Rapport de projet : TP n°3

Johan Medioni – Marie Julien

TP n°3 : Conception d'un Système Expert. Sujet choisi : identification des planètes au sein du système Solaire.

Introduction

Le but du système expert proposé est d'identifier, à partir de données initialement proposées par l'utilisateur, quelle planète pourrait correspondre. En cas d'insuffisance des données, le système devra être capable de demander à l'utilisateur d'en saisir davantage.

Ce système expert peut trouver application si on veut identifier une planète du système solaire après avoir effectuer des calculs pour déterminer la période de rotation de cette planète, ou après avoir effectué des analyses spectrométriques aidant à déterminer la composition de son atmosphère.

On peut envisager d'étendre ce système expert aux satellites des planètes du système solaire.
Néanmoins, on ne pourra pas utiliser pour les caractériser les données relatives au lieu avec le Soleil (période de révolution, distance au soleil). A noter également que les satellites n'ont généralement pas d'atmosphère.
L'expertise est alors bien plus délicate. Il faudrait dans un premier temps déterminer à quelle planète est associé le satellite, afin de réduire le champ de recherche. Le cas des géantes gazeuses (notamment Jupiter et Saturne, qui ont chacune plus de 60 satellites) reste cependant épineux.
Pour cette raison, on n'effectuera pas cette extension.

Table des matières

I. Base de règles

1.1. Idée générale

L'idée générale des règles de ce système expert est d'éliminer des choix possibles en fonction des paramètres entrés par l'utilisateur. Ainsi chaque règle a-t-elle pour conclusion un couple (ensemble T/NIL).

Les ensembles sont des groupes de planètes ayant une ou plusieurs caractéristiques en commun. On détaillera plus loin ces critères.

La valeur T est affectée à un ensemble si la donnée examinée permet à la planète inconnue d'appartenir à cet ensemble. La valeur NIL l'est dans le cas contraire. Néanmoins, aucune règle n'a pour conclusion (ensemble NIL). On considère que l'absence de (ensemble T) revient à (ensemble NIL).

1.2. Paramètres retenus

Il a fallu choisir certains critères pour différencier les planètes. Voici une liste de ceux qui ont été retenus, car ils étaient les plus intéressants pour la conception d'un système expert : suffisamment déterminants, mais pas trop non plus, afin de pouvoir avoir une certaine profondeur dans la recherche de but. On peut évidemment envisager de compléter la base de règles en exploitant de plus en plus de critères.
Tous les critères retenus sont des données numériques. Elles ne sont jamais exploitées de manière exacte. On teste toujours si une valeur entrée pas l'utilisateur est inférieure ou supérieure à une valeur seuil (prémisse des règles).

- Densité : densite dans la base de règles. Il s'agit de la densité moyenne de la planète.
- Distance au soleil : distance-soleil<km> dans la base de règles. Elle est prise centre à centre, néanmoins elle supporte toutes les approximations puisque les données ne sont pas exploitées de manière exacte.
- Période de rotation : periode-de-rotation<jours> dans la base de règles. Il s'agit du temps que met une planète pour faire un tour autour d'elle-même.
- Rayon : rayon<km> dans la base de règles. Il s'agit du rayon moyen d'une planète, pris de son centre à son équateur.
- Complexité du système d'anneaux : complexité-des-anneaux<1=simple,2=complexe>. 1 pour simple, 2 pour complexe.
- Proportion du gaz dans l'atmosphère de la planète : nom-du-gaz<%> dans la base de règles. Différents seuils ont été établis :
    - =0 : molécule absente
    - <0,01% : traces
    - <2% : très faible
    - <10% : faible
    - <30% : moyenne
    - <80% : forte
    - ≥80% : très forte

1.3. Ensembles

Les ensembles retenus sont les suivants :

- Gazeuse
  - Appartient aux planètes gazeuses
  - Propriété sur la densité : de l'ordre de 1
  - Propriété sur la distance au Soleil : au delà de 448'794'000 km
  - Période de rotation assez faible (comparée à la Terre)
- Tellurique
  - Appartient aux planètes telluriques
  - Propriété sur la densité : entre 3,5 et 5,5.
  - Propriété sur la distance au Soleil : inférieure à 448'794'000 km
- TMaUN
  - Terre, Mars, Uranus ou Neptune
  - Période de rotation d'environ 0,85 jour.
- TMa
  - Terre ou Mars
  - Présence d'Argon très faible dans l'atmosphère
  - Présence d'Eau très faible dans l'atmosphère
  - Présence de Dihydrogène en traces dans l'atmosphère
  - Présence de Méthane très faible dans l'atmosphère
- UN
  - Uranus ou Neptune
  - Rayon de l'ordre de 25'000 km
  - Présence de Carbone moyenne dans l'atmosphère
  - Présence de Méthane faible dans l'atmosphère
- MeMa
  - Mercure ou Mars
  - Rayon de l'ordre de 2'750 km
- JS
  - Jupiter ou Saturne
  - Présence de Méthane très faible dans l'atmosphère
  - Rayon de l'ordre de 70'000 km
- TMaJ
  - Terre, Mars ou Jupiter
  - Présence d'Eau faible dans l'atmosphère
- VMa
  - Venus ou Mars
  - Présence de Dioxyde de Carbone très forte dans l'atmosphère
  - Présence de Diazote faible dans l'atmosphère
- VT
  - Venus ou Terre
  - Présence d'Hélium en trace dans l'atmosphère
  - Rayon de l'ordre de 6'000 km
- JUN
  - Jupiter, Uranus ou Neptune
  - Présence d'Hélium moyenne dans l'atmosphère

- SU
  - Saturne ou Uranus
  - Systèmes d'anneaux complexe
- JN
  - Jupiter ou Neptune
  - Système d'anneaux simple

1.4. Exploitation des sources

La source utilisée a été Wikipedia. Plusieurs articles ont été nécessaires.

Tout d'abord, ceux relatifs aux planètes gazeuses et telluriques, afin de déterminer les points communs majeurs entre les planètes de ces groupes.

Ensuite, les articles relatifs aux planètes ont permis d'établir un tableau de données.

Planet_name	Orbital_period	Synodic_period	Rotation_period	Mean_Radius
Mercure	88	116	58,7	2440
Venus	225	584	243	6052
Terre	365	/	1	6378
Mars	687	780	1	3376
Jupiter	4335	398	0,4	71492
Saturne	10757	378	0,4	60268
Uranus	30799	369	0,7	25559
Neptune	60190	367	0,7	24764

K	Na	O	Ar	H20	H2	O2	N	CO2	N2	SO2	CO	He	CH4
31,7	24,9	9,5	7,5	3,4	3,2	5,6	5,2	3,6	0	0	0	0	0
0	0	0	traces	traces	0	0	0	96,5	3,5	traces	traces	traces	0
0	0	0	0,934	0,4	traces	20	0	traces	78	0	traces	traces	traces
0	0	0	1,6	0,03	traces	0,13	0	95	3	0	0,07	0	traces
0	0	0	0	0,1	86	0	0	0	0	0	0	13	0,1
0	0	0	0	0	96	0	0	0	0	0	0	3	0,4
0	0	0	0	0	83	0	0	0	0	0	0	15	2,3
0	0	0	0	0	80	0	0	0	0	0	0	19	1,5

De ce tableau de données est né un autre tableau avec des approximations sur les données précises obtenues dans les articles de Wikipedia.

Planet_name	Orbital_period	Synodic_period	Rotation_period	Mean_Radius
Mercure	1,00E+02	1,00E+02	6,00E+01	2,00E+03
Venus	2,00E+02	6,00E+02	2,00E+02	6,00E+03
Terre	4,00E+02	/	1	6,00E+03
Mars	7,00E+02	8,00E+02	1	3,00E+03
Jupiter	4,00E+03	4,00E+02	0,4	7,00E+04
Saturne	1,00E+05	4,00E+02	0,4	7,00E+04
Uranus	3,00E+05	4,00E+02	0,7	2,50E+04
Neptune	6,00E+05	4,00E+02	0,7	2,50E+04

K	Na	O	Ar	H20	H2	O2	N	CO2	N2	SO2	CO	He	CH4
++++	++++	++	++	++	++	++	++	++	0	0	0	0	0
0	0	0	-	-	0	0	0	+++++	++	-	-	-	0
0	0	0	+		-	+++	0	-	+++++	0	-	-	-
0	0	0	+	+	-	+	0	+++++	++	0	+	0	-
0	0	0	0	+	+++++	0	0	0	0	0	0	+++	+
0	0	0	0	0	+++++	0	0	0	0	0	0	++	+
0	0	0	0	0	+++++	0	0	0	0	0	0	+++	++
0	0	0	0	0	+++++	0	0	0	0	0	0	+++	+

L'établissement de ces approximations permet d'avoir une meilleure idée des ressemblances entre les différentes planètes. C'est ce qui permet d'établir les ensembles décrits précédemment, et donc les règles.

1.5. Etablissement des règles

Nous allons vous présenter les règles par donnée exploitée. Le choix de représentation fait est le suivant :

(Ri ((prémisse 1)(prémisse 2) … (prémisse n)) ((conclusion 1)(conclusion 2) … (conclusion n)))

Choix classique qui permet une bonne exploitation des règles par le moteur d'inférence. De plus on peut leur lisibilité est bonne, et leur correction en est rendue plus aisée.

Etablissement d'une fonction presque :

```
(defun presque (donnee valeur)
                (and
                        (< donnee (* 1.30 valeur))
                        (> donnee (* 0.70 valeur))
                )
 )
```

Cette fonction permet d'exploiter les données entrées par l'utilisateur. Elle remplace l'égalité, dans la mesure où il est difficile d'obtenir des valeurs vraiment exactement semblables à celles qu'on recherche. Elle permet également de réduire le nombre de règles à établir en rapprochant les paramètres de plusieurs planètes.

1.5.1. Distance au Soleil

Ces règles permettent d'établir si une planète est tellurique ou gazeuse. Elles découlent directement des propriétés de ces ensembles (Cf. 1.3.).

(R1 ((<= distance-soleil<km> 448794000)) ((tellurique T)))
(R2 ((> distance-soleil<km> 448794000)) ((gazeuse T)))

1.5.2. Densité

Même commentaire que pour la distance au soleil.

(R3 ((>= densite 3.5) (<= densite 5.5)) ((tellurique T)))
(R4 ((<= densite 1.2) (>= densite 0.8)) ((gazeuse T)))

1.5.3. *Période de rotation*

L'exploitation de la période de rotation permet d'identifier nettement Venus et Mercure grâce à leur très grande période. On peut aussi établir l'appartenance de la planète inconnue à un des deux ensembles TMaUN et Gazeuse.

(R5 ((presque période-de-rotation<jours> 0.85)) ((TMaUN T)))
(R8 ((presque période-de-rotation<jours> 0.55)) ((gazeuse T)))
(R9 ((presque période-de-rotation<jours> 200)) ((Venus T)))
(R10 ((presque période-de-rotation<jours> 60)) ((Mercure T)))

1.5..4 Rayon

L'exploitation du rayon permet de déterminer l'appartenance de la planète inconnue à un des quatre groupes suivants : VT, MeMa, UN, JS.

(R11 ((presque rayon<km> 6000)) ((VT T)))
(R14 ((presque rayon<km> 2750)) ((MeMa T)))
(R17 ((presque rayon<km> 25000)) ((UN T)))
(R19 ((presque rayon<km> 70000)) ((JS T)))

1.5.5. Molécules dans l'atmosphère

Les règles résultant de l'exploitation de la proportion d'une molécule dans l'atmosphère de la planète permettent de définir l'appartenance de celle-ci à un groupe ou à l'identifier parfaitement. Mercure notamment, par sa composition atmosphérique très particulière, est très facilement identifiable de la sorte ; une dizaine de règles de cette catégorie ont pour conclusion (Mercure T).

(R22 ((> K<%> 30) (<= K<%> 80)) ((Mercure T)))
(R23 ((> Na<%> 30) (<= Na<%> 80)) ((Mercure T)))
(R24 ((> O<%> 2) (<= O<%> 10)) ((Mercure T)))
(R25 ((> Ar<%> 0.01) (<= Ar<%> 2)) ((TMa T)))
(R26 ((> Ar<%> 0) (<= Ar<%> 0.01)) ((Venus T)))
(R27 ((> Ar<%> 2) (<= Ar<%> 10)) ((Mercure T)))
(R28 ((> H2O<%> 2) (<= H2O<%> 10)) ((Mercure T)))
(R29 ((> H20<%> 0.01) (<= H20<%> 2)) ((TMaJ T)))
(R34 ((> H2<%> 2) (<= H2<%> 10)) ((Mercure T)))
(R35 ((eq H2<%> 0)) ((Venus T)))
(R36 ((> H2<%> 0) (<= H2<%> 0.01)) ((TMa T)))
(R37 ((> H2<%> 80)) ((gazeuse T)))
(R38 ((> O2<%> 10) (<= O2<%> 30)) ((Terre T)))
(R39 ((> O2<%> 2) (<= O2<%> 10)) ((Mercure T)))
(R40 ((> O2<%> 0.01) (<= O2<%> 2)) ((Mars T)))
(R41 ((> N<%> 2) (<= N<%> 10)) ((Mercure T)))
(R42 ((> CO2<%> 2) (<= CO2<%> 10)) ((Mercure T)))
(R43 ((> CO2<%> 80)) ((VMa T)))
(R49 ((> CO2<%> 0) (<= CO2<%> 0.01)) ((Terre T)))
(R50 ((> N2<%> 2) (<= N2<%> 10)) ((VMa T)))
(R51 ((> He<%> 0) (<= He<%> 0.01)) ((VT T)))
(R52 ((> He<%> 2) (<= He<%> 10)) ((Saturne T)))
(R53 ((> He<%> 10) (<= He<%> 30)) ((JUN T)))
(R56 ((> CH4<%> 0) (<= CH4<%> 0.01)) ((TMa T)))
(R57 ((> CH4<%> 0.01) (<= CH4<%> 2)) ((JS T)))
(R58 ((> CH4<%> 2) (<= CH4<%> 10)) ((UN T)))

1.5.6. Complexité des anneaux

Ces règles permettent de déterminer si la planète appartient à JN ou SU. A noter que la complexité des anneaux est le seul moyen d'identifier Uranus et Neptune. En effet ces planètes sont quasiment identiques, mis à part la complexité de leurs anneaux.

(R59 ((eq ring-comp 2)) ((SU T)))
(R60 ((eq ring-comp 1)) ((JN T)))

1.5.7. Règles d'intersection

Ces règles permettent de croiser tous les ensembles auxquels appartient la planète inconnue.

(R6((eq TMaUN T) (eq tellurique T))((TMa T))
(R7 ((eq TMaUN T) (eq gazeuse T)) ((UN T)))
(R12 ((eq VT T) (eq TMaUN T)) ((Terre T)))
(R13 ((eq VT T) (eq TMa T)) ((Terre T)))
(R15 ((eq MeMa T) (eq TMaUN T)) ((Mars T)))
(R16 ((eq MeMa T) (eq TMa T)) ((Mars T)))
(R30 ((eq TMaJ T)(eq gazeuse T)) ((Jupiter T)))
(R31 ((eq TMaJ T)(eq TMaUN T)) ((TMa T)))
(R32 ((eq TMaJ T)(eq JS T)) ((Jupiter T)))
(R33 ((eq TMaJ T)(eq tellurique T)) ((TMa T)))
(R44 ((eq VMa T)(eq TMaUN T)) ((Mars T)))
(R45 ((eq VMa T)(eq TMa T)) ((Mars T)))
(R46 ((eq VMa T)(eq VT T)) ((Venus T)))
(R47 ((eq VMa T)(eq MeMa T)) ((Mars T)))
(R48 ((eq VMa T)(eq TMaJ)) ((Mars T)))
(R54 ((eq JUN T)(eq TMaJ)) ((Jupiter T)))
(R55 ((eq JUN T)(eq JS T)) ((Jupiter T)))
(R61 ((eq SU T)(eq UN T)) ((Uranus T)))
(R62 ((eq SU T)(eq JUN T)) ((Uranus T)))
(R63 ((eq SU T)(eq JS T)) ((Saturne T)))
(R64 ((eq JN T)(eq UN T)) ((Neptune T)))
(R65 ((eq JN T)(eq TMaJ)) ((Jupiter T)))
(R66 ((eq JN T)(eq JS T)) ((Jupiter T)))

II. Moteur d'inférence

Le rôle du moteur d'inférence est d'utiliser la base de règles pour déduire de nouveaux faits, et ce jusqu'à ce qu'il ait atteint son but, ou qu'il soit bloqué. Ici, le but est de trouver une planète du système solaire à partir des données entrées par l'utilisateur. Le moteur d'inférence est bloqué lorsqu'il ne peut plus appliquer de règle, et qu'il n'a plus de donnée à demander à l'utilisateur.

En dehors de la fonction principale de recherche, nous avons utilisé plusieurs fonctions de service que nous allons présenter plus loin.

2.1. Variables globales et fonctions de base

2.1.1. Variables globales

Nous utilisons les variables suivantes :

- bf : la base de faits, vide au début de la recherche, et complétée au fur et à mesure. Les faits seront de la forme (propriété valeur).

- br : la base de règles, définie dans la partie I.

- règles-appliquées : la liste des règles déjà appliquées, vide au début. Elle ne contiendra que l'identifiant des règles, par exemple R1, R5, etc.

- liste-planètes : la liste des planètes du système solaire, qui nous servira à vérifier que notre but est atteint ou non.

- propriétés-a-ne-pas-demander : une liste de propriétés que le système ne devra pas demander à l'utilisateur. En effet, elles doivent être déduites des autres faits : par exemple le fait JUN (Jupiter ou Uranus ou Neptune).

2.1.2. Fonctions de bases

Nous avons écrit des fonctions très basiques, mais qui nous permettent d'alléger notre code et d'améliorer sa lisibilité.

- Prémisses(règle) :

```
(defun prémisses (règle)
 (cadr règle)
)
```

Elle renvoie la liste des prémisses d'une règle

- Conclusion(règle) :

```
(defun conclusion (règle)
 (caddr règle)
)
```

Elle renvoie la liste des conclusions d'une règle.

- Appartient-BF(fait) :

```
(defun appartient-BF (fait)
  (assoc fait *bf*)
 )
```

Renvoie NIL si le fait n'est pas dans la base de faits.

2.2. Fonctions de service

Elles sont essentielles au fonctionnement du moteur d'inférence.

- règles-applicables : renvoie la liste des règles pouvant être appliquées par rapport aux faits contenus dans la base de faits

- choix-règle : renvoie une règle choisie dans la liste des règles applicables

- appliquer-règle : applique la règle passée en paramètre

- mettre-a-jour-r : met à jour la liste des règles appliquées afin que certaines règles ne soient plus applicables, en fonction des faits

- but-atteint : renvoie soit NIL, si le but n'a pas été atteint, soit le nom de la planète trouvée

- ask-user : demande à l'utilisateur une donnée manquante et la met dans la base de faits. Si elle renvoie NIL, cela veut dire qu'aucune donnée ne peut être demandée

-

2.2.1. Règles-applicables

Algorithme :

```
FONCTION REGLES-APPLICABLES (BF)
liste ← NIL
        Pour chaque R de Base-règles
                prém ← prémisses ( R )
                Ok ← T
                Pour chaque Pré de prém
                        valeur ← cadr (assoc (cadr pré) BF))
                        Sauf si ((fait ≠ NIL)
                                et (valeur ≠ ne-sait-pas)
                                et (funcall (car pré) valeur (cadr pré))) alors
                                Ok ← NIL
                        Fin si
                Fin pour
                Si ((Ok = T) et (R n'appartient pas à règles-appliquées)) alors
                        liste ← R
                Fin si
        Fin pour
Retourner liste
```

On parcourt la liste des règles, et pour chacune d'entre elle, on parcourt les prémisses, et on vérifie :

- que le fait existe dans la base de faits
- que sa valeur ne vaut pas ne-sait-pas (l'utilisateur a dit qu'il ne connaissait pas cette valeur)
- que la valeur correspond à la prémisse de la règle

Si OK est vrai à la fin du parcourt, cela signifie que toutes les prémisses sont valides, et donc qu'on peut appliquer la règle. On vérifie qu'elle n'a pas déjà été appliquée, et si ce n'est pas le cas, on la met dans la liste à renvoyer.

En lisp, cela donne :

(defun règles-applicables (BF) (let ((liste NIL)) (dolist (R *br*) (let ((prém (prémisses R)) (OK T)) (setq OK T) (dolist (pré prém) (let ((valeur (cadr (assoc (cadr pré) *BF*)))) (unless (and	On créée une variable locale avec let. On parcourt la base de règles. On utilise une variable locale OK qui vaut T au début. On parcourt les prémisses de la règle. On vérifie que valeur n'est nulle, qu'elle

(not (null valeur)) (not (eq 'ne-sait-pas valeur)) (funcall (car pré) valeur (caddr pré)))	n'est pas égale à 'ne-sait-pas, et qu'elle correspond à la prémisse.
(setq OK NIL)))) ;; fin dolist	Si une des conditions n'est pas vérifiée, OK vaut NIL. A la fin du dolist, si OK vaut NIL, alors au moins une prémisse n'est pas vérifiée.
(if (and OK (not (member (car R) *règles-appliquées*))) (setq liste (append liste (list R)))))) liste))	Si OK = T et R n'est pas déjà appliquée, alors on la met dans la liste à renvoyer. Enfin, on renvoie la liste.

2.2.2. *Appliquer-règle*

Algorithme :

```
FONCTION APPLIQUER-REGLE (règle)
Pour tout fait de (conclusion règle) faire
        Si (fait n'appartient pas à BF) alors
                BF ← fait
        Fin si
Fin pour
```

Cette fonction se contente de parcourir les conclusions de la règle à appliquer, qui sont des faits, et de les mettre dans la base de faits (s'ils n'y sont pas déjà).

Soit en Lisp :

(defun appliquer-règle (règle) (dolist (fait (conclusion règle)) (if (null (appartient-BF fait)) (push fait *bf*))) *bf*)	On parcourt les conclusions de la règle à appliquer. S'ils n'appartiennent pas déjà à la base de faits, on les met dans celle-ci. Enfin, on renvoie la base de faits.

2.2.3. But-atteint

Algorithme :

```
FONCTION BUT-ATTEINT (BF)
Pour chaque planète de liste-planètes faire
        Si (cadr(assoc planète BF) = T) alors
                Retourner planète
        Fin si
Fin pour
```

C'est pour cette fonction que nous avons défini une variable globale liste-planètes. Elle nous sert à chercher si l'on a un fait de la forme (planète T) dans la base de faits, en regardant planète par planète.
Si on trouve une planète, on sort de la boucle et renvoie le nom de cette planète.
Si on ne trouve rien, le dolist renverra NIL.

Code lisp :

(defun but-atteint (BF) (dolist (planète *planètes*) (if (eq (cadr (assoc planète BF)) T) (return planète))))	On parcourt la liste des planètes. On regarde si on a dans la base de faits un fait de la forme (planète T). Si oui, on sort de la boucle et on retourne le nom de la planète.

2.2.4. Ask-user

Algorithme :

```
FONCTION ASK-USER (BF)
Pour chaque R de base-règles faire
        prémisses ← prémisses ( R )
        fin ← NIL
        Si ((conclusion de R n'appartient pas à BF) et (R n'est pas déjà appliquée)) alors
                fin ← Pour chaque pré de prémisses faire
                        fait ← assoc ((cadr pré) BF)
                        valeur ← NIL
                        Si ((fait = NIL) et (fait n'appartient pas aux propriétés à ne pas
demander)                       alors
                                Afficher « Entrer une valeur »
                                valeur ← lire la valeur entrée
                                BF ←  nouveau fait
                                Retourner T
                        Fin si
                        Fin pour
        Fin si
        Si (fin = T) alors
                Retourner T
        Fin si
Fin pour
```

Cette fonction cherche la première règle dont la conclusion n'est pas déjà dans les faits, et qui n'est pas appliquée. Elle parcourt ensuite ses prémisses, vérifie que les faits ne sont pas non plus dans la base de faits, et que ce ne sont pas des propriétés à ne pas demander.

Le code est le suivant :

(defun ask-user (BF)	On parcourt la base de règles.
(dolist (R *br*)	
(let ((prémisses (prémisses R))	On utilise une variable locale fin pour savoir ce que le dolist a renvoyé.
(fin NIL))	
(if (not (member (car R) *règles-appliquées*)))	Si la règle n'est pas déjà appliquée, alors :
(progn	
(setq fin (dolist (pré prémisses)	On parcourt toutes les prémisses de la règle.
(let ((fait (assoc (cadr pré) BF))	
(valeur NIL))	On utilise des variables locales fait pour accéder plus facilement aux champs de la prémisse, et valeur.
;; variables locales fait et valeur	
(if (and (null fait)	

(null (member (cadr pré) *propriétés-a-ne-pas-demander*))) (progn (format T "Entrez une valeur pour ~a (ou ne-sait-pas)" (cadr pré)) (setq valeur (read)) (setq *bf* (ajout (list (cadr pré) valeur) *bf*)) (return T)))))))) (if fin (return T)))))	Si le fait est nul et qu'il n'appartient pas aux propriétés à ne pas demander, alors : On demande à l'utilisateur de rentrer une donnée. On la met dans valeur, puis on met ce nouveau fait dans la base de faits. On sort de la boucle dès qu'on a ajouté un fait, et fin vaut T. Si fin vaut T, on doit aussi sortir du premier dolist, et on renvoie T. Si l'utilisateur n'a entré aucune nouvelle donnée, la fonction renvoie NIL.

2.2.5. Mettre-a-jour-br

Cette fonction consiste à mettre dans la liste des règles appliquées *règles-appliquées* les règles considérées comme inutiles à appliquer. Elle est utilisée après chaque application de règles. L'idée est de tester si certains faits sont dans la base de faits pour savoir quelles règles sont inutiles.

Typiquement, si le fait B est dans la base de faits, toute règle ayant pour conclusion B ne sera plus utile. Comme on utilise également une notion ensembliste dans les faits, si on trouve un fait A inclus dans un fait B dans la base de faits, alors toute règle menant à B ou A ne sera pas utile.

Le schéma du code lisp est le suivant :

```
(defun mettre-a-jour-br ()
 (cond
        ((equal (assoc 'tellurique *bf*) '(tellurique T))
                (dolist (x '(R1 R2 R3 R4 R7 R8 R16 R19 R30 R32 R37 R52 R53 R54 R55 R57 R58
R59 R60 R61 R62 R63 R64 R65 R66))
                  (pushnew x *règles-appliquées*))
        ))
        ((condition_2
                (dolist ' (x (liste de règles))
                  (pushnew x *règles-appliquées*))
        ))
...
 )
)
```

La liste de règles en argument du dolist est la liste de règles devenues obsolète en présence du fait correspondant dans le cond. Si ce fait est présent on pousse donc une à une les règles obsolètes dans la liste de règles appliquées afin de ne pas les appliquer plus tard.

2.2.6. Choix-règle

Cette fonction est très simple. En effet nous avons décidé de choisir simplement la première règle de la liste de règles applicables passée en paramètre, car les autres fonctions restreignent suffisamment les règles possibles.

Le code lisp se résume à (pop règles-applicables).

2.3. Fonction principale de recherche

C'est la fonction qui représente le fonctionnement du moteur d'inférence. Son algorithme est le suivant :

```
FONCTION RECHERCHE (BF)
Afficher « On cherche une planète à partir de ses caractéristiques »
Boucle
        Si (but-atteint est vrai) alors
                Afficher « Fin de la recherche, la planète cherchée est (but-atteint) »
                Sortir de la boucle
        Sinon si (règles-applicables = NIL) alors
                Afficher « Pas de règle à appliquer »
                Si (ask-user = NIL) alors
                        Sortir de la boucle
                Fin si
        Sinon
                règles-candidates ← règles-applicables
                règle-choisie ← choix-règle (règles-applicables)
                BF ← Appliquer-règle (règle-choisie)
                Règles-appliquées ← règle-choisie
                Mettre à jour
        Fin si
Fin boucle
```

On voit que le moteur d'inférence est bloqué s'il ne trouve plus de règles à appliquer ni de faits à demander à l'utilisateur. Dans ce cas, il affiche un message et sort de la boucle.

Soit en lisp :

(defun recherche (BF) (format T "On cherche une planète à partir de ses caractéristiques :~%") (loop (cond ((but-atteint BF) (format T "Fin de la recherche ! La planète cherchée est ~a." (but-atteint BF)) (return-from NIL))	On fait une boucle avec loop. Si le but est atteint, on affiche un message, on sort de la boucle.

((null (règles-applicables BF)) (format T "Pas de règles à appliquer ~%") (if (null (ask-user *bf*)) (progn (format T "Fin de la recherche, pas de résultat possible.~%") (return-from NIL))))	S'il n'y a pas de règles applicables, on regarde ce que renvoie ask-user. Soit c'est T et on recommence la boucle, soit c'est NIL et on est bloqués, donc on sort de la boucle.
(T (let ((règles-candidates NIL) (règle-choisie NIL)) (setq règles-candidates (règles-applicables BF)) (setq règle-choisie (choix-règle règles-candidates)) (format T "On applique la règle ~a ~%" (car règle-choisie)) (setq *bf* (appliquer-règle règle-choisie)) (format T "Conclusion : ~s ~%" (conclusion règle-choisie)) (setq *règles-appliquées* (append *règles-appliquées* (list (car règle-choisie)))) (mettre-a-jour-br))))) (format T "Pour faire une nouvelle recherche, réévaluer les trois dernières lignes du code."))	Sinon, on choisit une règle parmi les règles applicables, on l'applique, on l'ajoute à la liste de règles appliquées. Enfin, on met à jour cette liste de règles appliquées.

III. Exemples d'utilisation

Nous allons tester ici plusieurs recherches avec les faits suivants :

- distance-soleil = 58 000 000, période de rotation = 58

> (recherche *bf*)
On cherche une planète à partir de ses caractéristiques :
Pas de règles à appliquer
Entrez une valeur pour DISTANCE-SOLEIL<KM> (ou ne-sait-pas)**58000000**
On applique la règle R1
Conclusion : ((TELLURIQUE T))
Pas de règles à appliquer
Entrez une valeur pour PÉRIODE-DE-ROTATION<JOURS> (ou ne-sait-pas)**58**
On applique la règle R10
Conclusion : ((MERCURE T))
Fin de la recherche ! La planète cherchée est MERCURE.Pour faire une nouvelle recherche, réévaluer les trois dernières lignes du code.
NIL

Nous n'avons ici pas besoin d'entrer de données supplémentaires, car la restriction des règles fait qu'on trouve Mercure très rapidement.

- Avec des valeurs aberrantes telles que :

> (recherche *bf*)
On cherche une planète à partir de ses caractéristiques :
Pas de règles à appliquer
Entrez une valeur pour DISTANCE-SOLEIL<KM> (ou ne-sait-pas)**50000000000**
On applique la règle R2
Conclusion : ((GAZEUSE T))
Pas de règles à appliquer
Entrez une valeur pour PÉRIODE-DE-ROTATION<JOURS> (ou ne-sait-pas)**20**
Pas de règles à appliquer
Entrez une valeur pour RAYON<KM> (ou ne-sait-pas)**20**
Pas de règles à appliquer
Entrez une valeur pour H20<%> (ou ne-sait-pas)**0**
Pas de règles à appliquer
Entrez une valeur pour HE<%> (ou ne-sait-pas)**0**
Pas de règles à appliquer
Entrez une valeur pour CH4<%> (ou ne-sait-pas)**0**
Pas de règles à appliquer
Entrez une valeur pour COMPLEXITE-DES-ANNEAUX<1=SIMPLE,2=COMPLEXE> (ou ne-sait-pas)**1**

On applique la règle R60
Conclusion : ((JN T))
Pas de règles à appliquer
Fin de la recherche, pas de résultat possible.
Pour faire une nouvelle recherche, réévaluer les trois dernières lignes du code.

- Distance-soleil = 777909600, Période-de-rotation = 0,41, Rayon inconnu, %H20 = 0,1

> (recherche *bf*)
On cherche une planète à partir de ses caractéristiques :
Pas de règles à appliquer
Entrez une valeur pour DISTANCE-SOLEIL<KM> (ou ne-sait-pas)**777909600**
On applique la règle R2
Conclusion : ((GAZEUSE T))
Pas de règles à appliquer
Entrez une valeur pour PÉRIODE-DE-ROTATION<JOURS> (ou ne-sait-pas)**0.41**
Pas de règles à appliquer
Entrez une valeur pour RAYON<KM> (ou ne-sait-pas)**ne-sait-pas**
Pas de règles à appliquer
Entrez une valeur pour H20<%> (ou ne-sait-pas)**0.1**
On applique la règle R29
Conclusion : ((TMAJ T))
On applique la règle R30
Conclusion : ((JUPITER T))
Fin de la recherche ! La planète cherchée est JUPITER

Conclusion

Nous avons conçu un système expert capable d'identifier une planète parmi celles du systèmes solaires à partir de divers types de caractéristiques.

La stratégie utilisée dans la recherche d'une conclusion est celle du chaînage avant. En effet, on part des prémisses, et on recherche des conclusions diverses pour arriver à une conclusion globale.

Les règles sont donc adaptées à cette stratégie puisqu'elles permettent de croiser les faits déduits par le système seul (les ensembles).

La fonction de recherche principale l'est également : elle met à jour la liste de règles à appliquer au fur et à mesure des nouveaux faits entrés dans la base de faits afin de ne pas faire de parcours inutile. D'autre part elle comporte une condition d'arrêt qui permet de savoir quand est-ce qu'on a fini, c'est-à-dire quand on a trouvé une planète.

Ce projet aura été l'occasion de mettre en application toutes les connaissances en programmation lisp acquises au cours du semestre, ainsi que les notions de recherches et de système expert. Il a également mis l'accent sur la conception d'une base de règles en utilisant des recherches effectuées par nous-mêmes.

On peut envisager maintes améliorations à ce systèmes experts, comme indiqué dans l'introduction et dans le 1.2. Néanmoins on peut également imagine une variante plus calculatoire : à partir de quelques données numériques, le système expert serait capable de calculer d'autres valeurs, comme s'il résolvait un problème de physique. Ainsi en rentrant la masse et la distance au soleil de l'objet inconnu, ce système expert pourrait calculer sa période et conclure si l'objet est Mercure ou Venus (Cf. 1.5.3).