'inférence en LISP. Le but de ce programme est de comprendre et d'implémenter les fondements du raisonnement déductif dans les Systèmes à Base de Connaissances (SBC).

Stratégie de contrôle et gestion des E.C.R

Nous avons choisi de créer une fonction permettant de classer dans un ordre précis la base de règles en fonction du nombre de prémisses gauches que chaque règle contient.

Heuristique du chainage avant

L'heuristique choisie permet d'extraire la règle ayant **le plus grand nombre** de prémisses gauches. Pour cela, nous comptons le nombre de conditions pour chaque règle dans l'ensemble de conflit. Notre méthode prend la première règle qui contient le maximum de conditions.

```
; Renvoie le nombre de premisses gauches de la regle r
(defun nbr_premiss (r)
  (length (partie_gauche_regle r))
)

; Renvoie le nombre maximal des premisses gauche de listeregle
(defun nbr_max_premiss (listeregle)
  (if (null listeregle)
     0
     (max (nbr_premiss (car listeregle))
        (nbr_max_premiss (cdr listeregle))
     )
  )
)

; Heuristique : retourne la regle ayant le nombre maximal de
premisses
(defun heuristique (rlist)
  (if (null rlist)
  NIL
  (if (equal (nbr_premiss (car rlist)) (nbr_max_premiss rlist) )
     (car rlist)
     (heuristique (cdr rlist))
     )
  )
)
```

Heuristique du chainage arrière

L'heuristique choisie pour le chainage arrière consiste à déterminer la règle ayant le minimum de prémisses inconnues, c'est-à-dire des faits dont la véracité n'est pas spécifiée. Ceci revient à privilégier des règles ayant un maximum de faits vrais.

Pour cela nous avons eu besoin d'implémenter une fonction permettant de calculer le nombre de prémisses inconnues dans une règle r :

```
;  Compte  le  nombre  de  prémisses  non-vraies  dans  une  regle  r
(premisses pas forcement fausses, mais dont la veracite n'a pas ete
defini)
(defun nbr_premiss_inconnues(r)
  (count_prop_inconnues (partie_gauche_regle r))
)
```

Celle-ci fait appel à une fonction auxiliaire qui calcule le nombre de prémisses inconnues d'une liste de propositions :

```
; Compte le nombre de prémisses non-vraies dans liste_de_prop (pas
forcement fausses)
(defun count_prop_inconnues (liste_de_prop)
  (cond
     ((null liste_de_prop) 0)
     ((not(vraie_prop (car liste_de_prop)))
        (+ 1 (count_prop_inconnues (cdr liste_de_prop)))
     )
     ((vraie_prop (car liste_de_prop))
        (+ 0 (count_prop_inconnues (cdr liste_de_prop)))
     )
  )
)
```

La fonction principale qui effectue le choix de la règle est la suivante :

```
;  Heuristique  :  retourne  la  regle  ayant  le  nombre  minimal  de
premisses
(defun heuristique_ARR (rlist)
  (if (null rlist)
  NIL
  (if   (equal   (nbr_premiss   (recuperer_regle_entiere   (car   rlist)
regles)) (nbr_min_premiss_ARR rlist) )
     (recuperer_regle_entiere (car rlist) regles)
     (heuristique_ARR (cdr rlist))
     )
  )
)
```

Elle fait appel à une fonction *nbr_min_premiss_ARR (listeregle)* qui calcule le minimum des prémisses gauches inconnues.

Contrôle des heuristiques

L'activation et désactivation des heuristiques sont contrôlées par les fonctions suivantes :

```
;  Permet  de  connaitre  la  valeur  de  la  propriete  heuristique  d'un
fait OK
(defun verif_heurisitque (p) (get p 'heuristique) )

; Met a true la propriete heuristique d'un fait OK
(defun activer_heurisitque (p) (putprop p 't 'heuristique) )

; Met a false la propriete heuristique d'un fait OK
(defun desactiver_heurisitque (p) (putprop p 'nil 'heuristique) )
```

Spécification des fonctions

Dans cette partie, nous présentons dans un premier temps le processus pour lancer le programme correctement, ensuite nous verrons les fonctions de contrôle du moteur d'inférence.

Fonctions de lancement

Les fonctions de lancement permettront de gérer le choix-utilisateur de l'algorithme du chainage, puis l'initialisation de l'instance LISP.

La fonction *(lancement)*

À travers d'une série de question-réponse, le programme arrive à déduire l'ensemble d'informations dont il a besoin pour exécuter le moteur d'inférence. Ceci se déroule en plusieurs étapes :

- Choix de la base de règle (base personnelle ou base de Winston) : *(choix_base_regles)*
- Choix de la stratégie de contrôle (lexicographique ou plus grand nombre de prémisses) : *(activer_heuristique 'config) ou (desactiver_heuristique 'config)*
- Choix du chainage avant ou arrière : *(choix_avant) ou (choix_arriere)*

```
; Un atome Config est cree
; Celui-ci prendra comme propriete l'ensemble des valeurs dont nous
avons besoins telles que
; la propriete heuristique a true pour l'heuristique
(setq config 'config)


(defun lancement ()

 ;Selection de la base de regle
 (choix_base_regles)

 ; Selection de l'heuristique (lexicographique ou plus grand nombre
de premisse)
 (Princ "Souhaitez-vous utiliser l'heuristique lexicographique (LX)
ou l'heuristique sur le plus grand nombre de premisses (PR) ?")
 (terpri)
 (setq heuristique (read))
 (if (member heuristique '(PR)) (activer_heurisitque 'config)
(desactiver_heurisitque 'config))

 ; Selection du chainage (avant ou arriÃ¨re)
 (Princ "Souhaitez-vous utiliser le chainage avant (AV) ou arriÃ¨re
(AR) ?")
 (terpri)
 (setq chainage (read))
 (cond
    ((member chainage '(AV)) (choix_avant))
    ((member chainage '(AR)) (choix_arriere))
    (t
       (princ "Quel dommage ! Nous n'avons malheuresement pas
compris ! Vous pouvez re-essayer (On est bien gentil quand meme!)")
       (terpri)
       (lancement)
```

```
      )
  )
)
```

Le choix de l'utilisateur entraine la saisie d'un fait ou des faits selon le chainage choisi :

- Le chainage avant entraine la saisie de la base de fait initiale :

```
(defun choix_avant ()
 ; Selection des faits
  (Princ "Bon, maintenant il va falloir saisir la base de faits
initiale !")
  (terpri)
  (Princ "(Sous la forme (fait_1 fait_2 ... fait_N). Merci de votre
comprÃ©hension.)")
  (terpri)
  (setq liste_de_prop (read))
  (terpri)
  (chainer_avant liste_de_prop)
)
```

- Le chainage arrière entraine la saisie du fait à démontrer :

```
(defun choix_arriere ()
 ; Selection des faits
  (Princ "Bon, maintenant il va falloir saisir le fait qu'on veut
demontrer !")
  (terpri)
  (Princ "(Sous la forme (fait_a_demontrer). Merci de votre
comprÃ©hension.)")
  (terpri)
  (setq a_demontrer (read))
  (terpri)
  (probleme a_demontrer)
)
```

Initialisation de l'instance LISP : *(initialiser)*

L'initialisation *« (initialiser) »* permet de vérifier qu'une base de règle est présente, puis se charge de générer une première base de fait en utilisant les primitives de compilation de règle. De plus elle permet « d'expliquer le raisonnement » c'est-à-dire à fixer les pas d'inférences du programme. Cette dernière reste au choix de l'utilisateur final.

La primitive (*initialiser)* commence tout d'abord par :

- Vérifier l'existence de la base de règle, une erreur est générée dans le cas contraire ;
- Fixer les pas d'inférences *(setq md t) ;*
- Compiler les règles afin de générer une première base de faits. Elle consiste à extraire l'ensemble des faits (prémisses et conclusions) et mettre à jour leurs propriétés en fonction de leurs spécificités ;
- Rafraichir les propriétés des différents faits *(RAZ, Remise À Zéro* des propriétés*) ;*
- Rafraichir les propriétés des règles.

```
(defun initialiser()
  (cond
     ; DEBUT COND 1
     (
     ; CONDITION
        (not (boundp 'regles))
         ; RESULT
        (print '(Vous ne m avez pas donne de regles))
           (error)
     ) ; FIN COND 1
  ) ; FIN COND

  (cond
     (
     (not (boundp 'propositions))
     (compile_regles)
     )
  )

  (Princ "Voulez vous que j'affiche mes pas d'inferences ?")
  (terpri)
  (setq reponse (read))

  (cond
     (
        (member reponse '(o y oui))
        (setq md t)
     )
     (
     ; Pas de gestions des caracteres bizares
     ; Creer une fonction recursive
        t
        (setq md nil)
     )
  )

  (mapc 'rafraichir_prop propositions)
  (mapc 'rafraichir_regle regles)
)
```

Chainage avant

Dans cette partie, nous expliciterons l'ensemble des primitives nécessaires pour le fonctionnement du chainage avant.

La fonction *« (chainer_avant liste_de_prop) »* est la fonction principale du chainage avant. Elle permet d'initialiser les propositions avec les propriétés adéquates, puis de lancer l'algorithme de chainage qui finit par présenter les résultats demandés.

```
(defun chainer_avant (liste_de_prop)
  (initialiser)
  (mapc 'marquer_vraie_prop liste_de_prop)
  (chainer_avant1)
  (presenter_resultats)
)
```

En utilisant la fonction « *(chainer_avant1)* », l'algorithme permet de spécifier le mode de chainage, avec ou sans heuristique en fonction de ce que l'utilisateur a choisi au démarrage de programme.

```
(defun chainer_avant1 ()
 (if (verif_heurisitque 'config)
    (setq rc (some_heuristique 'appliquer_regle regles))
    (setq rc (some_classique 'appliquer_regle regles))
    )
 (cond
    (
    ( null rc ) () )
    ( t (chainer_avant1) )
 )
)
```

La fonction *(some pred liste)* a été implémentée en deux versions :

- *(some_heuristique pred regles)* : permet d'appliquer le prédicat sur la règle déterminée par l'heuristique, cette dernière est choisie de la liste *regles.* En d'autres termes, le prédicat est appliqué sur la liste de règles ordonnée selon le nombre de prémisses gauches par ordre décroissant.

```
(defun some_heuristique (pred liste)
 (cond
    (
       (null liste)
       nil
    )
    (
       (apply pred
          (list (heuristique liste))
       )
          (heuristique liste)
    )
    (
    t
       (some_heuristique pred
          ; on retire la regles deja  testee
          (remove (heuristique liste) liste)
       )
    )
 )
)
```

- *(some_classique pred regles)* : permet d'appliquer le prédicat sur la règle déterminée *classiquement* par ordre lexicographique, cette dernière est choisie de la liste *regles.* En d'autres termes, le prédicat est appliqué sur la liste de règles ordonnée lexicographiquement.

```
(defun some_classique (pred liste)
 (cond
    (
       (null liste)
       nil
    )
    (
       (apply pred
```

```
            (list (car liste))
        )
            (car liste)
    )
    (
    t
        (some_classique pred
            (cdr liste)
        )
    )
  )
)
```

Chainage arrière

Le chainage arrière est contrôlé par la fonction principale *(probleme p),* elle permet d'initialiser l'instance LISP puis d'essayer de démontrer le fait p par chainage arrière. Elle finit par présenter les résultats trouvés.

```
(defun probleme (p)
  (initialiser)
  (etablir_prop p)
  (presenter_resultats)
)
```

La fonction *(etablir_prop p)* est une fonction récursive qui essaie de démontrer le fait donné en paramètre selon si il est vrai, terminal ou demandable. Dans le cas ou le fait est terminal ou demandable, le moteur d'inférence questionnera de l'utilisateur de sa véracité, puis il tentera de l'inférer selon la réponse de ce dernier.

```
(defun etablir_prop (p)
(cond ( (recherchee_prop p)
        ( cond
           ( (vraie_prop p) t)
           (t nil)
        )
    )
      ( (terminal p)
       (marquer_recherchee_prop p)
       (questionner_et_conclure_prop p)
    )
      ( (demandable p)
    ( cond
       ( (questionner_et_conclure_prop p) t)
          ( (equal reponse '?) (inferer_prop p ) )
    )
      )
      (t
    (marquer_recherchee_prop p)
    (inferer_prop p)
  )
))
```

La fonction *(inferer1 p lisreg1)* est la fonction qui tente de démontrer la véracité du fait donné en paramètre selon si l'heuristique a été choisie ou non :

```
(defun inferer1 (p lisregl)
 (princ "LES REGLES SUIVANTES RESTENT A DEMONTRER ") (princ lisregl)
 (terpri)
 (cond
    ( (null lisregl)  nil)
    (t
       ; Si l'heurisitique ets activé, on en passe pas car lisregl
mais la regle ayant le plus de premisses
       (if (verif_heurisitque 'config)
          (setq rc (heuristique_ARR lisregl))
          (setq rc (recuperer_regle_entiere (car lisregl) regles))
       )
       (princ "REGLE SLECTIONNEE : ") (princ rc)
       (terpri)
       (essaye_regle rc)
       (cond
          ((vraie_prop p) t)
          (t
             (inferer1 p (cdr lisregl))
          )
       )
    )
 )
)
```

Fonctions utilitaires

Définition de la fonction *putprop* en *CLISP*

```
; PutProp
(defun putprop (symbole valeur propriete)
    (setf
       (get symbole propriete)
       valeur
    )
    symbole
)
```

Partie gauche d'une règle

```
; Permet determiner la liste des premisses relative a la regle
passee en parametre
(defun partie_gauche_regle (r)(caddr r ))
```

Partie droite d'une règle

```
; Retourne la liste des conclusions d'une regle
(defun partie_droite_regle (r) (cadr(cdddr r )))
```

Nom du règle

```
; Retourne le nom d'une regle
```

```
(defun nom_regle (r) (car r ))
```

Propriété *appliquée* d'une règle

```
; Permet de savoir si une regle a deja ete utilisee
(defun appliquee (r) (get (nom_regle r) 'appliquee))

; Positionne a vrai la regle -appliquee- pour la regle
(defun marquer_appliquee (r) (putprop (nom_regle r) 't 'appliquee))
```

Règles concluantes sur un fait donné

La fonction suivante retourne toutes les règles ou le fait p est une conclusion.

```
; Donne, pour un fait donne, la liste des regles ou il fait partie
de la conclusion
(defun r_concluent_sur_prop (p) (get p 'concl))
```

Atome terminal

La fonction suivante vérifie l'atome donné est terminal, c'est-à-dire si il figure jamais en tant que conclusion dans une règle.

```
;  Un  atome  est  dit  terminal  si  la  liste  retournee  par
r_concluent_sur est vide
(defun terminal (p) (null (get p 'concl)))
```

Propriété *demandable* d'un fait

```
; Permet de connaitre la valeur de la propriete demandable d'un fait
(defun demandable (p) (get p 'demandable) )

; Met a true la propriete demandable d'un fait
(defun marquer_demandable_prop (p) (putprop p 't 'demandable) )
```

```
; Met a true la propriete demandable de la liste de faits
(defun      marquer_demandable_liste_de_prop      (lp)      (mapcar
'marquer_demandable_prop lp ))
```

Propriété *recherchée* d'un fait

```
; Traite la proposition -recherchee-
(defun recherchee_prop (p) (get p 'recherchee) )

; Met a true la propriete recherchee d'un fait
(defun marquer_recherchee_prop (p) (putprop p 't 'recherchee) )
```

Propriété *vraie* d'un fait

```
; Vérifie la proposition -vraie-
(defun vraie_prop (p) (get p 'vraie))

; Met a true la propriete vrai d'un fait
(defun marquer_vraie_prop (p) (putprop p 't 'vraie))
```

```
; Verifie si l'atome est marqué recherchee et qu'il ne soit pas
marqué vraie
(defun faux_prop (p)
 (and
    (recherchee_prop p)
    (not
       (vraie_prop p)
    )
 )
)
(defun vraie_pg (liste_de_prop)
 (not
    (member nil
       (mapcar 'vraie_prop liste_de_prop)
    )
 )
)
```

Propriété *présentée* d'un fait

```
; Traite la proposition -presentee-
(defun presentee_prop (p) (get p 'presentee) )

; Met a true la propriete presentee d'un fait
(defun marquer_presentee_prop (p) (putprop p 't 'presentee) )
```

Propriété *heuristique* d'un fait

```
; Permet de connaitre la valeur de la propriete heuristique d'un
fait
(defun verif_heurisitque (p) (get p 'heuristique) )

; Met a true la propriete heuristique d'un fait
(defun activer_heurisitque (p) (putprop p 't 'heuristique) )

; Met a false la propriete heuristique d'un fait
(defun desactiver_heurisitque (p) (putprop p 'nil 'heuristique) )
```

Les bases

Base de Winston

```
(defun regle_winston ()
 (setq regles'(
    (r1       si     (a-des-poils)
          alors (est-un-mamifere))
    (r2       si     (donne-du-lait)
          alors (est-un-mamifere))
    (r3       si     (a-des-plumes)
          alors (est-un-oiseau))
    (r4       si     (vole
              donne-des-oeufs)
          alors (est-un-oiseau))
    (r5    si     (mange-viande)
          alors (est-un-carnivore))
```

```
    (r6      si    (a-des-dents-pointues
             a-des-griffes
             a-des-yeux-frontaux)
          alors (est-un-carnivore))
    (r7      si  (est-un-mamifere
             a-des-sabots)
          alors (est-ongule))
    (r8      si (est-un-mamifere
             est-un-ruminant)
          alors (est-ongule))
    (r9      si    (est-un-mamifere
             a-une-couleur-fauve
             est-un-carnivore)
          alors (est-un-guepard))
    (r10  si    (est-un-mamifere
             a-une-couleur-fauve
             est-un-carnivore
             a-des-rayures-noires)
          alors (est-un-tigre))
    (r11  si (a-des-taches-noires
             a-de-longues-pattes
             est-ongule
             a-un-long-cou)
          alors (est-une-girafe))
    (r12  si (a-des-rayures-noires
             est-ongule)
          alors (est-un-zebre))
    (r13  si (a-un-long-cou
             est-un-oiseau
             ne-vole-pas
             est-noir-et-blanc)
          alors (est-une-autruche))
    (r14  si (ne-vole-pas
             est-un-oiseau
             est-noir-et-blanc
             nage)
          alors (est-un-pingouin))
    (r15  si (est-un-oiseau
             vole-bien)
          alors (est-un-albatros))
    )
  )
)
```

Base personnelle

Nous avons créé une base de connaissance qui aiderait les bacheliers à choisir leurs orientations post-bac, essentiellement leur cursus supérieur.

```
(defun regle_perso ()
  (setq regles
     '(
     (R1  si  (avoir-le-Bac  Vouloir-Travailler)  alors  (Inscrire-BTS-
IUT))
     (R2 si (avoir-le-Bac Vouloir-Etudier) alors (Inscrire-Licence))
```

```
     (R3 si (Inscrire-Licence avoir-le-Bac Bac-S) alors (Inscrire-
Licence-Scientifique))
     (R4 si (Inscrire-Licence avoir-le-Bac Bac-STG) alors (Inscrire-
Licence-Gestion))
     (R5 si (Inscrire-Licence avoir-le-Bac Bac-ES) alors (Inscrire-
Licence-Economie))
     (R6 si (Inscrire-Licence avoir-le-Bac Bac-Litterature) alors
(Inscrire-Licence-Litterature))
     (R7 si (avoir-la-Licence Vouloir-Travailler) alors (Inscrire-
Master-Professionnel))
     (R8 si (avoir-la-Licence Vouloir-Etudier) alors (Inscrire-
Master-Recherche))
     (R9 si (avoir-le-Master Vouloir-Travailler) alors (Inscrire-
Entretien-Entreprise))
     (R10 si (avoir-le-Master Vouloir-Etudier) alors (Inscrire-These-
Laboratoire))
     )
  )
)
```

Jeux d'essais

Lancement du programme

Le programme se lance en appelant la fonction *lancement ()*, celle-ci oriente l'utilisateur à faire des choix spécifiques afin configurer le moteur d'inférence. En cas d'erreur saisie par l'utilisateur, la fonction implémente un mécanisme récursif pour remédier à ce type de problème.

L'utilisateur choisi au départ :

- Choix de base de règles (Base de Winston ou Base personnelle)
- Choix du chainage à effectuer (Chainage avant ou arrière)
- Choix d'un chainage avec ou sans heuristique
- Saisie de la base de faits initiale pour un chainage avant, ou le fait à démontrer pour un chainage arrière
- Activation ou désactivation des pas d'inférences
- Saisie des faits dont la véracité est vraie en chainage arrière
- Saisie des faits demandables en chainage arrière

```
CL-USER 71 > (lancement)

"*************************************************************"
"***                  PROJET DE LISP                       ***"
"***                  Protagonistes :                      ***"
"***                    ARFI Victor                        ***"
"***                  GOLETTO Michael                      ***"
"***                  GUENNOUNI Salim                      ***"
"*************************************************************"
"***               Bienvenue sur notre projet IA !         ***"
"*************************************************************"


              X   X   XXXXX   XXXXX   XXXXX   X   X
              XX XX   X       X   X   X   X    X X
              X X X   XXX     XXXXX   XXXXX     X
              X   X   X       X  X    X  X      X
              X   X   XXXXX   X   X   X   X     X


XXXXX   X   X   XXXXX   XXX   XXXXX   XXXXX   X   X   XXXXX   XXXXX
X       X   X   X   X    X    X         X     XX XX   X   X   X
X       XXXXX   XXXXX    X    XXXXX     X     X X X   XXXXX   XXXXX
X       X   X   X  X     X        X     X     X   X   X   X       X
XXXXX   X   X   X   X   XXX   XXXXX     X     X   X   X   X   XXXXX


Souhaitez-vous utiliser la base de regle de Winston (WI winston Winston WINSTON) ou la notre (PE perso Perso PERSO) ?
Winston
Voici l'ensemble des règles :
(r1  si (a-des-poils) alors (est-un-mamifere))
(r2  si (donne-du-lait) alors (est-un-mamifere))
(r3  si (a-des-plumes) alors (est-un-oiseau))
(r4  si (vole donne-des-oeufs) alors (est-un-oiseau))

 (r1  si (a-des-poils) alors (est-un-mamifere))
 (r2  si (donne-du-lait) alors (est-un-mamifere))
 (r3  si (a-des-plumes) alors (est-un-oiseau))
 (r4  si (vole donne-des-oeufs) alors (est-un-oiseau))
 (r5  si (mange-viande) alors (est-un-carnivore))
 (r6  si (a-des-dents-pointues a-des-griffes a-des-yeux-frontaux) alors (est-un-carnivore))
 (r7  si (est-un-mamifere a-des-sabots) alors (est-ongule))
 (r8  si (est-un-mamifere est-un-ruminant) alors (est-ongule))
 (r9  si (est-un-mamifere a-une-couleur-fauve est-un-carnivore) alors (est-un-guepard))
 (r1  si (est-un-mamifere a-une-couleur-fauve est-un-carnivore a-des-rayures-noires) alors (est-un-tigre))
 (r11 si (a-des-taches-noires a-de-longues-pattes est-ongule a-un-long-cou) alors (est-une-girafe))
 (r12 si (a-des-rayures-noires est-ongule) alors (est-un-zebre))
 (r13 si (a-un-long-cou est-un-oiseau ne-vole-pas est-noir-et-blanc) alors (est-une-autruche))
 (r14 si (ne-vole-pas est-un-oiseau est-noir-et-blanc nage) alors (est-un-pingouin))
 (r15 si (est-un-oiseau vole-bien) alors (est-un-albatros))

 Souhaitez-vous utiliser le chainage avant (AV) ou arrière (AR) ?
 AV
 Souhaitez-vous utiliser l'heuristique lexicographique (LX) ou l'heuristique sur le plus grand nombre de premisses (PR) ?
 PR
 Bon, maintenant il va falloir saisir la base de faits initiale !
 (Sous la forme (fait_1 fait_2 ... fait_N). Merci de votre compréhension.)
 (a-des-poils est-un-ruminant a-une-couleur-fauve est-un-carnivore a-des-rayures-noires)
 Voulez vous que j'affiche mes pas d'inferences ?
 o
```

Gestion des erreurs

```
"************************************************************"
"***                 PROJET DE LISP                       ***"
"***                 Protagonistes :                      ***"
"***                   ARFI Victor                        ***"
"***                 GOLETTO Michael                      ***"
"***                 GUENNOUNI Salim                      ***"
"************************************************************"
"***              Bienvenue sur notre projet IA !         ***"
"************************************************************"


              X   X   XXXXX   XXXXX   XXXXX   X   X
              XX XX   X       X   X   X   X    X X
              X X X   XXX     XXXXX   XXXXX     X
              X   X   X       X  X    X  X      X
              X   X   XXXXX   X   X   X   X     X


XXXXX   X   X   XXXXX   XXX   XXXXX   XXXXX   X   X   XXXXX   XXXXX
X       X   X   X   X    X    X         X     XX XX   X   X   X
X       XXXXX   XXXXX    X    XXXXX     X     X X X   XXXXX   XXXXX
X       X   X   X  X     X        X     X     X   X   X   X       X
XXXXX   X   X   X   X   XXX   XXXXX     X     X   X   X   X   XXXXX


Souhaitez-vous utiliser la base de regle de Winston (WI winston Winston WINSTON) ou la notre (PE perso Perso PERSO) ?
ERREUR
Quel dommage ! Nous n'avons malheuresement pas compris ! Vous pouvez re-essayer (On est bien gentil quand même!)
Souhaitez-vous utiliser la base de regle de Winston (WI winston Winston WINSTON) ou la notre (PE perso Perso PERSO) ?
ERREUR
Quel dommage ! Nous n'avons malheuresement pas compris ! Vous pouvez re-essayer (On est bien gentil quand même!)
Souhaitez-vous utiliser la base de regle de Winston (WI winston Winston WINSTON) ou la notre (PE perso Perso PERSO) ?
PROBLEME


Souhaitez-vous utiliser la base de regle de Winston (WI winston Winston WINSTON) ou la notre (PE perso Perso PERSO) ?
ERREUR
Quel dommage ! Nous n'avons malheuresement pas compris ! Vous pouvez re-essayer (On est bien gentil quand même!)
Souhaitez-vous utiliser la base de regle de Winston (WI winston Winston WINSTON) ou la notre (PE perso Perso PERSO) ?
ERREUR
Quel dommage ! Nous n'avons malheuresement pas compris ! Vous pouvez re-essayer (On est bien gentil quand même!)
Souhaitez-vous utiliser la base de regle de Winston (WI winston Winston WINSTON) ou la notre (PE perso Perso PERSO) ?
PROBLEME
Quel dommage ! Nous n'avons malheuresement pas compris ! Vous pouvez re-essayer (On est bien gentil quand même!)
Souhaitez-vous utiliser la base de regle de Winston (WI winston Winston WINSTON) ou la notre (PE perso Perso PERSO) ?
PE
Voici l'ensemble des règles :
(R1  si (avoir-le-Bac Vouloir-Travailler) alors (Inscrire-BTS-IUT))
(R2  si (avoir-le-Bac Vouloir-Etudier) alors (Inscrire-Licence))
(R3  si (Inscrire-Licence avoir-le-Bac Bac-S) alors (Inscrire-Licence-Scientifique))
(R4  si (Inscrire-Licence avoir-le-Bac Bac-STG) alors (Inscrire-Licence-Gestion))
(R5  si (Inscrire-Licence avoir-le-Bac Bac-ES) alors (Inscrire-Licence-Economie))
(R6  si (Inscrire-Licence avoir-le-Bac Bac-Litterature) alors (Inscrire-Licence-Litterature))
(R7  si (avoir-la-Licence Vouloir-Travailler) alors (Inscrire-Master-Professionnel))
(R8  si (avoir-la-Licence Vouloir-Etudier) alors (Inscrire-Master-Recherche))
(R9  si (avoir-le-Master Vouloir-Travailler) alors (Inscrire-Entretien-Entreprise))
(R10 si (avoir-le-Master Vouloir-Etudier) alors (Inscrire-These-Laboratoire))

Souhaitez-vous utiliser le chainage avant (AV) ou arrière (AR) ?
```

Base de Winston

Chainage avant

Chainage avant sans heuristique

Ci-dessous une impression écran du chainage avant sans heuristique sur la base de Winston

```
Souhaitez-vous utiliser la base de regle de Winston (WI winston Winston WINSTON) ou la notre (PE perso Perso PERSO) ?
WI
Voici l'ensemble des règles :
(r1  si (a-des-poils) alors (est-un-mamifere))
(r2  si (donne-du-lait) alors (est-un-mamifere))
(r3  si (a-des-plumes) alors (est-un-oiseau))
(r4  si (vole donne-des-oeufs) alors (est-un-oiseau))
(r5  si (mange-viande) alors (est-un-carnivore))
(r6  si (a-des-dents-pointues a-des-griffes a-des-yeux-frontaux) alors (est-un-carnivore))
(r7  si (est-un-mamifere a-des-sabots) alors (est-ongule))
(r8  si (est-un-mamifere est-un-ruminant) alors (est-ongule))
(r9  si (est-un-mamifere a-une-couleur-fauve est-un-carnivore) alors (est-un-guepard))
(r1  si (est-un-mamifere a-une-couleur-fauve est-un-carnivore a-des-rayures-noires) alors (est-un-tigre))
(r11 si (a-des-taches-noires a-de-longues-pattes est-ongule a-un-long-cou) alors (est-une-girafe))
(r12 si (a-des-rayures-noires est-ongule) alors (est-un-zebre))
(r13 si (a-un-long-cou est-un-oiseau ne-vole-pas est-noir-et-blanc) alors (est-une-autruche))
(r14 si (ne-vole-pas est-un-oiseau est-noir-et-blanc nage) alors (est-un-pingouin))
(r15 si (est-un-oiseau vole-bien) alors (est-un-albatros))

Souhaitez-vous utiliser le chainage avant (AV) ou arrière (AR) ?
AV
Souhaitez-vous utiliser l'heuristique lexicographique (LX) ou l'heuristique sur le plus grand nombre de premisses (PR) ?
LX
Bon, maintenant il va falloir saisir la base de faits initiale !
(Sous la forme (fait_1 fait_2 ... fait_N). Merci de votre compréhension.)
(a-des-poils est-un-ruminant a-une-couleur-fauve est-un-carnivore a-des-rayures-noires)
Voulez vous que j'affiche mes pas d'inferences ?
o

J'applique la règle :
(R1 SI (A-DES-POILS) ALORS (EST-UN-MAMIFERE))
J'arrive à en déduire : EST-UN-MAMIFERE
J'applique la règle :
(R8 SI (EST-UN-MAMIFERE EST-UN-RUMINANT) ALORS (EST-ONGULE))
J'arrive à en déduire : EST-ONGULE
J'applique la règle :
(R9 SI (EST-UN-MAMIFERE A-UNE-COULEUR-FAUVE EST-UN-CARNIVORE) ALORS (EST-UN-GUEPARD))
J'arrive à en déduire : EST-UN-GUEPARD
J'applique la règle :
(R10 SI (EST-UN-MAMIFERE A-UNE-COULEUR-FAUVE EST-UN-CARNIVORE A-DES-RAYURES-NOIRES) ALORS (EST-UN-TIGRE))
J'arrive à en déduire : EST-UN-TIGRE
J'applique la règle :
(R12 SI (A-DES-RAYURES-NOIRES EST-ONGULE) ALORS (EST-UN-ZEBRE))
J'arrive à en déduire : EST-UN-ZEBRE


Voici ce que j'en conclut

(EST-UN-ZEBRE EST VRAIE)
(A-DES-RAYURES-NOIRES EST VRAIE)
(EST-UN-TIGRE EST VRAIE)
(A-UNE-COULEUR-FAUVE EST VRAIE)
(EST-UN-GUEPARD EST VRAIE)
(EST-UN-RUMINANT EST VRAIE)
(EST-ONGULE EST VRAIE)
(EST-UN-CARNIVORE EST VRAIE)
(A-DES-POILS EST VRAIE)
(EST-UN-MAMIFERE EST VRAIE)
CEST_TOUT

CL-USER 73 >
```

Chainage avant avec heuristique

```
(r1  si (a-des-poils) alors (est-un-mamifere))
(r2  si (donne-du-lait) alors (est-un-mamifere))
(r3  si (a-des-plumes) alors (est-un-oiseau))
(r4  si (vole donne-des-oeufs) alors (est-un-oiseau))
(r5  si (mange-viande) alors (est-un-carnivore))
(r6  si (a-des-dents-pointues a-des-griffes a-des-yeux-frontaux) alors (est-un-carnivore))
(r7  si (est-un-mamifere a-des-sabots) alors (est-ongule))
(r8  si (est-un-mamifere est-un-ruminant) alors (est-ongule))
(r9  si (est-un-mamifere a-une-couleur-fauve est-un-carnivore) alors (est-un-guepard))
(r1  si (est-un-mamifere a-une-couleur-fauve est-un-carnivore a-des-rayures-noires) alors (est-un-tigre))
(r11 si (a-des-taches-noires a-de-longues-pattes est-ongule a-un-long-cou) alors (est-une-girafe))
(r12 si (a-des-rayures-noires est-ongule) alors (est-un-zebre))
(r13 si (a-un-long-cou est-un-oiseau ne-vole-pas est-noir-et-blanc) alors (est-une-autruche))
(r14 si (ne-vole-pas est-un-oiseau est-noir-et-blanc nage) alors (est-un-pingouin))
(r15 si (est-un-oiseau vole-bien) alors (est-un-albatros))

Souhaitez-vous utiliser le chainage avant (AV) ou arrière (AR) ?
AV
Souhaitez-vous utiliser l'heuristique lexicographique (LX) ou l'heuristique sur le plus grand nombre de premisses (PR) ?
PR
Bon, maintenant il va falloir saisir la base de faits initiale !
(Sous la forme (fait_1 fait_2 ... fait_N). Merci de votre compréhension.)
(a-des-poils est-un-ruminant a-une-couleur-fauve est-un-carnivore a-des-rayures-noires)
Voulez vous que j'affiche mes pas d'inferences ?
o

J'applique la règle :
(R1 SI (A-DES-POILS) ALORS (EST-UN-MAMIFERE))
J'arrive à en déduire : EST-UN-MAMIFERE
J'applique la règle :
(R10 SI (EST-UN-MAMIFERE A-UNE-COULEUR-FAUVE EST-UN-CARNIVORE A-DES-RAYURES-NOIRES) ALORS (EST-UN-TIGRE))
J'arrive à en déduire : EST-UN-TIGRE
J'applique la règle :
(R9 SI (EST-UN-MAMIFERE A-UNE-COULEUR-FAUVE EST-UN-CARNIVORE) ALORS (EST-UN-GUEPARD))
J'applique la règle :
(R1 SI (A-DES-POILS) ALORS (EST-UN-MAMIFERE))
J'arrive à en déduire : EST-UN-MAMIFERE
J'applique la règle :
(R10 SI (EST-UN-MAMIFERE A-UNE-COULEUR-FAUVE EST-UN-CARNIVORE A-DES-RAYURES-NOIRES) ALORS (EST-UN-TIGRE))
J'arrive à en déduire : EST-UN-TIGRE
J'applique la règle :
(R9 SI (EST-UN-MAMIFERE A-UNE-COULEUR-FAUVE EST-UN-CARNIVORE) ALORS (EST-UN-GUEPARD))
J'arrive à en déduire : EST-UN-GUEPARD
J'applique la règle :
(R8 SI (EST-UN-MAMIFERE EST-UN-RUMINANT) ALORS (EST-ONGULE))
J'arrive à en déduire : EST-ONGULE
J'applique la règle :
(R12 SI (A-DES-RAYURES-NOIRES EST-ONGULE) ALORS (EST-UN-ZEBRE))
J'arrive à en déduire : EST-UN-ZEBRE


Voici ce que j'en conclut

(EST-UN-ZEBRE EST VRAIE)
(A-DES-RAYURES-NOIRES EST VRAIE)
(EST-UN-TIGRE EST VRAIE)
(A-UNE-COULEUR-FAUVE EST VRAIE)
(EST-UN-GUEPARD EST VRAIE)
(EST-UN-RUMINANT EST VRAIE)
(EST-ONGULE EST VRAIE)
(EST-UN-CARNIVORE EST VRAIE)
(A-DES-POILS EST VRAIE)
(EST-UN-MAMIFERE EST VRAIE)
CEST_TOUT

CL-USER 72 >
```

Chainage arrière

Chainage arrière sans heuristique

```
Voici l'ensemble des règles :
(r1  si (a-des-poils) alors (est-un-mamifere))
(r2  si (donne-du-lait) alors (est-un-mamifere))
(r3  si (a-des-plumes) alors (est-un-oiseau))
(r4  si (vole donne-des-oeufs) alors (est-un-oiseau))
(r5  si (mange-viande) alors (est-un-carnivore))
(r6  si (a-des-dents-pointues a-des-griffes a-des-yeux-frontaux) alors (est-un-carnivore))
(r7  si (est-un-mamifere a-des-sabots) alors (est-ongule))
(r8  si (est-un-mamifere est-un-ruminant) alors (est-ongule))
(r9  si (est-un-mamifere a-une-couleur-fauve est-un-carnivore) alors (est-un-guepard))
(r1  si (est-un-mamifere a-une-couleur-fauve est-un-carnivore a-des-rayures-noires) alors (est-un-tigre))
(r11 si (a-des-taches-noires a-de-longues-pattes est-ongule a-un-long-cou) alors (est-une-girafe))
(r12 si (a-des-rayures-noires est-ongule) alors (est-un-zebre))
(r13 si (a-un-long-cou est-un-oiseau ne-vole-pas est-noir-et-blanc) alors (est-une-autruche))
(r14 si (ne-vole-pas est-un-oiseau est-noir-et-blanc nage) alors (est-un-pingouin))
(r15 si (est-un-oiseau vole-bien) alors (est-un-albatros))

Souhaitez-vous utiliser le chainage avant (AV) ou arrière (AR) ?
AR
Souhaitez-vous utiliser l'heuristique lexicographique (LX) ou l'heuristique sur le plus petit nombre de premisses inconnues (PR) ?
LX
Bon, maintenant il va falloir saisir le fait qu'on veut démontrer !
(Sous la forme fait_a_demontrer. Merci de votre compréhension.)
est-une-girafe

Saisissez les faits à initialiser à vrais sous la forme (fait_1 fait_2 ... fait_N).
S'il n'y en a pas, saisissez non.
non

Saisissez les faits demandables sous la forme (fait_1 fait_2 ... fait_N).
S'il n'y en a pas, saisissez non.
non

Voulez vous que j'affiche mes pas d'inferences ?
oui
J'ai le choix entre les règles suivantes : (R11)
Je selectionne cette règle : (R11 SI (A-DES-TACHES-NOIRES A-DE-LONGUES-PATTES EST-ONGULE A-UN-LONG-COU) ALORS (EST-UNE-GIRAFE))
Je vais essayer cette regle : (R11 SI (A-DES-TACHES-NOIRES A-DE-LONGUES-PATTES EST-ONGULE A-UN-LONG-COU) ALORS (EST-UNE-GIRAFE))

Est-ce vrai(e) : A-DES-TACHES-NOIRES ?
o

Est-ce vrai(e) : A-DE-LONGUES-PATTES ?
o
J'ai le choix entre les règles suivantes : (R8 R7)
Je selectionne cette règle : (R8 SI (EST-UN-MAMIFERE EST-UN-RUMINANT) ALORS (EST-ONGULE))
Je vais essayer cette regle : (R8 SI (EST-UN-MAMIFERE EST-UN-RUMINANT) ALORS (EST-ONGULE))
J'ai le choix entre les règles suivantes : (R2 R1)
Je selectionne cette règle : (R2 SI (DONNE-DU-LAIT) ALORS (EST-UN-MAMIFERE))
Je vais essayer cette regle : (R2 SI (DONNE-DU-LAIT) ALORS (EST-UN-MAMIFERE))
Est-ce vrai(e) : DONNE-DU-LAIT ?
o
J'arrive à en déduire : EST-UN-MAMIFERE
Est-ce vrai(e) : EST-UN-RUMINANT ?
o
J'arrive à en déduire : EST-ONGULE
Est-ce vrai(e) : A-UN-LONG-COU ?
o
J'arrive à en déduire : EST-UNE-GIRAFE


Voici ce que j'en conclut

Voici ce que j'en conclut

(A-UN-LONG-COU EST VRAIE)
(A-DE-LONGUES-PATTES EST VRAIE)
(A-DES-TACHES-NOIRES EST VRAIE)
(EST-UNE-GIRAFE EST VRAIE)
(EST-UN-RUMINANT EST VRAIE)
(EST-ONGULE EST VRAIE)
(DONNE-DU-LAIT EST VRAIE)
(EST-UN-MAMIFERE EST VRAIE)
CEST_TOUT
```

Chainage arrière avec heuristique

```
Souhaitez-vous utiliser la base de regle de Winston (WI winston Winston WINSTON) ou la notre (PE perso Perso PERSO) ?
WI
Voici l'ensemble des règles :
(r1  si (a-des-poils) alors (est-un-mamifere))
(r2  si (donne-du-lait) alors (est-un-mamifere))
(r3  si (a-des-plumes) alors (est-un-oiseau))
(r4  si (vole donne-des-oeufs) alors (est-un-oiseau))
(r5  si (mange-viande) alors (est-un-carnivore))
(r6  si (a-des-dents-pointues a-des-griffes a-des-yeux-frontaux) alors (est-un-carnivore))
(r7  si (est-un-mamifere a-des-sabots) alors (est-ongule))
(r8  si (est-un-mamifere est-un-ruminant) alors (est-ongule))
(r9  si (est-un-mamifere a-une-couleur-fauve est-un-carnivore) alors (est-un-guepard))
(r1  si (est-un-mamifere a-une-couleur-fauve est-un-carnivore a-des-rayures-noires) alors (est-un-tigre))
(r11 si (a-des-taches-noires a-de-longues-pattes est-ongule a-un-long-cou) alors (est-une-girafe))
(r12 si (a-des-rayures-noires est-ongule) alors (est-un-zebre))
(r13 si (a-un-long-cou est-un-oiseau ne-vole-pas est-noir-et-blanc) alors (est-une-autruche))
(r14 si (ne-vole-pas est-un-oiseau est-noir-et-blanc nage) alors (est-un-pingouin))
(r15 si (est-un-oiseau vole-bien) alors (est-un-albatros))

Souhaitez-vous utiliser le chainage avant (AV) ou arrière (AR) ?
AR
Souhaitez-vous utiliser l'heuristique lexicographique (LX) ou l'heuristique sur le plus petit nombre de premisses inconnues (PR) ?
PR
Bon, maintenant il va falloir saisir le fait qu'on veut démontrer !
(Sous la forme fait_a_demontrer. Merci de votre compréhension.)
est-une-girafe

Saisissez les faits à initialiser à vrais sous la forme (fait_1 fait_2 ... fait_N).
S'il n'y en a pas, saisissez non.
non

Saisissez les faits demandables sous la forme (fait_1 fait_2 ... fait_N).
S'il n'y en a pas, saisissez non.


Saisissez les faits à initialiser à vrais sous la forme (fait_1 fait_2 ... fait_N).
S'il n'y en a pas, saisissez non.
non

Saisissez les faits demandables sous la forme (fait_1 fait_2 ... fait_N).
S'il n'y en a pas, saisissez non.
non

Voulez vous que j'affiche mes pas d'inferences ?
o
J'ai le choix entre les règles suivantes : (R11)
Je selectionne cette règle : (R11 SI (A-DES-TACHES-NOIRES A-DE-LONGUES-PATTES EST-ONGULE A-UN-LONG-COU) ALORS (EST-UNE-GIRAFE))
Je vais essayer cette regle : (R11 SI (A-DES-TACHES-NOIRES A-DE-LONGUES-PATTES EST-ONGULE A-UN-LONG-COU) ALORS (EST-UNE-GIRAFE))

Est-ce vrai(e) : A-DES-TACHES-NOIRES ?
o

Est-ce vrai(e) : A-DE-LONGUES-PATTES ?
o
J'ai le choix entre les règles suivantes : (R8 R7)
Je selectionne cette règle : (R8 SI (EST-UN-MAMIFERE EST-UN-RUMINANT) ALORS (EST-ONGULE))
Je vais essayer cette regle : (R8 SI (EST-UN-MAMIFERE EST-UN-RUMINANT) ALORS (EST-ONGULE))
J'ai le choix entre les règles suivantes : (R2 R1)
Je selectionne cette règle : (R2 SI (DONNE-DU-LAIT) ALORS (EST-UN-MAMIFERE))
Je vais essayer cette regle : (R2 SI (DONNE-DU-LAIT) ALORS (EST-UN-MAMIFERE))

Est-ce vrai(e) : DONNE-DU-LAIT ?
o
J'arrive à en déduire : EST-UN-MAMIFERE
Est-ce vrai(e) : EST-UN-RUMINANT ?
o
J'arrive à en déduire : EST-ONGULE
Est-ce vrai(e) : A-UN-LONG-COU ?
```

```
Est-ce vrai(e) : DONNE-DU-LAIT ?
o
J'arrive à en déduire : EST-UN-MAMIFERE
Est-ce vrai(e) : EST-UN-RUMINANT ?
o
J'arrive à en déduire : EST-ONGULE
Est-ce vrai(e) : A-UN-LONG-COU ?
o
J'arrive à en déduire : EST-UNE-GIRAFE


Voici ce que j'en conclut

(A-UN-LONG-COU EST VRAIE)
(A-DE-LONGUES-PATTES EST VRAIE)
(A-DES-TACHES-NOIRES EST VRAIE)
(EST-UNE-GIRAFE EST VRAIE)
(EST-UN-RUMINANT EST VRAIE)
(EST-ONGULE EST VRAIE)
(DONNE-DU-LAIT EST VRAIE)
(EST-UN-MAMIFERE EST VRAIE)
CEST_TOUT

CL-USER 75 >
```

Chainage arrière avec heuristique (saisie des faits vrais et demandables)

```
Souhaitez-vous utiliser la base de regle de Winston (WI winston Winston WINSTON) ou la notre (PE perso Perso PERSO) ?
WI
Voici l'ensemble des règles :
(r1  si (a-des-poils) alors (est-un-mamifere))
(r2  si (donne-du-lait) alors (est-un-mamifere))
(r3  si (a-des-plumes) alors (est-un-oiseau))
(r4  si (vole donne-des-oeufs) alors (est-un-oiseau))
(r5  si (mange-viande) alors (est-un-carnivore))
(r6  si (a-des-dents-pointues a-des-griffes a-des-yeux-frontaux) alors (est-un-carnivore))
(r7  si (est-un-mamifere a-des-sabots) alors (est-ongule))
(r8  si (est-un-mamifere est-un-ruminant) alors (est-ongule))
(r9  si (est-un-mamifere a-une-couleur-fauve est-un-carnivore) alors (est-un-guepard))
(r1  si (est-un-mamifere a-une-couleur-fauve est-un-carnivore a-des-rayures-noires) alors (est-un-tigre))
(r11 si (a-des-taches-noires a-de-longues-pattes est-ongule a-un-long-cou) alors (est-une-girafe))
(r12 si (a-des-rayures-noires est-ongule) alors (est-un-zebre))
(r13 si (a-un-long-cou est-un-oiseau ne-vole-pas est-noir-et-blanc) alors (est-une-autruche))
(r14 si (ne-vole-pas est-un-oiseau est-noir-et-blanc nage) alors (est-un-pingouin))
(r15 si (est-un-oiseau vole-bien) alors (est-un-albatros))

Souhaitez-vous utiliser le chainage avant (AV) ou arrière (AR) ?
AR
Souhaitez-vous utiliser l'heuristique lexicographique (LX) ou l'heuristique sur le plus petit nombre de premisses inconnues (PR) ?
PR
Bon, maintenant il va falloir saisir le fait qu'on veut démontrer !
(Sous la forme fait_a_demontrer. Merci de votre compréhension.)
est-un-albatros


Saisissez les faits à initialiser à vrais sous la forme (fait_1 fait_2 ... fait_N).
S'il n'y en a pas, saisissez non.
(vole-bien)
Saisissez les faits demandables sous la forme (fait_1 fait_2 ... fait_N).
S'il n'y en a pas, saisissez non.
(est-un-oiseau)
Voulez vous que j'affiche mes pas d'inferences ?
o
J'ai le choix entre les règles suivantes : (R15)
Je selectionne cette règle : (R15 SI (EST-UN-OISEAU VOLE-BIEN) ALORS (EST-UN-ALBATROS))
Je vais essayer cette regle : (R15 SI (EST-UN-OISEAU VOLE-BIEN) ALORS (EST-UN-ALBATROS))

Est-ce vrai(e) : EST-UN-OISEAU ?
o
```

```
J'arrive à en déduire : EST-UN-ALBATROS


Voici ce que j'en conclut

(VOLE-BIEN EST VRAIE)
(EST-UN-ALBATROS EST VRAIE)
(EST-UN-OISEAU EST VRAIE)
CEST_TOUT
```

Base personnelle

Chainage avant *(avec heuristique)*

```
Souhaitez-vous utiliser la base de regle de Winston (WI winston Winston WINSTON) ou la notre (PE perso Perso PERSO) ?
PE
Voici l'ensemble des règles :
(R1  si (avoir-le-Bac Vouloir-Travailler) alors (Inscrire-BTS-IUT))
(R2  si (avoir-le-Bac Vouloir-Etudier) alors (Inscrire-Licence))
(R3  si (Inscrire-Licence avoir-le-Bac Bac-S) alors (Inscrire-Licence-Scientifique))
(R4  si (Inscrire-Licence avoir-le-Bac Bac-STG) alors (Inscrire-Licence-Gestion))
(R5  si (Inscrire-Licence avoir-le-Bac Bac-ES) alors (Inscrire-Licence-Economie))
(R6  si (Inscrire-Licence avoir-le-Bac Bac-Litterature) alors (Inscrire-Licence-Litterature))
(R7  si (avoir-la-Licence Vouloir-Travailler) alors (Inscrire-Master-Professionnel))
(R8  si (avoir-la-Licence Vouloir-Etudier) alors (Inscrire-Master-Recherche))
(R9  si (avoir-le-Master Vouloir-Travailler) alors (Inscrire-Entretien-Entreprise))
(R10 si (avoir-le-Master Vouloir-Etudier) alors (Inscrire-These-Laboratoire))

Souhaitez-vous utiliser le chainage avant (AV) ou arrière (AR) ?
AV
Souhaitez-vous utiliser l'heuristique lexicographique (LX) ou l'heuristique sur le plus grand nombre de premisses (PR) ?
PR
Bon, maintenant il va falloir saisir la base de faits initiale !
(Sous la forme (fait_1 fait_2 ... fait_N). Merci de votre compréhension.)
(avoir-le-bac Vouloir-Etudier avoir-le-Bac Bac-S Vouloir-Travailler)
Voulez vous que j'affiche mes pas d'inferences ?
o

J'applique la règle :
(R1 SI (AVOIR-LE-BAC VOULOIR-TRAVAILLER) ALORS (INSCRIRE-BTS-IUT))
J'arrive à en déduire : INSCRIRE-BTS-IUT
J'applique la règle :
(R2 SI (AVOIR-LE-BAC VOULOIR-ETUDIER) ALORS (INSCRIRE-LICENCE))
J'arrive à en déduire : INSCRIRE-LICENCE
J'applique la règle :
(R3 SI (INSCRIRE-LICENCE AVOIR-LE-BAC BAC-S) ALORS (INSCRIRE-LICENCE-SCIENTIFIQUE))
J'arrive à en déduire : INSCRIRE-LICENCE-SCIENTIFIQUE


J'applique la règle :
(R1 SI (AVOIR-LE-BAC VOULOIR-TRAVAILLER) ALORS (INSCRIRE-BTS-IUT))
J'arrive à en déduire : INSCRIRE-BTS-IUT
J'applique la règle :
(R2 SI (AVOIR-LE-BAC VOULOIR-ETUDIER) ALORS (INSCRIRE-LICENCE))
J'arrive à en déduire : INSCRIRE-LICENCE
J'applique la règle :
(R3 SI (INSCRIRE-LICENCE AVOIR-LE-BAC BAC-S) ALORS (INSCRIRE-LICENCE-SCIENTIFIQUE))
J'arrive à en déduire : INSCRIRE-LICENCE-SCIENTIFIQUE


Voici ce que j'en conclut
CEST_TOUT
```

Chainage arrière *(sans heuristique)*

```
Souhaitez-vous utiliser la base de regle de Winston (WI winston Winston WINSTON) ou la notre (PE perso Perso PERSO) ?
PE
Voici l'ensemble des règles :
(R1  si (avoir-le-Bac Vouloir-Travailler) alors (Inscrire-BTS-IUT))
(R2  si (avoir-le-Bac Vouloir-Etudier) alors (Inscrire-Licence))
(R3  si (Inscrire-Licence avoir-le-Bac Bac-S) alors (Inscrire-Licence-Scientifique))
(R4  si (Inscrire-Licence avoir-le-Bac Bac-STG) alors (Inscrire-Licence-Gestion))
(R5  si (Inscrire-Licence avoir-le-Bac Bac-ES) alors (Inscrire-Licence-Economie))
(R6  si (Inscrire-Licence avoir-le-Bac Bac-Litterature) alors (Inscrire-Licence-Litterature))
(R7  si (avoir-la-Licence Vouloir-Travailler) alors (Inscrire-Master-Professionnel))
(R8  si (avoir-la-Licence Vouloir-Etudier) alors (Inscrire-Master-Recherche))
(R9  si (avoir-le-Master Vouloir-Travailler) alors (Inscrire-Entretien-Entreprise))
(R10 si (avoir-le-Master Vouloir-Etudier) alors (Inscrire-These-Laboratoire))

Souhaitez-vous utiliser le chainage avant (AV) ou arrière (AR) ?
AR
Souhaitez-vous utiliser l'heuristique lexicographique (LX) ou l'heuristique sur le plus petit nombre de premisses inconnues (PR) ?
LX
Bon, maintenant il va falloir saisir le fait qu'on veut démontrer !
(Sous la forme fait_a_demontrer. Merci de votre compréhension.)
Inscrire-These-Laboratoire

Voulez vous que j'affiche mes pas d'inferences ?
o
J'ai le choix entre les règles suivantes : (R10)
Je selectionne cette règle : (R10 SI (AVOIR-LE-MASTER VOULOIR-ETUDIER) ALORS (INSCRIRE-THESE-LABORATOIRE))
Je vais essayer cette regle : (R10 SI (AVOIR-LE-MASTER VOULOIR-ETUDIER) ALORS (INSCRIRE-THESE-LABORATOIRE))

Est-ce vrai(e) : AVOIR-LE-MASTER ?
o



Est-ce vrai(e) : VOULOIR-ETUDIER ?
o
J'arrive à en déduire : INSCRIRE-THESE-LABORATOIRE


Voici ce que j'en conclut

(INSCRIRE-THESE-LABORATOIRE EST VRAIE)
(AVOIR-LE-MASTER EST VRAIE)
(VOULOIR-ETUDIER EST VRAIE)
CEST_TOUT
```

Listing du projet

```
; Un atome Config est cree
; Celui-ci prendra comme propriete l'ensemble des valeurs dont nous
avons besoins telles que
; la propriete heuristique a true pour l'heuristique
(setq config 'config)


(defun presentation ()
  (print
"************************************************************")
     (print  "***                                          PROJET  DE  LISP
***")
     (print  "***                                          Protagonistes  :
***")
     (print  "***                                              ARFI  Victor
***")
     (print  "***                                          GOLETTO  Michael
***")
     (print  "***                                          GUENNOUNI  Salim
***")
```

```
  (print
"************************************************************")
     (print "***            Bienvenue   sur   notre   projet   IA   !
***")
  (print
"************************************************************")
  (terpri)
  (terpri)
  (terpri)
  (noel)
  (terpri)
  (terpri)
)

(defun noel ()
  (princ "                    X    X    XXXXX    XXXXX    XXXXX    X
X")(terpri)
  (princ "                   XX  XX    X         X    X    X    X     X
X")(terpri)
  (princ "                      X  X  X     XXX        XXXXX     XXXXX
X")(terpri)
  (princ "                     X    X    X          X   X      X   X
X")(terpri)
  (princ "                     X    X    XXXXX    X    X    X    X
X")(terpri)
  (terpri)
  (terpri)
  (princ "XXXXX   X   X   XXXXX   XXX   XXXXX   XXXXX   X   X   XXXXX
XXXXX")(terpri)
  (princ "X       X   X   X   X    X    X         X     XX XX   X   X
X")(terpri)
  (princ "X       XXXXX   XXXXX    X    XXXXX     X     X X X   XXXXX
XXXXX")(terpri)
  (princ "X       X   X   X  X     X        X     X     X   X   X   X
X")(terpri)
  (princ "XXXXX   X   X   X   X   XXX   XXXXX     X     X   X   X   X
XXXXX")(terpri)
)

(defun lancement ()

  (presentation)
  ;Selection de la base de regle
  (choix_base_regles)
     ; Selection du chainage (avant ou arrière)
  (Princ "Souhaitez-vous utiliser le chainage avant (AV) ou arrière
(AR) ?")
  (terpri)
  (setq chainage (read))
  (cond
     ((member chainage '(AV)) (choix_avant))
     ((member chainage '(AR)) (choix_arriere))
     (t
        (princ "Quel dommage ! Nous n'avons malheuresement pas
compris ! Vous pouvez re-essayer (On est bien gentil quand meme!)")
        (terpri)
        (lancement)
```

```
    )
  )
)

(defun choix_avant ()

  ; Selection de l'heuristique (lexicographique ou plus grand nombre
de premisse)
  (Princ "Souhaitez-vous utiliser l'heuristique lexicographique (LX)
ou l'heuristique sur le plus grand nombre de premisses (PR) ?")
  (terpri)
  (setq heuristique (read))
  (if  (member  heuristique  '(PR))  (activer_heurisitque  'config)
(desactiver_heurisitque 'config))

  ; Selection des faits
  (Princ "Bon, maintenant il va falloir saisir la base de faits
initiale !")
  (terpri)
  (Princ "(Sous la forme (fait_1 fait_2 ... fait_N). Merci de votre
compréhension.)")
  (terpri)
  (setq liste_de_prop (read))
  (terpri)
  (chainer_avant liste_de_prop)
)

(defun choix_arriere ()

  ; Selection de l'heuristique (lexicographique ou plus grand nombre
de premisse)
  (Princ "Souhaitez-vous utiliser l'heuristique lexicographique (LX)
ou l'heuristique sur le plus petit nombre de premisses inconnues
(PR) ?")
  (terpri)
  (setq heuristique (read))
  (if  (member  heuristique  '(PR))  (activer_heurisitque  'config)
(desactiver_heurisitque 'config))

  ; Selection des faits
  (Princ "Bon, maintenant il va falloir saisir le fait qu'on veut
démontrer !")
  (terpri)
  (Princ  "(Sous  la  forme  fait_a_demontrer.  Merci  de  votre
compréhension.)")
  (terpri)
  (setq a_demontrer (read))
  (terpri)
  (fait_vraie)
  (terpri)
  (fait_demandable)
  (terpri)
  (probleme a_demontrer)
)

(defun fait_demandable ()
```

```
  (Princ "Saisissez les faits demandables sous la forme (fait_1
fait_2 ... fait_N).")
  (terpri)
  (Princ "S'il n'y en a pas, saisissez non.")
  (terpri)
  (setq faits_demandable (read))
  (cond
     ((member faits_demandable '(non no n NON NO N)) nil)
     (t
        (marquer_demandable_liste_de_prop faits_demandable)
     )
  )
)

(defun fait_vraie ()
  (Princ "Saisissez les faits à initialiser à vrais sous la forme
(fait_1 fait_2 ... fait_N).")
  (terpri)
  (Princ "S'il n'y en a pas, saisissez non.")
  (terpri)
  (setq fait_vraie (read))
  (cond
     ((member fait_vraie '(non no n NON NO N)) nil)
     (t
        (marquer_vraie_liste_de_prop fait_vraie)
     )
  )
)

; Renvoie le nombre de premisses gauches de la regle r
(defun nbr_premiss (r)
  (length (partie_gauche_regle r))
)

;  Compte  le  nombre  de  prémisses  non-vraies  dans  une  regle  r
(premisses pas forcement fausses, mais dont la veracite n'a pas ete
defini)
(defun nbr_premiss_inconnues(r)
  (count_prop_inconnues (partie_gauche_regle r))
)

; Compte le nombre de prémisses non-vraies dans liste_de_prop (pas
forcement fausses)
(defun count_prop_inconnues (liste_de_prop)
  (cond
     ((null liste_de_prop) 0)
     ((not(vraie_prop (car liste_de_prop)))
        (+ 1 (count_prop_inconnues (cdr liste_de_prop)))
     )
     ((vraie_prop (car liste_de_prop))
        (+ 0 (count_prop_inconnues (cdr liste_de_prop)))
     )
  )
)

; Renvoie le nombre maximal des premisses gauche de listeregle
(defun nbr_max_premiss (listeregle)
```

```
  (if (null listeregle)
     0
     (max (nbr_premiss (car listeregle))
        (nbr_max_premiss (cdr listeregle))
     )
  )
)

; Heuristique : retourne la regle ayant le nombre maximal de
premisses
(defun heuristique (rlist)
  (if (null rlist)
  NIL
  (if (equal (nbr_premiss (car rlist)) (nbr_max_premiss rlist) )
     (car rlist)
     (heuristique (cdr rlist))
     )
  )
)

; Heuristique : retourne la regle ayant le nombre minimal de
premisses
(defun heuristique_ARR (rlist)
  (if (null rlist)
  NIL
  (if   (equal   (nbr_premiss   (recuperer_regle_entiere   (car   rlist)
regles)) (nbr_min_premiss_ARR rlist) )
     (recuperer_regle_entiere (car rlist) regles)
     (heuristique_ARR (cdr rlist))
     )
  )
)

; Renvoie le nombre maximal des premisses gauche de listeregle
(defun nbr_min_premiss_ARR (listeregle)
  (if (null listeregle)
     ; Nombre suffisamment grand, taille de la base de regles + 1
     (+ 1 (length regles))
     (min   (nbr_premiss   (recuperer_regle_entiere   (car   listeregle)
regles))
        (nbr_min_premiss_ARR (cdr listeregle))
     )
  )
)

(defun choix_base_regles ()

  ; Selection de la base de règle
  (Princ "Souhaitez-vous utiliser la base de regle de Winston (WI
winston Winston WINSTON) ou la notre (PE perso Perso PERSO) ?")
  (terpri)
  (setq choix_regles (read))
  (cond
     ((member   choix_regles   '(WI   winston   Winston   WINSTON))
(regle_winston)(afficher_BR_WI))
     ((member    choix_regles    '(PE    perso    Perso    PERSO))
(regle_perso)(afficher_BR_Perso))
```

```
    (t
       (princ "Quel dommage ! Nous n'avons malheuresement pas
compris ! Vous pouvez re-essayer (On est bien gentil quand même!)")
       (terpri)
       (choix_base_regles)
    )
 )
)

(defun regle_perso ()
 (setq regles
    '(
    (R1 si (avoir-le-Bac Vouloir-Travailler) alors (Inscrire-BTS-
IUT))
    (R2 si (avoir-le-Bac Vouloir-Etudier) alors (Inscrire-Licence))
    (R3 si (Inscrire-Licence avoir-le-Bac Bac-S) alors (Inscrire-
Licence-Scientifique))
    (R4 si (Inscrire-Licence avoir-le-Bac Bac-STG) alors (Inscrire-
Licence-Gestion))
    (R5 si (Inscrire-Licence avoir-le-Bac Bac-ES) alors (Inscrire-
Licence-Economie))
    (R6 si (Inscrire-Licence avoir-le-Bac Bac-Litterature) alors
(Inscrire-Licence-Litterature))
    (R7 si (avoir-la-Licence Vouloir-Travailler) alors (Inscrire-
Master-Professionnel))
    (R8 si (avoir-la-Licence Vouloir-Etudier) alors (Inscrire-
Master-Recherche))
    (R9 si (avoir-le-Master Vouloir-Travailler) alors (Inscrire-
Entretien-Entreprise))
    (R10 si (avoir-le-Master Vouloir-Etudier) alors (Inscrire-These-
Laboratoire))
    )
 )
)

(defun regle_winston ()
 (setq regles'(
    (r1      si    (a-des-poils)
          alors (est-un-mamifere))
    (r2      si    (donne-du-lait)
          alors (est-un-mamifere))
    (r3      si    (a-des-plumes)
          alors (est-un-oiseau))
    (r4      si    (vole
             donne-des-oeufs)
          alors (est-un-oiseau))
    (r5   si    (mange-viande)
          alors (est-un-carnivore))
    (r6      si    (a-des-dents-pointues
             a-des-griffes
             a-des-yeux-frontaux)
          alors (est-un-carnivore))
    (r7      si  (est-un-mamifere
             a-des-sabots)
          alors (est-ongule))
    (r8      si (est-un-mamifere
             est-un-ruminant)
```

```
           alors (est-ongule))
     (r9      si    (est-un-mamifere
              a-une-couleur-fauve
              est-un-carnivore)
           alors (est-un-guepard))
     (r10  si    (est-un-mamifere
              a-une-couleur-fauve
              est-un-carnivore
              a-des-rayures-noires)
           alors (est-un-tigre))
     (r11  si (a-des-taches-noires
              a-de-longues-pattes
              est-ongule
              a-un-long-cou)
           alors (est-une-girafe))
     (r12  si (a-des-rayures-noires
              est-ongule)
           alors (est-un-zebre))
     (r13  si (a-un-long-cou
              est-un-oiseau
              ne-vole-pas
              est-noir-et-blanc)
           alors (est-une-autruche))
     (r14  si (ne-vole-pas
              est-un-oiseau
              est-noir-et-blanc
              nage)
           alors (est-un-pingouin))
     (r15  si (est-un-oiseau
              vole-bien)
           alors (est-un-albatros))
     )
  )
)


(defun afficher_BR_WI ()
  (princ "Voici l'ensemble des règles :")(terpri)
  (princ "(r1  si (a-des-poils) alors (est-un-mamifere))")(terpri)
  (princ "(r2  si (donne-du-lait) alors (est-un-mamifere))")(terpri)
  (princ "(r3  si (a-des-plumes) alors (est-un-oiseau))")(terpri)
  (princ   "(r4      si    (vole    donne-des-oeufs)    alors    (est-un-
oiseau))")(terpri)
  (princ "(r5  si (mange-viande) alors (est-un-carnivore))")(terpri)
  (princ  "(r6    si  (a-des-dents-pointues  a-des-griffes  a-des-yeux-
frontaux) alors (est-un-carnivore))")(terpri)
  (princ   "(r7     si   (est-un-mamifere   a-des-sabots)   alors   (est-
ongule))")(terpri)
  (princ  "(r8    si  (est-un-mamifere  est-un-ruminant)  alors  (est-
ongule))")(terpri)
  (princ   "(r9     si   (est-un-mamifere   a-une-couleur-fauve   est-un-
carnivore) alors (est-un-guepard))")(terpri)
  (princ   "(r1     si   (est-un-mamifere   a-une-couleur-fauve   est-un-
carnivore a-des-rayures-noires) alors (est-un-tigre))")(terpri)
  (princ "(r11 si (a-des-taches-noires a-de-longues-pattes est-ongule
a-un-long-cou) alors (est-une-girafe))")(terpri)
```

```
  (princ "(r12 si (a-des-rayures-noires est-ongule) alors (est-un-
zebre))")(terpri)
  (princ "(r13 si (a-un-long-cou est-un-oiseau ne-vole-pas est-noir-
et-blanc) alors (est-une-autruche))")(terpri)
  (princ "(r14 si (ne-vole-pas est-un-oiseau est-noir-et-blanc nage)
alors (est-un-pingouin))")(terpri)
  (princ "(r15 si (est-un-oiseau vole-bien) alors (est-un-
albatros))")(terpri)
  (terpri)
)

(defun afficher_BR_Perso ()
  (princ "Voici l'ensemble des règles :")(terpri)
  (princ "(R1 si (avoir-le-Bac Vouloir-Travailler) alors (Inscrire-
BTS-IUT))")(terpri)
  (princ "(R2 si (avoir-le-Bac Vouloir-Etudier) alors (Inscrire-
Licence))")(terpri)
  (princ "(R3 si (Inscrire-Licence avoir-le-Bac Bac-S) alors
(Inscrire-Licence-Scientifique))")(terpri)
  (princ "(R4 si (Inscrire-Licence avoir-le-Bac Bac-STG) alors
(Inscrire-Licence-Gestion))")(terpri)
  (princ "(R5 si (Inscrire-Licence avoir-le-Bac Bac-ES) alors
(Inscrire-Licence-Economie))")(terpri)
  (princ "(R6 si (Inscrire-Licence avoir-le-Bac Bac-Litterature)
alors (Inscrire-Licence-Litterature))")(terpri)
  (princ "(R7 si (avoir-la-Licence Vouloir-Travailler) alors
(Inscrire-Master-Professionnel))")(terpri)
  (princ "(R8 si (avoir-la-Licence Vouloir-Etudier) alors (Inscrire-
Master-Recherche))")(terpri)
  (princ "(R9 si (avoir-le-Master Vouloir-Travailler) alors
(Inscrire-Entretien-Entreprise))")(terpri)
  (princ "(R10 si (avoir-le-Master Vouloir-Etudier) alors (Inscrire-
These-Laboratoire))")(terpri)
  (terpri)
)

; LES FONCTIONS _PROP ONT ETE TESTEES SUR UN ATOME : (setq f 'atome)

;  Permet determiner la liste des premisses relative a la regle
passee en parametre OK
(defun partie_gauche_regle (r)(caddr r ))

; Retourne la liste des conclusions d'une regle OK
(defun partie_droite_regle (r) (cadr(cdddr r )))

; Retourne le nom d'une regle OK
(defun nom_regle (r) (car r ))

; Permet de savoir si une regle a deja ete utilisee OK
(defun appliquee (r) (get (nom_regle r) 'appliquee))

; Positionne a vrai la regle -appliquee- pour la regle OK
(defun marquer_appliquee (r) (putprop (nom_regle r) 't 'appliquee))

; Donne, pour un fait donne, la liste des regles ou il fait partie
de la conclusion
(defun r_concluent_sur_prop (p) (get p 'concl)) ;OK
```

```
;  Un  atome  est  dit  terminal  si  la  liste  retournee  par
r_concluent_sur est vide
(defun terminal (p) (null (get p 'concl))) ;OK

; Permet de connaitre la valeur de la propriete demandable d'un fait
OK
(defun demandable (p) (get p 'demandable) )

; Met a true la propriete demandable d'un fait OK
(defun marquer_demandable_prop (p) (putprop p 't 'demandable) )

; Met a true la propriete demandable de la liste de faits OK
(defun      marquer_demandable_liste_de_prop      (lp)      (mapcar
'marquer_demandable_prop lp ))

; Traite la proposition -recherchee-, equivalent a Get OK
(defun recherchee_prop (p) (get p 'recherchee) )

; Met a true la propriete recherchee d'un fait  OK
(defun marquer_recherchee_prop (p) (putprop p 't 'recherchee) )

; Vérifie la proposition -vraie-, equivalent a Get OK
(defun vraie_prop (p) (get p 'vraie))

; Met a true la propriete vrai d'un fait  OK
(defun marquer_vraie_prop (p) (putprop p 't 'vraie))

; Verifie si l'atome est marqué recherchee et qu'il ne soit pas
marqué vraie
(defun faux_prop (p)
 (and
    (recherchee_prop p)
    (not
       (vraie_prop p)
    )
 )
)

; Met a true la propriete vraie de la liste de faits OK
(defun marquer_vraie_liste_de_prop (lp) (mapcar 'marquer_vraie_prop
lp ))

; Traite la proposition -presentee-, equivalent a Get OK
(defun presentee_prop (p) (get p 'presentee) )

; Met a true la propriete presentee d'un fait  OK
(defun marquer_presentee_prop (p) (putprop p 't 'presentee) )

; TERPI : Retour Chariot
; Pose une question => Ne recupère pas la reponse !
(defun poser_question_prop (p)
 (terpri)
 (princ "Est-ce vrai(e) : ")
 (princ p)
 (princ " ? ")
 (terpri)
```

```
)

; A ETE MODIFIE
; Fonction inconnue :P
(defun some_classique (pred liste)
 (cond
    (
       (null liste)
       nil
    )
    (
       (apply pred
          (list (car liste))
       )
          (car liste)
    )
    (
    t
       (some_classique pred
          (cdr liste)
       )
    )
 )
)

(defun some_heuristique (pred liste)
 (cond
    (
       (null liste)
       nil
    )
    (
       (apply pred
          (list (heuristique liste))
       )
          (heuristique liste)
    )
    (
    t
       (some_heuristique pred
          ; on retire la regles déjà testée
          (remove (heuristique liste) liste)
       )
    )
 )
)


;
(defun vraie_pg (liste_de_prop)
 (not
    (member nil
       (mapcar 'vraie_prop liste_de_prop)
    )
 )
)
```

```
(defun marquer_recherchee (p)(putprop p 't 'recherchee))

; putprop
(defun putprop (symbole valeur propriete)
    (setf
       (get symbole propriete)
       valeur
    )
    symbole
)


;  Permet  de  connaitre  la  valeur  de  la  propriete  heuristique  d'un
fait OK
(defun verif_heurisitque (p) (get p 'heuristique) )

; Met a true la propriete heuristique d'un fait OK
(defun activer_heurisitque (p) (putprop p 't 'heuristique) )

; Met a false la propriete heuristique d'un fait OK
(defun desactiver_heurisitque (p) (putprop p 'nil 'heuristique) )

(defun recuperer_regle_entiere (r base_de_regles)
  (if (equal (nom_regle (car base_de_regles))  r )
     (car base_de_regles)
     (recuperer_regle_entiere r (cdr base_de_regles))
  )
)

; LES FONCTIONS _PROP ONT ETE TESTEES SUR UN ATOME : (setq f 'atome)

;  Permet  determiner  la  liste  des  premisses  relative  a  la  regle
passee en parametre OK
(defun partie_gauche_regle (r)(caddr r ))

; Retourne la liste des conclusions d'une regle OK
(defun partie_droite_regle (r) (cadr(cdddr r )))

; Retourne le nom d'une regle OK
(defun nom_regle (r) (car r ))

; Permet de savoir si une regle a deja ete utilisee OK
(defun appliquee (r) (get (nom_regle r) 'appliquee))

; Positionne a vrai la regle -appliquee- pour la regle OK
(defun marquer_appliquee (r) (putprop (nom_regle r) 't 'appliquee))

; Donne, pour un fait donne, la liste des regles ou il fait partie
de la conclusion
(defun r_concluent_sur_prop (p) (get p 'concl)) ;OK

;  Un  atome  est  dit  terminal  si  la  liste  retournee  par
r_concluent_sur est vide
(defun terminal (p) (null (get p 'concl))) ;OK

; Permet de connaitre la valeur de la propriete demandable d'un fait
OK
```

```
(defun demandable (p) (get p 'demandable) )

; Met a true la propriete demandable d'un fait OK
(defun marquer_demandable_prop (p) (putprop p 't 'demandable) )

; Met a true la propriete demandable de la liste de faits OK
(defun      marquer_demandable_liste_de_prop      (lp)      (mapcar
'marquer_demandable_prop lp ))

; Traite la proposition -recherchee-, equivalent a Get OK
(defun recherchee_prop (p) (get p 'recherchee) )

; Met a true la propriete recherchee d'un fait  OK
(defun marquer_recherchee_prop (p) (putprop p 't 'recherchee) )

; Vérifie la proposition -vraie-, equivalent a Get OK
(defun vraie_prop (p) (get p 'vraie))

; Met a true la propriete vrai d'un fait  OK
(defun marquer_vraie_prop (p) (putprop p 't 'vraie))

; Verifie si l'atome est marqué recherchee et qu'il ne soit pas
marqué vraie
(defun faux_prop (p)
  (and
     (recherchee_prop p)
     (not
        (vraie_prop p)
     )
  )
)

; Met a true la propriete vraie de la liste de faits OK
(defun marquer_vraie_liste_de_prop (lp) (mapcar 'marquer_vraie_prop
lp ))

; Traite la proposition -presentee-, equivalent a Get OK
(defun presentee_prop (p) (get p 'presentee) )

; Met a true la propriete presentee d'un fait  OK
(defun marquer_presentee_prop (p) (putprop p 't 'presentee) )

; TERPI : Retour Chariot
; Pose une question => Ne recupère pas la reponse !
(defun poser_question_prop (p)
  (terpri)
  (princ "Est-ce vrai(e) : ")
  (princ p)
  (princ " ? ")
  (terpri)
)

; A ETE MODIFIE
; Fonction inconnue :P
(defun some_classique (pred liste)
  (cond
     (
```

```
        (null liste)
        nil
    )
    (
        (apply pred
            (list (car liste))
        )
            (car liste)
    )
    (
    t
        (some_classique pred
            (cdr liste)
        )
    )
 )
)

(defun some_heuristique (pred liste)
 (cond
    (
        (null liste)
        nil
    )
    (
        (apply pred
            (list (heuristique liste))
        )
            (heuristique liste)
    )
    (
    t
        (some_heuristique pred
            ; on retire la regles déjà testée
            (remove (heuristique liste) liste)
        )
    )
 )
)


;
(defun vraie_pg (liste_de_prop)
 (not
    (member nil
        (mapcar 'vraie_prop liste_de_prop)
    )
 )
)

(defun marquer_recherchee (p)(putprop p 't 'recherchee))

; putprop
(defun putprop (symbole valeur propriete)
    (setf
        (get symbole propriete)
        valeur
```

```
    )
    symbole
)


;  Permet  de  connaitre  la  valeur  de  la  propriete  heuristique  d'un
fait OK
(defun verif_heurisitque (p) (get p 'heuristique) )

; Met a true la propriete heuristique d'un fait OK
(defun activer_heurisitque (p) (putprop p 't 'heuristique) )

; Met a false la propriete heuristique d'un fait OK
(defun desactiver_heurisitque (p) (putprop p 'nil 'heuristique) )

(defun recuperer_regle_entiere (r base_de_regles)
 (if (equal (nom_regle (car base_de_regles))  r )
    (car base_de_regles)
    (recuperer_regle_entiere r (cdr base_de_regles))
 )
)

(defun initialiser()
 (cond
    ; DEBUT COND 1
    (
    ; CONDITION
       (not (boundp 'regles))
        ; RESULT
       (princ "Vous ne m avez pas donne de regles")
          (error)
    ) ; FIN COND 1
 ) ; FIN COND

 (cond
    (
    (not (boundp 'propositions))
    (compile_regles)
    )
 )

 (Princ "Voulez vous que j'affiche mes pas d'inferences ?")
 (terpri)
 (setq reponse (read))

 (cond
    (
       (member reponse '(o y oui))
       (setq md t)
    )
    (
    ; Pas de gestions des caracteres bizares
    ; Creer une fonction recursive
       t
       (setq md nil)
    )
 )
```

```
  (mapc 'rafraichir_prop propositions)
  (mapc 'rafraichir_regle regles)
)


(defun compile_regles ()
  (setq propositions nil)
  (mapc 'extraire_regles regles)
  'Compilees
)

(defun extraire_regles(r)
 (mapc
    #'(lambda(p)
       (cond
          (
             (not(member p propositions))
             (setq propositions (cons p propositions) )
          )
       )

          (cond
          (
             (not (member (nom_regle r) (get p 'concl) ) )
             (putprop p (cons (nom_regle r) (get p 'concl) ) 'concl)
          )
       )
    ) ; FIN DU LAMBDA

    (partie_droite_regle r)
 )

  (mapc
    #'(lambda (p)
       (cond
          (
             (not(member p propositions))
             (setq propositions (cons p propositions) )
          )
       )
    ) ; FIN DU LAMBDA

    (partie_gauche_regle r)
 )
)


(defun rafraichir_prop (p)

  (remprop p 'recherchee)
  (remprop p 'vraie)
  (remprop p 'presentee)
 )

(defun rafraichir_regle (r)
  (remprop (nom_regle r) 'appliquee)
```

```
)

(defun chainer_avant (liste_de_prop)
  (initialiser)
  (mapc 'marquer_vraie_prop liste_de_prop)
  (chainer_avant1)
  (presenter_resultats)
)

(defun chainer_avant1 ()
  (if (verif_heurisitque 'config)
     (setq rc (some_heuristique 'appliquer_regle regles))
     (setq rc (some_classique 'appliquer_regle regles))
     )
  (cond
     (
     ( null rc ) () )
     ( t (chainer_avant1) )
  )
)

(defun appliquer_regle (r)
  (cond
  ((appliquee r) nil)
  ((vraie_pg (partie_gauche_regle r))
  (marquer_appliquee r)
  (cond (
     (and (boundp 'md) md)
     (terpri)
     (princ "J'applique la règle : ")
     (terpri)
     (princ r)
     (terpri)
     )
  )
  (applique_partie_droite (partie_droite_regle r))
  )
  )
)

(defun presenter_resultats ()
  (terpri)(terpri)(terpri)
  (princ "Voici ce que j'en conclut")
  (terpri)
  (mapc 'presenter_prop propositions)
  'Cest_tout
)

(defun presenter_prop (p)
  (cond
     (  (presentee_prop p) nil)
     (t (marquer_presentee_prop p)
        (cond ((vraie_prop p) (print (list p 'est 'vraie)))
              ((faux_prop p)  (print (list p 'est 'faux)))
               (md
                  nil
```

```
                ;(princ "Je n'arrive à déduire rien de : ")(princ p
)(terpri)
              )
        )
     )
  )
)

(defun questionner_et_conclure_prop (p)
  (poser_question_prop p)
  (setq reponse (read))
  (cond (( member reponse '(y yes oui o))
  (marquer_vraie_prop p)
  t)
  ((member reponse '(non ? no n))
  nil)
  (t
  (princ "Repondez oui ou non ou ? (si vous ne savez pas)")
  (terpri)
  (questionner_et_conclure_prop p)))
)

(defun inferer_prop (p)
  (let
  ((regles_nouvelles (r_concluent_sur_prop p)))
  (inferer1 p regles_nouvelles))
)

(defun inferer1 (p lisregl)
  (princ "J'ai le choix entre les règles suivantes : ") (princ
lisregl)
  (terpri)
  (cond
     ( (null lisregl)  nil)
     (t
        ; Si l'heurisitique ets activé, on en passe pas car lisregl
mais la regle ayant le plus de premisses
        (if (verif_heurisitque 'config)
           (setq rc (heuristique_ARR lisregl))
           (setq rc (recuperer_regle_entiere (car lisregl) regles))
        )
        (princ "Je selectionne cette règle : ") (princ rc)
        (terpri)
        (essaye_regle rc)
        (cond
           ((vraie_prop p) t)
           (t
              (inferer1 p (cdr lisregl))
           )
        )
     )
  )
)


(defun applique_partie_droite (lp)
  (mapc 'marquer_vraie_prop lp)
```

```
  (mapc 'montre_deduction_prop lp)
)

(defun montre_deduction_prop (p)
  (cond
     (md
        (princ "J'arrive à en déduire : ")(princ p)
     )
  )
)


(defun essaye_regle (r)
  (cond
     (md
     (princ "Je vais essayer cette regle : ")(princ r) (terpri) ))

  (cond (
     (appliquee r) nil)
     (t
        (let
           (
              (v (etablir_partie_gauche (partie_gauche_regle r) ) )
           )
           (cond
              ( (null v) nil) ; La partie gauche est fausse ? La
regle echoue
              (t (applique_partie_droite (partie_droite_regle r)))
           )
           (marquer_appliquee r)
        )
     )
  )
)


(defun etablir_partie_gauche (liste_de_prop)
  (cond
     ; Y a-t-il une prop deja connue fausse ?
     ((some_classique 'faux_prop liste_de_prop) nil) ; Si oui, on
s'arrette la et on retourne nil
     (t (etablir_pg1 liste_de_prop) ) ; Sinon a la suivante
  )
)


(defun etablir_pg1 (liste_de_prop)
  (cond
     ((null liste_de_prop) t)
     ((null (etablir_prop (car liste_de_prop))) nil)
     (t (etablir_pg1 (cdr liste_de_prop)))))

(defun probleme (p)
  (initialiser)
  (etablir_prop p)
  (presenter_resultats)
)
```

```
(defun etablir_prop (p)
 (cond ( (recherchee_prop p)
          ( cond
             ( (vraie_prop p) t)
             (t nil)
          )
       )
       ( (terminal p)
          (marquer_recherchee_prop p)
          (questionner_et_conclure_prop p)
       )
       ( (demandable p)
       ( cond
          ( (questionner_et_conclure_prop p) t)
             ( (equal reponse '?) (inferer_prop p ) )
       )
       )
       (t
       (marquer_recherchee_prop p)
       (inferer_prop p)
    )
 )
)
```