CRIQUET. Un outil de base pour construire des systèmes experts. Version 5. Manuel d'utilisateur

Philippe Vignard

INRIA

CENTRE
SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:954 90 20

Rapports Techniques

N° 64

CRIQUET
UN OUTIL DE BASE POUR CONSTRUIRE DES SYSTÈMES EXPERTS
Version 5
MANUEL D'UTILISATEUR

Philippe VIGNARD

Décembre 1985

CRIQUET

un outil de base pour

construire des systemes experts

(Version 5)
Manuel d'utilisateur

Philippe VIGNARD +

+ INRIA, Sophia-Antipolis
06560 Valbonne FRANCE
Tel.:(93) 65-77-77
(poste 7875)

RESUME

Ce document décrit les commandes du système CRIQUET et comporte quelques remarques sur l'utilisation d'outils particuliers tels que les méta-règles. CRIQUET est un logiciel d'Intelligence Artificielle développé pour servir d'outil de base pour la construction de systèmes experts utilisant les règles de production

ABSTRACT

The system CRIQUET is described. We present a general guide to use the different commands. Some notes are presented about the use of special tools. CRIQUET is a production system for designing expert systems.

PLAN

TRANSPORT DE CRIQUET

Le code de CRIQUET version 5 est ecrit en Le_Lisp V15, compatible V14. Il tourne sur Vax 750/780 (Unix, VMS), HB68 (Multics), SM90 et micros (Configuration de 1 M). Il peut etre porte partout ou il y a Le_lisp V15. Le logiciel n'utilise que des instructions du langage de base (Le_Lisp).

Chargement du systeme

Il faut avoir dans l'environnement de travail tous les modules constituant le logiciel (voir hierarchie, chapitre gestion memoire).

Modifier dans le module "top.ll" le contenu de la variable globale #:criquet:criquet-dir qui contient le nom du directory (emplacement) ou se situent tous les modules composant le logiciel.

Charger le module de chargement : (load "top.ll")

Puis lancer le chargement : (top_criquet)

Le systeme demande le type de terminal utilise.
Cette information est necessaire pour la gestion de l'ecran et surtout pour permettre l'utilisation de l'editeur pleine page (pepe). Sans cette information, ou si le type de terminal donne est inconnu du systeme, le logiciel est inutisable.

La description du terminal est constituee d'un ensemble de definitions de primitives d'affichage.
Un exemple est donne avec le module te100.ll.
Le_Lisp connait par avance un grand nombre de terminaux tels que : te100, vt100, twist, tvi912, tvi920, h19, facit hp2621, dku. (voir documentation Le_Lisp).

Dans la version complete , le systeme demande aussi si on veut utiliser une gestion memoire avec overlay. Si ce n'est pas le cas, tous les modules sont charges en memoire. Sinon, seuls les modules du noyau sont charges.

Problemes de portabilite

Attention a l'utilisation de "comline" dans le code, ce qui est rare. Ces fonctions permettent d'utiliser des commandes du systeme hote, ce qui n'est pas portable. Il faut donc modifier des appels. Ils ne sont utilises que dans la version complete du logiciel.

Voir les commandes :

"sauve" et "fin" dans le module modnport.ll

1/ Dans la commande "sauve", on utilise un comline sur "ata". (comline "ata"). Sous multics cela permet de faire en sorte que tout ce qui sort a l'ecran, entre l'appel a la commande "sauve" et l'appel a la commande "fin", est sauvegarder dans un fichier.

2/ Dans la commande "fin", on ferme ce fichier de sauvegarde en le nommant "criquet" (comline "fo criquet; daf; ro ; dta ") Ces commandes ne sont utilisables que sous Multics.

UTILISATIONS DE CRIQUET

Ce logiciel, de tranport aise, permet de rapidement et facilement valider une approche systemes experts dans un nouveau domaine.

Mais l'insertion d'un grand nombre de fonctionnalites diverses, pour satisfaire cet objectif, empeche le systeme d'etre performant, caracteristique necessaire pour un produit definitif industriel.

Pour determiner precisement par une etude approfondie les fonctionnalites necessaires et les raisonnements adequats, il faut ensuite adapter les outils offerts et considerer le logiciel plutot comme un langage specialise pour la programmation du systeme adequat.
(voir paragraphe "extensibilite")
On peut aussi plus simplement modifier des fonctions existantes pour les adapter aux besoins de l'application (manipulation des agendas, des contextes, des meta-regles ...).

Ce systeme est ecrit en Le_Lisp . Les expressions fournies en reponse aux interrogations du systeme doivent etre terminees par un simple retour charriot. Pour signaler la fin d'un texte tenant sur plusieurs lignes,on termine par une ligne contenant le seul caractere 'tilda' ("~")

Les commandes peuvent etre utilisees sans parametres . Dans ce cas, les elements necessaires sont reclames en conversationnel par le systeme .

Le langage de commandes permet de ne pas parentheser les appels de fonctions et evite l'affichage de messages d'erreur de l'interpreteur lisp.

On ne peut pas dans les textes donnes, quoter des atomes pour eviter leur evaluation, par exemple lors de l'appel de fonctions dans les regles. Ceci car, pour faciliter les operations de lecture, on enleve toute signification speciale aux caracteres " ' , ; . et :
Pour corriger cela, il suffit soit de creer des fonctions qui n'evaluent pas leurs parametres, soit d'utiliser ces caracteres speciaux dans fichiers lisp qui sont ensuite charges dans l'environnement de travail .
Pour quoter des atomes on peut aussi utiliser la fonction lisp (quote) : (quote a) = 'a.

LE FORMALISME UTILISE

Un module d'interface en langage semi-naturel facilite l'expression des connaissances, mais il subsiste des contraintes. Les phrases employees ne doivent pas etre trop longue, pas plus de 4 mots significatifs. En effet le formalisme utilise pour les expressions dans la base de faits et de regles est le suivant :
(objet predicat attribut valeur)
Dans ce format :
- les verbes sont utilises a l'infinitif
- les noms au masculin singulier
- les adjectifs au masculin singulier .

Exemples :
dans les exemples suivants, l'objet est animal,
les predicats sont les verbes etre et avoir .
Il n'y a pas obligatoirement de valeur.

```
(animal avoir tache noir)
(animal manger viande)
(animal avoir dent pointu)
(animal etre carnivore)
```

Les valeurs sont souvent en fait des qualificateurs de l'attribut.

La representation interne d'une regle est la suivante :

```
<type numero
  <<cond1><cond2>   <condm>>
  <<act1><act2>   <actp>>
  <<commentaires> ..>
  <<conseils>..>
    coeff >
```

Le type est fixe par l'utilisateur , mais par defaut il y a 3 types predefinis : inference ,coherence ou equivalence .
Si une regle est donnee du type equivalence, le systeme genere automatiquement la regle opposee equivalente :
pour A --> B on ajoute B --> A
On peut aussi generer la contraposee des regles voulues.

Les actions <acti> sont de la forme :

```
      (ajoutfait (fait))
   ou (suppresfait(fait))
   ou (suppresbut (fait))
   ou (fonction (f))
```

ou f est une fonction definie dans l'environnement de travail.

On peut selon la version du systeme disposer de fonctions internes et/ou externes. Dans le premier cas ce sont de simples fonctions Lisp que l'utilisateur a definies avec les facilites donnees.
Dans l'autre cas ce sont aussi des fonctions Lisp mais qui

permettent d'appeler des programmes exterieurs dans d'autres environnements (Fortran Pascal).
Pour plus de precisions voir le manuel Le_Lisp sur les fonctions Defextern.

En partie condition on peut aussi avoir des fonctions et des termes prefixes par le mot cle "predicats". Ces derniers sont des expressions du genre :

predicat : operande operateur operande

ou l'operateur est parmi < <= = <> > >= , et l'operande est un atome, une variable ou un nombre.
Ces termes sont surtout utilises pour imposer des restrictions sur les variables utilisees. Ces restrictions sont prises en compte lors du mecanisme de filtrage pour decider de l'activation des regles.

Les fonctions en partie premisse doivent rendre t ou nil a l'evaluation.

Les commentaires sont constitues de texte pouvant etre consulte et utilise par le systeme pour s'expliquer par la suite.

Les conseils permettent a l'utilisateur d'imposer au systeme des regles a utiliser de preference apres la regle en question lors d'un cheminement avant .
On peut donner une liste des numeros de regles a considerer apres cette regle. On se rapproche ainsi d'une avance deterministe, mais cela peut accelerer le fonctionnement.
Cet artifice peut aussi etre implemente avec des meta-regles, mais de facon plus complexe.

Dans la base de faits un fait est note avec son coefficient de plausibilite . Un fait a donc le format suivant :

((fait) coeff)

Un exemple de base de faits peut etre :

((animal avoir poil) 0.5)
((animal manger viande) 0.8)

LE LANGAGE D'EXPRESSION

Le langage ne permet pas la manipulation du quantificateur existentiel ($\exists$), mais seulement du quantificateur universel ($\forall$).

La disjonction ne peut pas etre manipulee, mais des regles de re-ecriture permettent de l'exprimer logiquement :

SI A ou B ---> C
s'ecrit
SI A -->C et SI B --> C

et SI A --> B ou C
peut s'ecrire logiquement
SI A et non B --> C et SI A et non C --> B

Du point de vue programmation, la manipulation de la disjonction est assez facile a implementer en chainage avant et arriere mais sans variables libres (voir chapitre "raisonnements").

Enfin, la manipulation de la negation est possible (non A) ou ($\neg$ A), mais on ne suppose pas travailler dans un univers clos. Donc, le fait (non A) est vrai si il apparait explicitement dans la base de connaissances. Dans un univers clos (non A) est vrai si A n'apparait pas dans la base de faits.

Dans la version a venir (CRIQUET avec une base d'objets), le choix du type de negation utilisee sera aussi possible. Ce sera possible avec la commande "avecnon", parametres 1 ou 2 (voir commande avecnon).

Les termes utilisables sont des quadruplets de la forme :

(objet predicat attribut valeur)

- Dans les regles, on peut aussi utiliser des termes particuliers, prefixes par les mots cles "predicat" ou "fonction". Ces termes ne sont pas filtres avec la base de faits, mais evalues seulement apres instanciation.

  Le mot cle "predicat" permet d'indiquer des conditions sur les variables utilisees dans les regles. On specifie ainsi des conditions qui limite les possibilites d'instanciation des regles.

  Exemple :

```
       SI (toto mesurer ?x)
       ET (toto mesurer ?y)
       ET (predicat (?y > ?x))
    ALORS (suppresfait (toto mesurer ?x))
```

  Le mot cle "fonction" permet d'indiquer l'appel de fonction internes ou externes dans les premisses et actions des regles. En partie premisse, les evaluations des fonctions doivent rendre t ou nil.

  Exemple :

```
       SI (toto mesurer ?x)
       ET (toto avoir age ?y)
       ET (fonction (test-age-taille ?y ?x))
    ALORS (ajoutfait (toto etre trop grand))
```

  Ces termes (avec "fonction" ou "predicat") sont donc utiles quand certaines premisses pour etre satisfaites, ne doivent pas etre unifiees avec la base de faits mais evaluees. Le resultat de l'evaluation seul importe.

- Des termes predicatifs sont aussi utilisables, c'est a dire des termes de la forme:

operande operateur operande

avec operateur dans (< <= > >= = <>).
Lors de l'ajout d'un terme predicatif tel que : (X = 3) dans la base de faits, la variable mathematique X est aussi creee dans l'environnement de travail et initialisee a 3.
Ces variables sont utilisables dans toutes fonctions lisp.

Lors de l'activation d'un terme tel que :

(suppresfait (X = 3))

la variable mathematique X est remise a nil.

Le systeme gere, au sujet des termes predicatifs :

- la negation : (non (x > 3)) est memorise (x <= 3)
- le filtrage entre termes predicatifs:
  (x > 3) filtre avec (x > 2)
  (x = 2) ne filtre pas (x < 1)

Ces termes predicatifs avec ces quelques facilites de manipulations permettent l'utilisation d'un langage arithmetique reduit.

- On peut aussi utiliser des appels de fonctions dans les termes. Lors de l'utilisation par le systeme de tels termes, les appels de fonctions sont evalues et les resultats des evaluations remplacent l'appel des fonctions dans les termes.

  Les evaluations de ces appels se font apres leur instanciation avec la liste des instances connues a cet instant.

  C'est le terme resultat ainsi construit, qui est filtre avec la base de faits pour juger de l'activation de la regle ou pas.

  Les appels de fonctions peuvent contenir des variables classiques et des variables mathematiques.

  Exemple :

```
      SI (toto mesurer ?x)
ALORS (ajoutfait ( poids = (* poids-moyen ?x)))
```

  ?x est instanciee classiquement par identification avec la base de faits
  "poids-moyen" est une variable mathematique affectee au prealable

  l'activation de cette regle aura pour effet de creer la variable "poids".

  Si de tels appels de fonctions en partie premisses contiennent des variables classiques, il faut prendre soin de placer ces termes en fin de l'ensemble des premisses. De ce fait, on est assure que les variables seront instanciees lors de l'evaluation des appels de fonction.

  Ce probleme d'ordonnancement des termes existe aussi pour les termes prefixes par les mots cles "predicat" et "fonction" mais il est pris en charge par le systeme, lors de la saisie en interactif des regles (ajr) ou lors du chargement d'un fichier de regles (charger).

- On peut enfin introduire des variables locales, syntaxiques et/ou mathematiques avec le mot cle "sachant". Par exemple :

```
      SI (produit-X coute ?x)
      ET (sachant ?y = (Fonction-Y ?x))
      ET (fonction (test-normalite ?y))
    ALORS ajoutfait : produit-X vendu ?y
```

Le terme "sachant ?y = (Fonction-X ?x)" ne teste pas l'egalite des 2 membres comme le ferait le meme terme prefixe avec "predicat", mais il introduit simplement une nouvelle variable ?y. Ces termes sont voisins des termes prefixes par "fonction" mais en plus ils permettent de definir de nouvelles variables sans jouer de role decisif au sujet de l'activation de la regle ou pas.

Il n'est pas obligatoire de placer une variable syntaxique derriere "sachant". Si ce qu'il y a apres le signe "=" est un nombre, une variable mathematique est declaree (dans l'exemple, la variable "y" est declaree et non pas "?y" qui designe que la variable syntaxique).

Ce terme permet de definir dynamiquement des variables mathematiques correspondantes a des variable syntaxiques. Exemple :

```
      SI (toto mesurer ?x)
      ET (sachant taille = ?x)
    ALORS (ajoutfait (toto peser (evaler-poids taille))
```

Ainsi la variable mathematique "taille" est creee a l'activation de la regle et pourra etre utilisee dans d'autres regles.
Le terme prefixe par "sachant" est difficilement utilise en chainage arriere, car il introduit naturellement une variable libre, notion que le chainage arriere ne sait pas manipuler. Exemple :

```
      SI (a b ?x)
      ET (sachant ?y = (F ?x))
      ET (fonction (Test ?y))
    ALORS (ajoutfait (c d ?y))
```

Il y a dans cette regle une variable libre, qui est ?x. Elle apparait en premisses, mais pas en conclusions. De ce fait, l'instantion des conclusions en chainage arriere ne permet pas d'instancier toute la regle. Il reste donc des variables dans le but intermediaire suivant (a b ?x), ce qui fait echouer le chainage arriere. Ceci peut etre eviter en soignant la construction des regles et dans l'exemple en ecrivant :

```
      SI (a b ?x)
      ET (sachant ?y = (F ?x))
      ET (fonction (Test ?y))
    ALORS (ajoutfait (c d ?y ?x))
```

Il n'y a alors plus de variable libre.

ENVIRONNEMENT DE TRAVAIL

L'environnement de travail est essentiellement constitue de 7 variables libres :

- **rules = base de regles courante. Abreviation br
- **metarules = base des meta regles. Abreviation bmr
- **facts = base de faits courante. Abreviation bf
- **hypotheses = ensemble des faits que le systeme connait comme deductions finales possibles. Cet ensemble est construit par l'utilisateur en reponse a des questions du systeme lors d'ajouts de regles. Abreviation bh
- **buts = ensemble des buts partiels que l'utilisateur peut donner avant chaque activation d'un cheminement avant Ainsi en phase de choix s'il y a des buts partiels donnes le systeme cherche d'abord la regle qui permet d'atteindre le but courant partiel. Abreviation bb
- **agenda = un agenda pour permettre la generation de plan
- **contexte = le contexte courant de travail

Le module d'interface utilise egalement 5 dictionnaires :
dicopt dicoetre dicoadj diconom dicoverbe .
A chacun est associee la liste des mots significatifs contenus.
Diverses variables indiquent le mode de travail utilise, le type de controle, l'utilisation de buts, des meta-regles et le type de gestion de coherence a utiliser.

Pour sauvegarder les br, bmr ,bf , bh, bb ,contexte et agenda, on utilise des listes qui gerent les noms utilises pour ces sauvegardes : **listebf **listebr **listebmr **listebh **listebb **lag **lcont .
Pour sauvegarder les noms des fonctions internes et externes on utilise respectivement : **listefct **listefctext.

D'autres variables globales sont utilises. Les noms de ces variables sont prefixees par **. Une description generale est donnee dans le module : pr.ll

LE MODULE D'INTERFACE

Un module d'interface en langage semi-naturel facilite l'expression des connaissances. Les phrases reconnues restent simples, avec 4 mots significatifs au plus.
Ces phrases doivent verifier le format :

nom verbe nom adjectif .

Le systeme reconnait les formes actives et passives.
Il utilise aussi 5 dictionnaires pour reconnaitre le type des mots et pour permettre une verification semantique : dictionnaires des noms, adjectifs, verbes, petits-mots et des conjugaisons de etre .

Commandes du module d'interface :

- prd
  pour imprimer le contenu d'un dictionnaire de type donne

- ajmd
  pour ajouter un mot tle dans le dictionnaire voulu

- spmd
  pour supprimer un mot dans le dictionnaire voulu

- mdm
  pour modifier le type d'un nom ou d'un verbe

- ajsd
  pour ajouter des synonymes

- rchm
  recherche d'un mot dans un dictionnaire

INTERFACE AVEC D'AUTRES ENVIRONNEMENTS

On peut interfacer CRIQUET avec d'autres environnements en utilisant des fonctions externes si la version disponible de Le_Lisp le permet .
Pour plus de precisions voir le Manuel Le_Lisp version 14 au moins au sujet de la fonction "defextern".

Cette fonction Le_Lisp permet d'associer a une fonction Lisp une fonction externe (Pascal, Fortran ..).
L'utilisation unique de cette fonction d'interface permet des liens avec d'autres environnements en restant portable.
Mais des problemes se posent au niveau des passages de parametres. Si l'utilisation visee necessite un taux eleve d'echange de parametres entre les 2 environnements, alors la fonction Defextern ne suffit plus et il faut reliser un mecanisme de communications dependant du systeme hote et donc non portable.

MANIPULATIONS DES FAITS

La base de faits peut etre manipulee dynamiquement par par des fonctions activees dans regles. Les 2 primitives suivantes peuvent etre utilisees comme fonctions :

ajoutfait fait --> pour ajouter le fait passe en parametre dans la bf

detruirefait fait --> pour detruire le fait dans la bf

Dans ce cas les faits donnes dans les fonctions doivent etre quotes .

Commandes de manipulation des faits :

- ajf
  ajoute des faits donnes en conversationnel dans la base de faits. Les faits ne peuvent pas contenir de variables

- spf
  pour supprimer un ou plusieurs faits dans la base de faits. Si on ne donne aucun parametre, le systeme fonctionne en conversationnel et interroge l'utilisateur pour chaque fait de la base de faits.

- spbf
  supprime toute la base de faits

- bf
  imprime la base de faits

- cbf
  permet la construction de la base de faits en conversationnel

- mdf
  pour modifier le coefficient de vraisemblance de certains faits dans la base de faits . Si on ne connait pas le texte exact du fait a modifier le systeme interroge l'utilisateur sur chaque fait .

- tribf
  pour trier la base de faits selon les coefficients de vraisemblance dans l'ordre decroissant

- rechbf
  recherche associative dans la base de faits. On donne une forme a rechercher qui peut contenir des variables .

GESTION DE COHERENCE DES FAITS

L'utilisateur peut rentrer des regles de coherence de la meme maniere que d'autres regles , avec (ajr) .
Pour signifier que les faits A et B sont incoherents il faut rentrer une regle telle que : SI (A) et (B) ALORS (erreur) .
On precise que la regle est une regle de coherence et il n'y a que les premisses a donner.

Les regles de coherence sont ponderees comme les autres regles. Ceci permet au systeme de ne pas rejeter toutes les incoherences potentielles. Il suffit qu'il puisse choisir parmi les regles activables une regle autre qu'une regle de coherence.
Un test de coherence des faits est execute au choix de l'utilisateur lors d'ajouts de nouveaux faits dans la base de faits et lors de chaque deduction dans le raisonnement.
La gestion ainsi etablie concerne la compatibilite semantique des faits. Cette gestion avec une base de faits valide permet de tester la validite de la base de regles.
La gestion de coherence des faits n'est assuree que au premier niveau, c'est a dire avec un seul cycle de deduction.

De meme une certaine gestion de coherence logique est assuree. Ainsi, il n'est pas possible d'ajouter dans la BF un fait si son oppose y est deja. Les gestions de coherence logique et semantique sont optionnelles. Les commandes avecglog avecgsem sansglog et sansgsem permettent de preciser ce point.

On peut si on le desire, construire une base de regles qui sera consideree par le systeme comme la seule base a utiliser pour les tests de coherence. Il suffit de construire cette base, de la sauvegarder dans une entite dont on choisit le non, par exemple xxxx. Puis avec la commande (avecbrcoh xxxx) on force le systeme a ne plus utiliser que le contenu de xxxx pour les tests de coherence semantique.
Si on donne un nom xxxx inconnu ou egal a nil pour la commande avecbrcoh, la base de regles courante est prise par defaut.

MANIPULATIONS DE LA BASE D'HYPOTHESES

Les hypotheses sont les faits qui apparaissent dans les "ajoutfaits" des regles et qui sont designes par le concepteur comme pouvant etre des conclusions finales lors d'un cheminement avant.

Ces hypotheses sont les faits que le systeme cherche a diagnostiquer en chainage arriere si l'utilisateur ne donne pas de but precis.

Les primitives de manipulation de l'ensemble des hypotheses sont sommaires et obligatoirement avec parametres.

- bh
  pour lister les elements de hypotheses

- ajh
  pour ajouter une hypothese.
  En donnant en parametre le texte a ajouter.

- sph
  pour supprimer une hypothese.
  En donnant en parametre le texte a supprimer.

- spbh
  pour supprimer toute la base d'hypotheses.

MANIPULATIONS DE LA BASE DE BUTS

Les buts sont utilises lors d'un cheminement mixte (chainage avant avec des buts- voir rapport de recherche INRIA 380) Ils permettent de diriger le cheminement sur des buts intermediaires.

La base de buts peut etre manipulee par l'utilisateur avec les commandes : bb ajb spb cbb spbb, mais aussi dynamiquement avec l'appel de fonctions dans les regles avec les primitives : ajoutbut detruirebut . Ces primitives sont appelees dans les regles comme des fonctions et doivent avoir pour parametre le texte du but a ajouter ou detruire. Les buts seront utilises comme le desire l'utilisateur, voir commandes avecbutord et sansbutord.

Commandes de manipulation de la base de buts :

- bb
  pour lister les buts

- ajb
  pour ajouter un but
  On ne peut pas utiliser de variables dans les buts.

- spb
  pour supprimer un but

- cbb
  pour creer l'ensemble buts

- spbb
  pour supprimer tous les buts

MANIPULATIONS DE CONTEXTES

Un contexte contient une bf, br, bmr, bb , bh et un agenda. On peut constituer, sauvegarder, charger et activer un contexte. La notion de contexte permet de hierarchiser une base de connaissances et ainsi de ne manipuler que de petites bases .

Les contextes peuvent etre manipules par l'utilisateur avec les commandes : crcont spcont prlcont chcont. Le systeme, avec les regles, peut aussi manipuler les contextes avec : crcont chcont spcont. Ces primitives sont utilisees dans les regles comme des fonctions et ont en parametre le nom du contexte a utiliser.

Les contextes peuvent etre manipules par les regles, c'est a dire dynamiquement en cours de raisonnement. Des regles peuvent activer des changements de contextes et ainsi permettre une evolution dans la hierarchie des contextes .

Commandes de manipulation des contextes :

- crcont
  creation d'un contexte a l'aide des bases courantes

- spcont
  supprimer un contexte deja memorise

- prlcont
  imprimer la liste des noms de contextes memorises

- chcont
  charger un contexte de nom donne

MANIPULATIONS DE L'AGENDA

Un agenda permet de mettre au point un plan de travail qui sera ensuite active et/ou memorise.
On peut placer dans l'agenda n'importe quelle fonction Lisp.
L'agenda courant, et en general tous les agendas crees et memorises peuvent etre utilises aussi bien par l'utilisateur que dynamiquement par le systeme avec les commandes :
sauvag chargag prlag ajag vidag extag actag pray.
Pour voir le parametrage de ces fonctions, utiliser help sur chacune d'elles.

Commandes de manipulation de l'agenda :

- sauvag
  memoriser l'agenda courant sous le nom donne en parametre

- chargag
  charger un agenda de nom donne

- prlag
  impression des noms d'agendas memorises

- ajag
  parametres : fonction position
  Le premier parametre est obligatoire.
  Pour ajouter une fonction dans l'agenda a la position donne.
  Sans position donnee, l'ajout se fait en tete de l'agenda.
  Si la position donnee est plus grande que la taille de l'agenda, l'ajout se fait en fin de l'agenda.

- spag
  1 parametre = une position
  Pour supprimer l'element de position donnee. Sans position, supprime le premier de l'agenda courant.

- vidag
  pour vider l'agenda courant et le reinitialiser a nil
  ou detruire un agenda memorise de nom donne.

- extag
  pour extraire un element de l'agenda courant en donnant sa position .

- actag
  2 parametres = 2 positions
  pour activer un element de l'agenda en donnant sa position
  ou pour activer les elements de l'agenda entre les 2 positions donnee.
  Sans position donnee tout l'agenda est active.

- prag
  pour imprimer l'agenda courant

MANIPULATIONS DES FONCTIONS

On peut manipuler des fonctions Lisp. Ce sont des fonctions internes c'est a dire definies dans l'environnment Lisp et se contentant de celui-ci. Elles peuvent etre referencees dans les regles en partie premisses et actions. Ces fonctions peuvent etre des primitives du systeme lui-meme. Le systeme peut donc se modifier sans intervention de l'utilisateur.
Ou bien ce sont des fonctions externes, c'est a dire des fonctions Lisp qui referencient des fonctions externes definies dans un tout autre environnement.
Voir les defextern dans un manuel Le_Lisp.

Des fonctions internes predefinies permettent d'utiliser un langage arithmetique. Ce sont les fonctions :

- *decl --> declaration d'une variable mathematique
- *affec --> affectation d'une variable
- *val --> donne la valeur d'une variable mathematique
- *ret --> vider une variable mathematique
- *add --> addition de reels
- *sub --> soustraction de reels
- *mult --> multiplication de reels
- *div --> division de reels

Une variable mathematique n'est pas instanciee comme les autres mais affectee et manipulee par des fonctions mathematique.

Si on ajoute dans la base de faits, un terme tel que : X = 3, une variable mathematique de nom X est creee dans l'environnement, et initialisee a 3.

Attention : lors de l'utilisation de fonctions dans les regles et meta-regles, il ne faut pas que la partie action ne soit constituee que d'appels a des fonctions. Dans ce cas, meme si la base de faits le permet en theorie la regle ne sera pas activee. On impose une telle restriction pour eviter des bouclages du raisonnement difficiles a deceler. Il faut utiliser au moins un ajoutfait qui donne un test d'arret au systeme.

De meme, des regles ne comportant que des fonctions en partie premisses ne sont pas prises en compte dans une base de connaissances compilee.

Les fonctions utilisees dans les regles doivent obligatoirement utiliser la fonction "quote" pour quoter les elements a ne pas evaluer. Par exemple:

(print "a b")

doit s'ecrire dans une regle (print (quote "a b"))
ceci afin de conserver lors des sauvegardes des regles les

notions des chaines non evaluables.

Il faut prendre soin de bien quoter les variables et parametres voulus dans les appels de fonctions utilises. Une evaluation incorrecte d'une fonction lors de la manipulation d'une regle fait echouer tout le raisonnement. Comme le symbole "'", le macro caractere ":" ne peut pas etre employe.

Les commandes d'ajouts de telles fonctions ne font qu'enregistrer le nom d'une nouvelle fonction si il n'a pas deja ete utilise. Elles obligent l'utilisateur a rentrer le code des fonctions dans des fichiers speciaux qui auront ete definis auparavant pour contenir specialement les definitions de ces fonctions.
Pour les fonctions internes, fichier fct.ll (variable **listefct)
Pour les fonctions externes, fichier fctext.ll
(variable **listefctext).
Ces fichiers doivent etre utilises pour permettre lors d'un chargement du systeme a partir d'une image memoire de se retrouver dans un etat complet et coherent.

Les fonctions utilisees dans les regles doivent etre enregistrees dans l'un de ces 2 fichiers pour etre acceptees par le systeme.

Pour les ajouts et modifications de fonctions, on travaille dans le fichier voulu avec l'editeur pleine page "pepe" .

Commandes de manipulation des fonctions :

- ajfct
  ajout ou modification d'une fonction interne
  Sans parametre : modifications des fonctions existantes

- spfct
  supprimer une fonction interne
  1 parametre = nom de la fonction

- prlfct
  liste des fonctions internes creees

- defext
  definir ou modifier une fonction externe
  Sans parametre : modifications des fonctions existantes

- spfext
  suppression d'une fonction externe
  1 parametre = nom de la fonction

- prlfext
  liste des fonctions externes definies

MANIPULATION DES REGLES

La base de regles peut aussi etre manipulee dynamiquement par le systeme lui-meme en cours de raisonnement.
Un type de modification de ce genre a ete programme avec la fonction mco qui peut etre appelee dans les regles et permet de modifier le coefficient de vraisemblance d'une regle : mco numero-regle coeff .
Ce principe peut etre etendu pour obtenir une base de regles dynamique et un systeme capable de generer, modifier ou detruire ses connaissances. Dans ce but il faut coder les fonctions necessaires, comme ici mco, qui seront appelees dans les regles. On peut utiliser a cette fin CRIQUET comme un langage de programmation .

Commandes de manipulation de la base de regles :

- pr
  imprime la regle de numero identificateur n donne en reponse a une interrogation du systeme

- prcoh
  impression des regles de coherence qui sont dans la base de regles courante

- prc
  impression de regles obtenues par une recherche sur les coefficients de vraisemblance . On peut rechercher les regles qui ont un coefficient donne, superieur ou inferieur a une borne fixee.

- prl
  imprime une liste de regles a partir d'une liste de numeros donnee par l'utilisateur

- br
  imprime toute la base de regles

- mdr
  pour modifier une regle
  On travail avec l'editeur pleine page pepe .
  Voir manuel Le_lisp.
  Lors de la modifications de regles, le systeme verifie le resultat rendu apres modifications pour verifier le format interne. S'il y a une erreur, il peut replacer l'utilisateur en edition dans le fichier concerne pour permettre la correction.

- mdrco
  pour ne modifier que le coeff de vraisemblance des regles

- spr
  supprime de la base de regles la regle de numero n

- spbr
  pour supprimer toute la base de regles

- ajr
  ajoute une regle dans la base de regles .
  On peut typer la regle comme on le desire, par defaut
  les 3 types existants sont : inference, coherence ou
  equivalence. Des variables peuvent etre employees dans
  les regles. Le nom des variables est prefixe par un ? .
  Les parties condition et action sont demandees en
  interactif a l'utilisateur ainsi que les autres constituants .
  On peut indiquer dans les premisses des restrictions
  sur les variables, avec le mot cle : predicat .
  ex : SI (T < ?T1)
       ET (T < ?T2)
       ET (predicat : ?T1 < ?T2)
  Pour chaque action il faut preciser si c'est un ajoutfait
  suppresfait , suppresbut ou fonction. Ce dernier mot cle
  est a utiliser si on veut appeler une fonction definie
  dans l'environnement LISP.
  Mais il y a quelques restrictions a respecter (voir ch
  sur les fonctions.)
  Des fonctions peuvent etre appelees en premisses. Dans ce cas
  elles doivent rendre t ou nil. Si on veut indiquer comme
  action la suppression de A et l'ajout de B on ecrit :
        ((suppresfait(A))(ajoutfait(B)))
  Pour une regle de coherence la partie action n'est pas
  demandee, elle se reduit a : (erreur) .
  Pour plus de precisions sur le formalisme des regles,
  voir les informations sur 'formalisme'

- req
  Pour generer la regle equivalente aux regles dont on
  donne les numeros.

- rcp
  Pour generer la contraposee des regles dont on donne
  les numeros.

GESTION DE COHERENCE DE LA BASE DE REGLES

Une gestion de la base de regles peut aussi etre obtenue avec la commande testbr. On maintient la coherence de la base de regles courante dans son ensemble et non pas a l'ajout de chaque regle. L'utilisateur peut ainsi travailler avec ou sans gestion de coherence comme il le desire.

La coherence des regles est assuree avec des meta-regles.
A une meta-regle particuliere pour la gestion de coherence correspond un type de test.
Il y a 8 types de tests predefinis : redondance, conflit total, conflit partiel, generalisation et 4 tests semantiques.
Ces derniers types de tests utilisent les regles de coherence utilisees pour la gestion des faits.

Pour plus de precisions voir le rapport sur la gestion de coherence dans la base de regles de CRIQUET.

- coh
  pour specifier une liste de faits incompatibles avec un fait donne en premier lieu .

- avecbrcoh
  avec un parametre = un nom d'entite utilisee pour sauvegarder une base de regles de coherence.
  Pour indiquer la source a utiliser pour les tests de coherence.

- testbr
  pour verifier la coherence de la base de regles dans son ensemble, en choisissant les tests a utiliser.

RECHERCHES DANS LES REGLES

- rhp
  Pour chercher dans la base de regles courante des regles ayant un ou plusieurs mots donnes dans leurs premisses . Ces regles sont chargees dans un fichier de nom donne

- rha
  meme fonctionnalite que rhp mais avec une recherche dans la partie action des regles

- rht
  recherche des regles selon un type donne

- rhc
  recherche des regles de coherence

- rhi
  recherche des regles d'inference

- rhfa
  recherche d'une forme dans les actions des regles .
  Une forme est un quadruplet du langage : objet predicat att valeur, prefixe par un mot cle s'il y a lieu .
  Ainsi si on recherche un ajoutfait particulier, la forme a utiliser est : ajoutfait : -- -- -- -- .
  Seul le predicat ne peut pas etre quantifie.

- rhfp
  recherche d'une forme dans les premisses des regles.
  Une forme est un quadruplet du langage : objet predicat att valeur, prefixe par un mot cle s'il y a lieu.
  Seul le predicat ne peut pas etre quantifie.

MANIPULATION DES META-REGLES

On peut contruire des meta-regles qui sont placees dans une base particuliere : **metarules . Ces meta-regles peuvent avoir en partie action des selections de regles objet par des recherches sur leur type ou sur un ou plusieurs mots dans les premisses ou actions. Les recherches peuvent aussi porter sur une forme a identifier dans les premisses ou les actions. Voir commandes : rha, rhp, rht, rhfa, rhfp.
Le mot cle "rhnb" permet de selectionner des regles sur leur numero.
Les meta-regles permettent aussi de deconseiller des regles (nrha nrhp), regles qui sont aussi selectionnees par des recherches de mots dans leurs premisses et actions.
Une meta-regle peut aussi entrainer l'ajout ou la suppression d'un fait dans la base de faits (ajoutfait, suppresfait).

Des fonctions peuvent aussi etre appelees en partie condition et action, comme dans les regles objets. Des predicats peuvent etre utilises comme dans les regles.

Si plusieurs commandes de recherches constituent la partie action d'une meta regle celle-ci rend la reunion des resultats des differentes recherches .
On peut utiliser de facon optionnelle les meta-regles disponibles. Les commandes avecmeta et sansmeta precisent ce point .
Si on les utilise , un metamoteur au debut de chaque cycle selectionne et active, si possible, une meta-regle ce qui a pour effet de reduire l'ensemble des regles objet manipulees .

La restriction de la base de regles ne doit pas etre definitive, du fait des possibilites de backtrak du raisonement.
Afin de resoudre ce probleme, sans avoir a memoriser les bases de regles apres chaque cycle, on decide de restaurer la base de regles initiale a chaque mouvement de backtrak.

Les meta-regles peuvent aussi etre employees comme structure de controle. Ceci est alors specifie avec la commande chx

Les variables peuvent aussi etre employees dans les meta-regles.
Les commandes disponibles pour la manipulations des meta-regles sont proches de celles utilisees pour les regles.

Commandes de manipulation des meta-regles :

- pmr
  impression d'une meta-regle

- bmr
  impression de la base de meta-regles

- metam
  active un cycle de metamoteur de maniere independante

- ajmr
  creation d'une nouvelle meta-regle
  Le formalisme a respecter est le meme que pour : ajr .
  Ici en partie action les mots cles sont : rha rhp rht rhfa
  rhfp rhnb nrha nrhp ajoutfait suppresfait fonction.
  Les 5 premiers mots correspondent a des commandes.
  (Faire help dessus)
  "rhnb" permet une recherche de regles a partir de leur numero.
  "nrha" et "nrhp" pour deconseiller des regles obtenues par
  recherches de mots dans leurs premisses et actions.
  "ajoutfait, suppresfait et fonction" sont a utiliser comme dans
  les regles.

- mdmr
  modification d'une meta-regle
  comme la commande mdr pour les regles objets

- spmr
  pour supprimer une meta-regle

- spbmr
  pour supprimer la base de meta-regles

UTILISATION DES META-REGLES

Langage d'expression

En partie premisse, on peut utiliser tous les termes utilisables dans les regles: quadruplets, termes prefixes par fonction, predicat et sachant, termes predicatif et des sous listes imbriquees.

Mais, certaines conditions doivent pouvoir porter sur les regles objets elles meme. Pour cela, il faut pouvoir designer, rechercher et acceder des regles . Ceci est possible avec l'expression reservee "regle ?..." qui permet de designer une regle avec la variable choisie.

Exemple :

```
meta  1
 SI regle ?x
 ET metal etre or
 ET fonction : member (quote (metal etre precieux))
                      (quote (si (quote ?x)))
ALORS fonction : rst ?x
```

Cette meta-regle permet de restreindre l'espace de recherche a la regle, designee par la variable ?x, qui verifie certaines conditions.

Cette possibilite n'est pas utilisable par defaut. Il faut pour cela utiliser dans le module "moteur.ll" la fonction "detav**" au lieu de "detav".

Politiques d'utilisations

De nombreux problemes susistent quant a l'utilisation des meta regles, sans meme parler de leur specification encore plus complexe selon les domaines.

Il y a 3 types utilisations essentielles des meta-regles :
- pour manipuler les connaissances de base
- pour evaluer la coherence des connaissances
- pour generer des nouvelles connaissances

Les meta-regles permettant de manipuler les connaissances de base servent :

- a restreindre l'espace de recherche au debut de chaque cycle elementaire, de facon a limiter l'ensemble des regles sur lesquelles va se faire l'operation de detection qui est couteuse. Une politique d'utilisation des meta-regles a ete choisie dans ce cas. A chaque cycle elementaire, les meta-regles manipulent la totalite de la base de regles et non pas que la partition creee au cycle precedent. Si dans le cycle courant, la partition constituee avec la meta-regle choisie ne contient pas de regles activables, alors le systeme repete l'operation de detection sur toute la base de regles sans plus se preoccuper des meta-regles.
Cette politique evite l'arret total du cheminement en cas de mauvaise specification des meta-regles.

- a indiquer un plan d'actions, a manipuler avec les primitives au sujet de l'agenda.

- au controle, c'est a dire pour choisir la regle a activer parmi l'ensemble des regles activables.
Dans ce cas, les meta-regles sont utilisees de facon non exhaustive et sans backtrak, c'est a dire que si la meta-regle choisie pour le controle ne permet pas de selectionner precisemment une regle a activer, le systeme choisit par defaut la permiere regle parmi les regles activables, sans chercher a utiliser une autre meta-regle.

Un controle correct repose donc sur la bonne specification des meta-regles.

Dans tous les cas, ces politiques d'utilisations sont a adapter aux besoins particuliers (dans le module "moteur.ll", les fonction "stepf" et "choix5").

Il est a noter que dans CRIQUET, la manipulation des meta-regles ne peut se faire que de facon interpretee (pas de compilation de la base de meta regles).

Des meta-regles sont aussi utilisees dans CRIQUET pour la gestion de coherence des regles (voir rapport de recherche INRIA 380).

La generation de connaissances est traitee par la suite, paragraphe "apprentissage", mais sans utiliser de meta-regles.

Au sujet de l'utilisation des meta-connaissances, il manque encore des essais et de l'experience pour decider si c'est possible des meilleuers techniques d'utilisation de ces outils.

MANIPUATION DES INDICATEURS

Les differents indicateurs sont :
rchex nrchex
avecmeta sansmeta
avecbutord sansbutord
avecgsem sansgsem
avecglog sansglog
avecomment sanscomment
avecbrdyn sansbrdyn
avectrace sanstrace
avecint sansint
avecnon

Pour plus de precisions taper help sur chaque commande.

- rchex nrchex : types de recherches
  choisir entre une recherche exhaustive ou non

- avecmeta
  pour indiquer l'utilisation des meta-regles

- sansmeta
  pour indiquer la fin d'utilisation des meta-regles

- avecbutord
  pour preciser au systeme que l'on veut respecter l'ordre donne des buts partiels . Un but partiel n'est supprime de la base de buts que si une instruction suppresbut est activee dans la partie action d'une regle.

- sansbutord
  L'ordre des buts partiels n'importe pas . Un but partiel est supprime s'il apparait dans la base de faits ou si une instruction suppresbut est activee .
  Par defaut on travaille avec les buts partiels ordonnes.

- avecgsem sansgsem
  pour indiquer repectivement si l'on veut travailler avec ou sans gestion semantique des faits

- avecglog sansglog
  pour indiquer repectivement si l'on veut travailler avec ou sans gestion logique des faits

- avecomment sanscomment
  pour indiquer repectivement si l'on veut travailler avec ou sans les commentaires attaches aux regles

- avecbrdyn sansbrdyn
  pour indiquer si on veut ou pas avoir une base de regles dynamique. Dans ce dernier cas, des regles peuvent en cours de raisonnement modifier la base de regles. Cette possibilite est consideree pour les possibilites de retour arriere lors des differents cheminements (arriere avant)

- avectrace
  Cette commande activee avant de lancer un raisonnement permet de travailler en mode trace . Avec un parametre (par exemple 1), on a une trace partielle et le systeme ne fait que donner a chaque cycle le numero de la regle activee. Sinon, durant le cheminement, quelqu'il soit, le systeme , a chaque cycle, expose les regles activables et cite la regle choisie .
  L'utilisateur peut confirmer le choix ou le refuser (oui-non). Il peut aussi demander l'explication du raisonnement tenu jusqu alors : exp . Il peut reinitialiser le dernier cycle de raisonnement: rappel. Il peut modifier la strategie de choix : chx. Il peut stopper completement le cheminement en cours avec "stop".
  Enfin, il peut rajouter des faits dans la BF : ajf, ou modifier le coefficient de certains : mdf .
  D'ou la liste des reponses possibles :
  oui non rappel stop exp chx ajf mdf
  Si on passe 0 en parametre, une trace partielle est utilisee. Le systeme alors a chaque cycle cite le numero de la regle utilisee sans donner le controle a l'utilisateur.

- sanstrace
  pour terminer un travail en mode trace

- avecint
  Pour utiliser le module d'interface en langage pseudo-naturel lors de chaque saisie d'informations (regles et faits).
  Si l'utilisateur ne precise rien, le systeme l'interroge une fois au sujet de l'utilisation ou pas du module. La reponse donnee est conservee comme valeur par defaut jusqu'a ce que l'utilisateur modifie sa decision avec les commandes "avecint" ou "sansint".

- sansint
  Pour ne plus utiliser le module d'interface.

- avecnon
  1 parametre = 1 ou 2
  commande qui permet de choisir le type de negation a utiliser:
  parametre 1 = (non A) est vrai si (non A) apparait dans la base de faits (Univers clos).
  parametre 2 = (non A) est vrai si (A) n'apparait pas dans la base de faits.
  Par defaut c'est le premier type qui est utilise.
  (Cette fonction n'est utilisable que dans la derniere version, CRIQUET avec une base d'objets).

LE RAISONNEMENT

- chav
  active le moteur d'inference en cheminement avant .
  Le raisonnement se fait avec la base de faits disponible .
  On peut donner un fait but a atteindre et le raisonnement
  s'arrete si le but precise est atteint sinon le raisonnement
  s'arrete quand il n'y a plus de faits deductibles. Ce but
  final donne peut contenir des variables et ainsi representer
  une forme a atteindre.

  Si des buts partiels sont precises dans la base de buts ,
  ils sont utilises en respectant leur ordre ou non selon
  la volonte de l'utilisateur.

  Il y a backtrak si on rencontre une incoherence ou tant que
  le but donne n'est pas atteint, sauf en version compilee
  de la base de connaissances. Dans ce cas aucun retour arriere
  n'est possible.

  Sans but donne, le chainage se poursuit jusqu'a epuisement
  de l'ensemble des regles activables.

  Une recherche exhaustive est possible. Mais dans ce cas, afin
  d'obtenir en dernier lieu une seule base de faits globale, on
  n'utilise a chaque cycle que les actions 'ajoutfait' des regles activees.
  Sans cette restriction, a cause des possibilites de suppressions
  de faits, il faudrait pouvoir considerer une base de faits
  differente pour chaque parcours possible.

- charr
  active le moteur en cheminement arriere . On peut donner un
  fait particulier a diagnostiquer . Sinon le systeme essaie de
  diagnostiquer l'un des faits qu'il connait comme deduction
  finale possible d'un raisonnement. A chaque question du systeme
  l'utilisateur peut repondre par un ? pour que le systeme lui
  justifie la question ou par ?? si il ne sait que repondre.
  Le systeme ne travaille en univers clos qu'avec l'accord de
  l'utilisateur.
  C'est a dire que le systeme ne memorise la negation d'un fait
  qu'avec l'accord de l'utilisateur.

  Le chainage arriere peut se faire avec des regles
  avec variables . Mais il ne doit pas y avoir de variables libres
  c'est a dire que les buts intermediaires utilises ne doivent
  pas contenir de variables. (voir paragraphe "langage d'expression
  utilisable" au sujet des termes prefixes par "sachant" au debut
  du manuel).

  Le but initial, si il est donne, peut par contre contenir le symbole
  "?" ou des variables. On peut meme donner plusieurs buts initiaux
  a traiter.

COMPILATION DE LA BASE DE CONNAISSANCES

- compilation, initialisation

La commande compilation permet de compiler la base de connaissances. Celle-ci est alors figee, mais les performances du systeme sont ameliorees. La commande initialisation permet d'initialiser la base compilee avec la base de faits initiale.
Si on veut utiliser plusieurs fois la meme base compilee avec des bases de faits initiales differentes, il faut sauvegarder la base compilee avant le premier appel a initialisation (commande sauvbr puis chargbr) .
On recupere cette base compilee vierge pour chaque nouvelle base de faits initiale.

Toutes les bases de regles ne peuvent pas etre compilees. Certains termes en partie premisses des regles ne peuvent pas etre compiles. Des termes classiques sont utilisables tels que:

- avec variables syntaxiques et mathematiques
- avec la negation
- les termes predicatifs
- les termes prefixes par "fonction" , "predicat" et "sachant"

Mais le systeme (en version classique) ne sait pas compiler les appels internes de fonctions dans un quadruplet (ex/ (bp < (*div moyenne 5))). Ces appels doivent alors etre utilises dans des termes prefixes par "predicat" ou "fonction".

Pour etre compile, le terme donne en exemple doit etre traduit :
ex/ pour le terme precedent on utilise 2 termes (bp = ?x) et (predicat (?x < (*div moyenne 5))).

De plus l'unification des termes predicatifs en version compilee n'est que syntaxique. Par exemple, en version interpretee (i = 8) match avec (i > 5) et (i < 15), mais pas en compile. La encore une traduction est necessaire telle que :

```
     SI i > 5                              SI i = ?x
     ET i < 15   doit etre traduit         ET predicat (?x >5)
ALORS ....             en                  ET predicat (?x < 15)
```

Ces traductions sont a soigner et peuvent etre difficiles.

Une version speciale du module de compilation (module compilation_spec.ll) permet une compilation directe de tous les termes, meme les termes predicatifs et avec sous-listes. Mais cette technique est plus lente que la premiere.

Pour plus de precisions voir le rapport sur la compilation de la base de connaissances dans CRIQUET.
(ref biblio sur Algo RETE, OPS5, PSC, TANGO)

PROBLEMES AU SUJET DU BACKTRAK

Dans certains cas, les possibilites de retour arriere correct sont impossible car cela necessiterait la sauvegarde de la base de connaissances complete a chaque appel a la fonction recursive qui realise le cheminement.

Pour eviter la saturation de l'espace, on ne sauvegarde que la base de faits a chaque appel et une fois la version initiale de la base de regles.
Cette sauvegarde de la base de regles permet de toujours pouvoir revenir dans un etat coherent.

Des restrictions proviennent de ces techniques:

- retour arriere impossible en chainage avant avec une base de connaissances compilee.

- lors d'un chainage avant avec une base de regles dynamique, c'est a dire qui peut etre modifiee en cours de raisonnement, on ne restaure que la base de regles initiale en cas de retour arriere.
  On ne memorise pas les differents etats de la base de regles.

Le chainage arriere fonctionne avec un backtrak complet et correct dans tous les cas.

Le retour arriere (backtrak) pratique par defaut n'est pas chronologique comme le fait Prolog, c'est a dire que lors du retour arriere le systeme ne choisit pas simplement la premiere regle activable qui reste non encore utilisee en ce point de decision.

Le systeme execute une nouvelle operation de choix sur ces regles qui restent activables. Mais, par defaut il utilise lors de cette operation la meme strategie de choix que celle utilisee precedemment. On peut revoir cette politique et indiquer au systeme une autre strategie a utiliser uniquement lors de choix a faire en cas de backtrak.

PROBLEMES AU SUJET DE LA RECHERCHE EXHAUSTIVE

En recherche exhaustive, en chainage avant et arriere, toutes les regles activables sont utilisees.

En chainage avant, une version complete de la recherche exhaustive peut amener a generer plusieurs bases de faits differentes selon les parcours. Pour eviter cela, impossible dans un systeme de production classique, on n'utilise en recherche exhaustive que les actions "ajoutfait" des regles activees.

De ce fait, tous les parcours aboutissent a la meme base de faits. Il faut donc faire attention lors de l'utilisation de la recherche exhaustive a la validite du raisonnement tenu et de la base de faits generee.

Lors d'une recherche exhaustive, pour faciliter la gestion de coherence de la base de faits si l'utilisateur l'utilise, un parcours en "profondeur d'abord" est utilise pour explorer l'arborescence de decisions. Mais, la encore, cette option peut etre modifiee si l'application le necessite.

GESTION DE LA CIRCULARITE EN CHAINAGE ARRIERE

Le systeme lors du chainage arriere gere une liste des buts intermediaires deja traites pour eviter un raisonnement qui boucle possible avec des regles telles que :

si a alors ajoutfait : b
si b alors ajoutfait : a

Si le but intermediaire traite fait partie de la liste des buts deja traites, le systeme stop le raisonnement et fait un retour arriere.

CHOIX FORCE EN CHAINAGE ARRIERE

En chainage arriere la phase de detection rend les regles dont les parties actions comporte le but intermediaire traite.

Il est possible que aucune de ces regles ne soient directement activables dans la base de faits disponible. C'est le cas si un ou plusieurs elements des premisses n'apparaissent pas dans la base de faits. De ce fait les structures de controle, telles que les fonctions de choix 1, 2 et 3 peuvent ne pas trouver de regles a activer dans l'ensemble de conflit pourtant construit correctement. Dans ce cas, a cause d'un manque d'information et parce que le controle choisi est inefficace, le systeme impose un choix simple qui est d'utiliser la premiere regle activable.

TRAVAIL AVEC TRACE : FACILITES DE MISE AU POINT

Diverses facilites de travail sont offertes en particuliers pour :

+ mettre au point une base de connaissances, voir les commandes avectrace, de sauvegardes, les tests de de coherence et les explications par niveaux du systeme .

+ sauvegarder un historique de travail avec les commandes sauve et fin

- sauve
  permet de faire se derouler une session de travail en audit sous Multics . Ainsi une trace de toutes les manipulations faites est conservee. (uniquement sous Multics)

- avecsauve
  permet la meme chose que la commande precedente (sauve), mais sans utiliser d'ordres Multics, le tout code en Le_Lisp. Commande non encore utilisable en Version V15.

- fin
  permet de fermer le fichier trace qui contient alors les traces des manipulations que vous avez fait durant la derniere session si vous avez utilise sauve. La commande fin permet aussi de sauvegarder tout l'environnement de travail dans un fichier nomme criquet.(utilisable que sous Multics)
  Pour sortir ensuite taper end.

- finsauve
  meme chose que la commande "fin", mais correspondant a la commande "avecsauve". La encore, cette commande est indepéndante du systeme hote, mais non encore utilisable en V15.

EXPLICATIONS

- expl
  explique le raisonnement du systeme a partir des regles connues .
  Il y a differents niveaux d'explications : 0, 1, 2.
  On peut avoir une explication complete, c'est a dire avec trace de tous les essais meme infructueux et des retours arriere (niv 2).
  Ou on peut n'avoir qu'une explication du parcours avec succes.
  Il suffit dans le premier cas de passer 2 en parametre a expl .
  Avec le parametre 0, le chemin qui a mene au succes est peu detaille, un peu plus avec le parametre 1.

  Dans les 3 cas on peut choisir de travailler avec ou sans les commentaires attaches aux regles s'il y en a : commandes avecomment sanscomment.

  Dans tous les cas, on peut demander des explications sur l'obtention d'un fait precis, dont on donne alors le texte en second parametre a la commande expl.

  Ce dernier outil ne fonctionne pas correctemment si les regles concernees utilisent des termes avec des sous listes imbriquees.

STRUCTURE DE CONTROLE

- chx
  pour choisir entre 7 strategies disponibles en phase de choix
  .la premiere simple consiste a choisir la premiere regle activable
  .4 strategies de choix par evaluation :
    -choisir la regle de plus fort coefficient
    -choisir la regle qui utilise les faits les plus recemment deduit
    -choisir la regle qui utilise les faits les plus importants
    -choisir la regle qui satisfait un but partiel

  .1 strategie avec meta-regles :
  choisir le controle par meta-regles revient a specifier
  une strategie a utiliser pour selectionner une meta-regle
  qui elle meme permettra de choisir la regle objet a activer.

  .la derniere strategie (6) permet de fonder le choix de la regle
  a activer sur l'evaluation des etats accessibles. Il faut donner
  au systeme le nom de la fonction d'evaluation a utiliser. Le
  systeme choisit alors comme regle a activer la regle qui permet
  d'atteindre le meilleur etat au sens de cette fonction d'evaluation.
  Cette fonction a ete au prealable definie par l'utilisateur avec
  "ajfct" (attention au codage). Pour l'instant, les fonctions utilisees
  ne peuvent porter que sur la base de faits. Mais ceci est a adapter
  aux cas traites.
  Cette derniere technique n'est utilisable que dans la derniere version
  du logiciel (CRIQUET avec une base d'objets).

  Cette commande peut etre utilisee avec parametres et appelee
  dynamiquement par les regles activees. Il y a dans ce cas 2
  parametres : le premier indique la strategie a utiliser
               le second indique la strategie a utiliser pour les
               meta-regles si la strategie est la 5ieme .

  Les possibilites de choix de la structure de controle peuvent
  amener a faire des erreurs de specifications. Il est difficile
  en particulier en chainage arriere de specifier un controle
  correct et efficace. Afin d'eviter le blocage du raisonnement
  dans certains cas, le systeme peut forcer le choix d'une regle
  au cas ou la structure de controle donnee est incorrecte.
  On conseil pour le chainage arriere d'utiliser les structures
  de controle 1,4 ou 5.

  Ce probleme s'apparente a celui de l'apprentissage des systemes.

  De plus, certaines fonctions sont beaucoup plus couteuses en temps
  d'execution que d'autres. Ainsi la fonction choix2 est en moyenne
  6 fois plus lente que la fonction choix1.
  Il faut donc aussi bien evaluer la relle efficacite des techniques
  de choix selon les applications visees.

RAISONNEMENT APPROXIMATIF

Les faits et les regles sont ponderes d'un coefficient de vraisemblance entre 0 et 1 . Lors de deductions la combinaison de ces coefficients est inspiree de celle utilisee dans MYCIN . Pour un fait 'b' deduit avec 'regle' constituee de 'premisses' :

$$CV(b) = CV(regle) * Min\ CV(premisses)$$

Si le fait 'b' est deduit 2 fois sous forme de 'b1' et 'b2', il y a confirmation du fait et pour cela augmentation de son coefficient d'importance :

$$CV(b) = CV(b1) + CV(b2) - CV(b1) * CV(b2)$$

SAUVEGARDES

On peut sauvegarder et charger tous les types d'ensembles manipules : br bmr bf bh bb contexte et agenda.
Des facilites particulieres sont offertes pour une sauvegarde rapide sans avoir a donner de nom pour les br et bf courantes.

Commandes de sauvegarde :

- sauvbfc
  sauvegarde de la base de faits courante sans donner de nom

- sauvbrc
  sauvegarde de la base de regles courante sans donner de nom

- resbf
  permet de restaurer la derniere base de faits utilisee

- resbr
  permet de restaurer la derniere base de regles

- sauvbf
  pour sauvegarder le contenu de la base de faits courante dans un fichier dont on donnera le nom

- sauvbr
  pour sauvegarder le contenu de la base de regles courante dans un fichier dont on donnera le nom
  Sauvegarde aussi de la version compilee si elle existe.

- sauvbmr
  pour sauvegarder la base de meta-regles courante

- sauvbb
  pour sauvegarder la base de buts courante

- sauvbh
  pour sauvegarder la base d'hypotheses courante

- brs
  pour lister le contenu d'un fichier de regles construit avec les commandes rhp rha rht rhi rhc chargbr

- bfs
  pour lister le contenu d'un fichier de faits

- rst
  1 parametre possible : soit un nom de fichier de regles,
  soit une liste de regles en extension ou une liste de numeros
  de regles.
  Pour restreindre la base de regles courante . Cette commande
  permet de charger dans la base de regles courante un ensemble
  de regles determine avec rhp rha rhfa rhfp ou rht .
  Ceci peut accelerer le raisonnement.

- pnbf
  liste la liste des noms de fichiers deja utilises pour
  sauvegarder une base de faits

- pnbr
  liste la liste des noms de fichiers deja utilises pour
  sauvegarder une base de regles

- pnbmr
  pour lister les noms utilises pour sauvegarder des bmr

- pnbb
  pour lister les noms utilises pour sauvegarder des bb

- pnbh
  pour lister les noms utilises pour sauvegarder des bh

- spnom
  premier parametre : le nom a supprimer
  second parametre : la liste de noms a utiliser
  Pour supprimer le nom voulu dans une liste de noms.
  Pour avoir la liste des noms de listes utilisees par
  le systeme taper help sur environnement.

- ajnom
  premier parametre : le nom a ajouter
  second parametre : la liste a utiliser
  Pour ajouter le nom voulu dans une liste de noms.
  Pour avoir les listes utilisables taper help sur
  environnement .

- chargbf
  pour charger dans la base de faits courante le contenu du fichier
  dont on donnera le nom

- chargbr
  pour charger dans la base de regles courante le contenu du
  fichier dont on donnera le nom

- chargbmr
  pour charger une bmr dans la base courante

- chargbh
  pour charger une bh dans la base courante

- chargbb
  pour charger une bb dans la base courante

- ruf
  1 ou 0 parametre : le nom de l'entite de sauvegarde a utiliser
  Permet de faire la reunion d'un fichier de faits de nom donne avec
  la base de faits courante. Le resultat est dans la base de faits.

- rur
  2 ou 0 parametres : le nom de l'entite de sauvegarde a utiliser
  et le type de contenu : br ou bmr.
  Permet la reunion d'un fichier de regles ou de meta-regles
  avec la base courante. Le resultat est dans la base de regles.
  On utilise un processus d'unification pour cette tache afin
  d'eviter la perte de regles lors de la reunion . Mais si on
  a une regle generale et certaines de ses specialisations
  obtenues par instanciations, on perd la regle generale pour
  garder toutes les specialisations .

- sauver
  pour sauvegarder un ensemble choisit (bf bb bh ag br bmr bo hist)
  dans un fichier de nom donne, hist pour historique.
  On peut donc memoriser les explications du systeme au sujet
  d'un cheminement.
  Si on utilise un fichier deja cree, le contenu precedent est
  ecrase .

- charger
  pour charger un fichier de nom donne par l'utilisateur.
  Ce fichier peut concerner un ensemble particulier (bf br bmr bo)
  ou autre chose (autre).

- edit
  pour rentrer en edition pleine page (pepe) dans le fichier
  dont on a donne le nom.

- end
  pour sortir du systeme sans sauvegarde aucune

- menu
  pour avoir la liste des commandes disponibles

- info
  pour avoir des informations sur l'ensemble du systeme

- help
  Pour avoir des informations sur un sujet precis ou une commande

DIVERS ET GESTION D'ECRAN

Une gestion d'ecran est automatiquement faite. Le defilement s'arrete en fin d'ecran. Les commandes disponibles sont alors:

- "a" (aborted) pour arreter le defilement et retourner au langage de commande de criquet.
- "=" pour avoir le numero de la ligne courante
- "?" pour avoir la liste des commandes disponibles
- SP pour continuer le defilement et avancer d'un ecran
- RC pour avancer d'une ligne

La cle "^G" (pour CTRL G) permet de retourner sous criquet a partir de Le_lisp.

Dans le langage de commande, la commande "**" utilisee en premier lieu permet d'executer ensuite n'importe quelle commande Lisp ou autre.

PORTABILITE

Le code est totalement portable sur tous les sites ou est implemente Le_Lisp version 15.
Des problemes peuvent neanmoins apparaitre dus a:

- la description du terminal necessaire pour la banniere et l'utilisation de l'editeur pleine page pepe.
  Si elle n'est pas deja connue du langage Le_Lisp, la description dois etre donnee dans un fichier special.
  Voir manuel Le_Lisp.
- l'utilisation de l'editeur pepe .
- l'utilisations des Defextern de Le_Lisp non implementees sur tous les sites.
- l'utilisation de commandes du systeme hote avec des comline de Le_Lisp : dans les commandes sauve et fin (que sous Multics) du module modnport (selon la version de criquet).

Pour le transport du code, creer les fichiers :
- description de terminal si non deja connu de Lelisp

Dans le version V15 du langage, un grand nombre de terminaux sont reconnus .

GESTION MEMOIRE : OVERLAY

(Description approchee car en evolution)

L'ensemble du logiciel a ete decoupe en 32 modules de code, (nombre variable selon la version) constituant une hierarchie. D'autres modules contiennent du texte (56 modules en tout) . La racine de la hierarchie est le noyau du systeme et doit etre maintenu en memoire. Les branches, constituees de modules independants, representent diverses fonctionnalites.

L'ensemble du logiciel source represente environ 340K, soit 10000 lignes de Lisp. Au total, avec les textes 430 K
La racine est d'environ 230K.
Si on ne peut pas tout conserver en memoire centrale, a chaque appel a 1 commande situee dans une branche, il faut charger le code porte par cette branche, apres avoir efface de la memoire suffisament de code pour liberer la place necessaire.
Ce systeme d'overlay est utilisable et necessaire en particulier pour des micro ordinateurs sans pagination.
Il est automatiquement utilise avec les 90K de texte, pour l'aide en ligne. On evite ainsi de saturer le zone des listes
Le chargement automatique est obtenu avec la fonction Autoload modifiee de Le_Lisp (voir manuel) .
La gestion de l'espace memoire est faite avec des tables pour gerer les noms des fichiers presents en memoire.
Si on desire utiliser cet overlay, il suffit de charger le module nomme overlay.ll .

Hierarchie de modules :

```
            chnoyau.ll top.ll initfile.ll
            reinit.ll  pr.ll
            modnport.ll lgc.ll
            expl.ll  menu.ll informations.ll
            moteur.ll match.ll stk.ll
            mpbf.ll mpbr.ll mph.ll              overlay.ll
            fct.ll fctext.ll                    chrestant.ll
            coherencefait.ll interface.ll       description
                                                  terminal
                                                criquet.out
                                                criquet_dupl.ll

 coherencebr.ll  metamoteur.ll  compilation.ll  informations.ll
 bmrcoh1.ll                                     info1.ll
 bmrcoh2.ll        mpbmr.ll  mpbb.ll  mpag.ll   info2.ll
 bmrcoh3.ll                                       .
 bmrcoh4.ll          mpcont.ll    mpfct.ll        .
 bmrcoh5.ll                                       .
 bmrcoh6.ll             apprentissage.ll
 bmrcoh7.ll
 bmrcoh8.ll                                     info14.ll
```

Cette gestion memoire en soft fonctionne mais ralentie trop le deroulement d'ensemble pour etre d'utilisation efficace.

Le noyau est constitue de 20 fichiers plus le fichier qui contient la description du terminal. La taille maximun du noyau est d'environ 230K, de source et sans aucune optimisation de place. La place necessaire pour le source peut etre reduite rien qu'en compactant le texte et en eliminant les commentaires.

Les espaces occupes :

NOYAU	RESTANT	
20 modules	10 modules + 22 modules de texte	
240 K	100 K	90 K

Nombre de cellules lisp occuppees :

NOYAU	TOTAL (sans texte)
version interpretee du code	
28579 cons	35717 cons
832 symbol	974 symbol
1568 string	1820 string
10 heap	11 heap
version compilee du code	
resultats non encore obtenus. Configuration trop petite pour compiler. Le nombre de cons devrait descendre autout de 17000	

PERFORMANCES

Divers laboratoires prives et publiques utilisent CRIQUET et aident ainsi a le debugger et a le mettre au point.

Le module de compilation de la base de connaissances ne permet pas de compiler une base avec toutes les possibilites d'expression citees : pas de sous-listes dans les termes, sauf dans les termes prefixes par "predicat" ou "fonction".

Des tests ont ete faits entre les performances en version interpretee et compilee (resulats approximatifs):

avec - une base de 110 regles
- 2 a 3 variables classiques par regles
- une base de faits permettant 7 sept cycles soit environ 800 manipulations elementaires de regles (c'est a dire filtrage avec la BF pour tester l'activation)

version interpretee : 210 s CPU soit 3.5 minutes CPU
version compilee : 5 secondes CPU

Avec une base de faits plus complexe qui entraine des retours arriere dans le processus de filtrage naif utilise en version interpretee :

version interpretee : 400 s CPU soit 6.5 mn CPU
version compilee : 10 s CPU

On ne compte pas dans ces mesures les appels eventuels au garbage collector.

D'autres tests sont en cours avec des bases de regles plus complexes (fonction, predicat).

Il apparait donc que le travail en version interpretee reste raisonnable avec des bases de regles de 50 regles environ.

Mais ces tests sont peu significatifs du fait de l'evolution permanente du systeme et de l'objectif de ce dernier. Ce systeme est avant tout un outil de maquettage, qui en tant que tel offre un grand nombre d'outils parmi lesquels il faut choisir ceux necessaires pour realiser le systeme adequat.

Cet apport d'outils, parfois inutiles, coute en realite assez peu en performance (1/10 a 1/8 du temps seulement est gagne en supprimant les outils inutiles).

Pour optimiser rellement et efficacement il faudrait:

- enlever les fonctions inutiles
- revoir la technique de compilation de la base de connaissances. Faire en sorte que base compilee fournisse directement a chaque cycle les regles activables et meme parmi celles-ci, celles qui apportent quelque chose de nouveau dans la base (technique des arbres d'unification).
- optimiser les fonctions de choix qui peuvent a chaque cycle prendre plus d'un tiers de temps de cycle.

EXTENSIBILITE

CRIQUET peut aussi etre vu comme un ensemble de primitives LISP, tres faciles a combiner et donc a etendre.
Dans ce cas, pour quelqu'un connaissant Le_ Lisp, CRIQUET est un langage fonctionnel de programmation pour elaborer rapidement le systeme voulu .

Le code a ete commente et l'ensemble decoupe en petits modules pour faciliter une telle utilisation.

On peut par exemple modifier les primitives de manipulation des contextes, des agendas ou modifier les politiques d'utilisation des meta-regles.

Le logiciel permet aussi de coder facilement d'autres techniques d'evolution dans l'espace de recherches.

1) un parcours proche de "generate and test" :

Avec une base de faits initiale incomplete, on peut :

- soit selectionner un ensemble de regles partiellement activables en cheminement avant puis faire des chainages arriere sur les conclusions des regles ainsi retenues.

- soit generer en chainage avant une liste de buts possibles, puis avec une autre base de regles, diagnostiquer ces buts en chainage arriere.

2) un parcours du type "choix d'etat, choix d'action" :

Le principe est le suivant : a un instant "t", on detecte toutes les regles activables sur la base de faits courante. On genere toutes les bases accessibles a partir de la base courante.
On calcul ensuite une fonction d'evaluation pour chacune de ces nouvelles bases, ce qui leur affecte une note.
On choist comme nouvelle base courante celle qui a la meilleure note. Une fonction de test d'arret porte sur la base courante.

Algorithme :

```
initialisation : bf_cour = bf_init
                 pile_bf_util = bf_cour
                 pile_bf_act = bf_cour

tant que (and (not(succes bf_cour))(not(null pile_bf_act)))

        act = (detection-av bf_cour base_regles)

        pour chaque element reg_act de act faire
             pile_bf_act = (empiler (deduction reg_act bf_cour))

        fin-pour

        (evaluer_note pile_bf_act)
        (trier_sur_note pile_bf_act)
        bf_cour = (depiler_sommet pile_bf_act)
        pile_bf_util = (empiler bf_cour)

 fin-tant-que
```

En fin de parcours, pile_bf_util donne la liste des bases de faits utilisees.

3) De meme, on peut utiliser de facon plus poussee les buts intermediaires donnes dans la base de buts. Comme dans TANGO, le systeme peut a chaque cycle choisir parmi les regles activables celle qui permet la jonction entre le parcours suivi jusqu'alors, en chainage avant, et un cheminement arriere a partir de l'un des buts partiels. (ref biblio TANGO).

APPRENTISSAGE

Le logiciel permet aussi dans une moindre mesure l'apprentissage et l'auto-apprentissage du systeme. L'aspect fonctionnel du langage de base, la modularite, extensibilite du code permettent d'implementer des taches de ce genre.
Le code, meme s'il n'est pas sous une forme purement declarative, est decoupe en petits modules que le systeme lui meme peut manipuler et combiner.

La ponderation des regles peut etre dynamique et modifiee en cours de raisonnement, par exemple a l'aide de meta-regles (avec la fonction mco). Le systeme peut donc revoir et modifier les techniques de choix qui sont basees sur la ponderation des regles.

L'apprentissage est aussi possible par memorisation d'un parcours ayant aboutit a un succes. La fonction "appr" permet de creer une nouvelle regle a partir de l'historique rendu apres un cheminement (fonctions internes : chunk, aj_chunk).

Dans une premiere version, cette technique ne permet de manipuler que des termes simples dans les regles (quadruplets ordinaires, ajoutfaits en partie action et appels de fonctions).

Le systeme ignore les fonctions qui apparaissent en partie premisses des regles utilisees, puisque par definition leur appel a ete satisfaisant. Par contre, les appels de fonctions apparaissant en partie action des regles utilisees apparaissent en partie action de la regle generee pour conserver la propriete de systeme ouvert au logiciel (actions possibles sur d'autres environnements).

Le systeme, s'il le peut, introduit ensuite des variables dans la regle generee de facon a generaliser le resultat.
Pour obtenir cette generalisation (introduction de variables), l'utilisateur doit appeler la commande "appr" avec le parametre "t". Sinon, la regle generee est sans variables.

Ce type d'apprentissage permet d'ameliorer les performances dans toutes les resolutions qui utilisent souvent la recherche d'un meme sous but.

Avec la base d'objets, des techniques de generation de concepts par abstraction et specialisation sont d'autres formes d'apprentissage a mettre en place.

liste des commandes disponibles

Commandes du modules d'interface :
prd : pour imprimer le contenu d un dictionnaire du module d'interface
ajmd : pour ajouter un mot dans un dictionnaire de l interface
spmd : pour supprimer un mot dans un dictionnaire
mdm : pour modifier le type d un nom ou verbe dans le dictionnaire associe
ajsd : pour ajouter un synonyme dans on dictionnaire
rchm : recherche d'un mot dans un dictionnaire

Commandes de manipulations des faits :
ajf : ajouter des faits en conversationnel dans la base de faits
spf : supprimer un fait dans la base de faits
mdf : modifier le coeff de plausibilite d un fait
cbf : construire la base de faits
spbf : detruire la base de faits
bf : imprimer la base de faits
tribf : pour trier la base de faits sur le coeff de vraisemblance
rechbf : recherche associative dans la base de faits

Commandes de manipulations des hypotheses :
bh : imprimer l ensemble hypotheses
ajh : ajouter une hypothese
sph : supprimer une hypothese
spbh : supprimer la base d'hypotheses

Commandes de manipulations des buts :
ajb : pour ajouter un but dans buts
spb : pour supprimer un but dans buts
cbb : pour creer l ensemble buts
spbb : pour detruire tout buts
bb : pour imprimer buts

Commandes de manipulations des regles :
pr : impression d une regle
prl : impression d une liste de regles
prcoh : impression des regles de coherence connues
prc : impression de regles selon une recherche sur sur le coefficient de vraisemblance
br : impression de la base de regles
mdr : pour modifier le texte d'une regle en edition de texte
mdrco : pour juste modifier le coeff de vraisemblance d'une regle
spr : suppression d une regle
ajr : ajout d une regle
rcp : pour generer la contraposee de regles
req : pour generer des regles equivalentes
spbr : supprimer toute la base de regles

coh : specifier une liste de faits incompatibles avec un fait
avecbrcoh : pour indiquer une base de regles de coherence

compilation : pour compiler la base de regles
initialisation : pour initialiser la base compilee avec la base
de faits

testbr : pour verifier la coherence de la base de regles

Commandes de manipulations des agendas :
sauvag : sauvegarder un agenda
chargag : charger un agenda
prlag : imprimer la liste des noms d'agendas utilises
ajag : ajouter un element dans l'agenda courant
spag : supprimer un element dans l'agenda courant
vidag : vider un agenda memorise ou le courant
extag : extraire un element de l'agenda courant
actag : activer un element de l'agenda courant, tout l'agenda
prag : impression de l'agenda courant

Commandes de manipulations des contextes :
crcont : creer un contexte
spcont : supprimer un contexte
prlcont : imprimer la liste des contextes crees
chcont : charger un contexte

Commandes de manipulations des fonctions :
ajfct : definir une nouvelle fonction interne ou en modifier
une existante
spfct : supprimer une fonction interne
prlfct : liste des fonctions internes creees

defext : definir une nouvelle fonction externe ou en modifier
une existante
spfext : supprimer une fonction externe
prlfext : liste des fonctions externes creees

Commandes de recherches parmi les regles :
rhp : recherches de regles sur un ou plusieurs mots dans
les premisses
rha : recherches de regles sur un ou plusieurs mots dans
les actions
rhfa : recherche d une forme dans les parties actions
rhfp : recherche d une forme dans les premisses
rht : recherche des regles selon leur type
rhc : recherche des regles de coherence
rhi : recherche des regles d inference
rst : restriction de la base de regles utilisees

Commandes de manipulations des meta-regles :
pmr : impression d une meta-regle
bmr : impression de la base de meta-regles
ajmr : ajout d une meta-regle
spmr : suppression d une meta-regle
spbmr : suppression de toute la base de regles
mdmr : pour modifier une meta-regle
metam : pour activer 1 cycle du meta-moteur

Commandes de manipulations des indicateurs :
avecmeta : pour travailler en utilisant les meta-regles
sansmeta : pour ne plus utiliser les meta-regles
avecbutord : pour tenir compte de l ordre des buts partiels
sansbutord : pour ne pas tenir compte de leur ordre
rchex : pour utiliser une recherche exhaustive
nrchex : pour ne plus utiliser de recherche exhaustive
avecglog : pour assurer la coherence logique lors des deductions
avecgsem : pour assurer la coherence semantique lors des deductions
sansglog : sans la gestion de coherence logique
sansgsem : sans la gestion de coherence semantique
avectrace : travailler avec traces
sanstrace : travailler sans traces - mode adopte par defaut
avecomment : pour utiliser les commentaires des regles
sanscomment : pour ne plus utiliser les commentaires des regles
avecbrdyn : pour permettre une base de regles dynamique
sansbrdyn : pour travailler avec une base de regles statique
avecint : pour utiliser le module d'interface
sansint : pour ne plus utiliser le module d'interface
avecnon : pour choisir un type de negation

Commandes pour activer un cheminement :
chav : activer le raisonnement en chainage avant
charr : activer le raisonnement en chainage arriere
charr-p : activer le raisonnement en chainage arriere partiel
chm : activer le raisonnement en chainage mixte

Commande d'apprentissage :
appr : pour generer une nouvelle regle a partir de l'historique
du dernier cheminement

Commande pour choisir une structure de controle :
chx : pour choisir la strategie de choix

Commande pour avoir les explications du systeme :
expl : pour avoir les explications sur le raisonnement

Commandes de sauvegardes et chargements :
sauvbrc : sauvegarde automatique sans parametre de la BR courante
sauvbfc : sauvegarde automatique sans parametre de la BF courante
resbr : restaurer l ancienne base de regles
resbf : recupere la derniere base de faits

sauvbf : sauvegarder la base de faits courante

sauvbr : sauvegarder la base de regles courante
sauvbmr : sauvegarder la base de meta-regles courante
sauvbb : sauvegarder la base de buts courante
sauvbh : sauvegarder la base d'hypotheses courante

brs : pour lister le contenu d un fichier de regles
bfs : pour lister un fichier de faits

chargbf : charger une nouvelle base de faits
chargbr : charger une nouvelle base de regles
chargbmr : charger une nouvelle base de meta-regles
chargbb : charger une nouvelle base de buts
chargbh : charger une nouvelle base d'hypotheses

pnbf : lister les noms utilises pour sauvegarder des BF
pnbr : lister les noms utilises pour sauvegarder des BR
pnbmr : lister les noms utilises pour sauvegarder des BMR
pnbh : lister les noms utilises pour sauvegarder des bases
d'hypotheses
pnbb : lister les noms utilises pour sauvegarder des bases
de buts

Commandes diverses :
spnom : supprimer un nom dans l'une des listes citees
ajnom : ajouter un nom dans une des listes citees

sauver : avec 1 param, sauvegarde dans un fichier Multics
charger : avec 1 param, charger un fichier Multics
edit : avec 1 param, acces en edition pleine page sur un
fichier Multics

ruf : pour faire la reunion d un fichier de faits avec la
base de faits courante
rur : pour faire la reunion d un fichier de regles avec la
base de regles courante , utilisable aussi avec des meta-regles

info : pour avoir des informations sur le systeme
help : pour avoir des informations sur une commande en particulier
sauve : pour garder une trace de toute la session
fin : pour sortir du systeme en sauvegardant l environnement
avecsauve : meme chose que sauve, mais independant du systeme hote
finsauve : meme chose que fin, mais independant du systeme hote
end : pour sortir du systeme sans rien sauvegarder

Les commandes de l'editeur pleine page :

-A		va au début de la ligne
-B	<-	recule d'un caractère
-C		sort de PEPE: retour par -E
-D		détruit le caractère courant
-E		va en fin de ligne
-F	->	avance d'un caractère
-G		annule la commande en cours
-K		détruit la ligne pointée par le curseur
-L		re-affiche tout l'écran
-M	RC	brise la ligne à la position du curseur
-N	v	passe à la ligne suivante
-O		brise la ligne au niveau du curseur
-P	^	passe à la ligne précédente
-S		recherche une chaîne
-V		passe à l'écran suivant
-Y		force à la position du curseur la dernière ligne détruite
DEL		détruit le caractère à gauche du curseur
ESC E		exécute le tampon courant
ESC F		change le nom du fichier courant
ESC I		insère un autre fichier
ESC R		lecture dans le tampon d'un nouveau fichier
ESC S		sauve le tampon courant dans le fichier courant
ESC V		passe à l'écran précédent
ESC W		écriture du tampon courant dans le fichier
ESC X		appel direct d'une fonction de PEPE
ESC Z		sauve sur disque le tampon et charge ce tampon en mémoire
ESC)		se positionne sur la) fermante suivante (à la LISP)
ESC]		se positionne sur le] fermant suivant (à la LISP)
ESC >		va en fin du tampon
ESC <		va au début du tampon
ESC ?		affiche un aide mémoire de PEPE

Une version reduite du logiciel est aussi disponible.

La taille de cette version est donnee ci-dessous:

19 modules source avec la compilation de la base de connaissances

230 K au total
environ 6000 lignes

version interpretee du code

28000 cons
723 symbol
1300 string
8 heap

version compilee de code

non encore obtenue
configuration trop restreinte
pour compiler.

Les fonctionnalites perdues sont :

-le module d'interface

-l'aide en ligne (help)

-gestion de coherence des faits et des regles

-gestion des contextes

-gestion des agendas

-l'apprentissage

On conserve les possibilites de compiler la base de connaissances. Mais on peut aussi ne pas charger les 2 modules concernes (compilation.ll et initialisation.ll), ce qui fait gagner 30K.

La liste des fonctionnalites disponibles peut etre adaptee selon la place et les besoins des utilisateurs.

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique