Rapport de IA01 :
Intelligence artificielle – Représentation des connaissances

UNIVERSITE DE TECHNOLOGIE DE COMPIEGNE

Automne 2016

Guillaume JOUNEL & Julien JERPHANION

Sujet du rapport :
TP03 : Réalisation d'un système expert d'ordre 0+

Département des étudiants :
Génie Informatique

Professeur :
Marie-Hélène ABEL

Table des matières

Liste des programmes

1 Introduction : Présentation du système expert

1.1 Problématique

Tous les programmeurs sont un jour confrontés au problème suivant :

> « *Quels de programmation et technologies sont les plus adaptés pour le projet que je souhaite développer dans mon cadre d'utilisation ?* »

Pour pallier à ce problème, nous allons concevoir un système expert qui propose différentes possibilités les plus adaptées selon l'usage.

Pour cela, nous prendrons en compte de multiples critères tels que le domaine d'application (calcul numérique, intelligence artificielle...), l'expérience de l'utilisateur ou encore les caractéristiques de sa machine (Linux, MacOS...).

1.2 Sources de connaissances sur le sujet

Les sources d'expertise ne manquent pas : il existe de nombreux sites et ressources sur le Net qui donnent les avantages et inconvénients de tous les langages de programmation existants selon les cas d'utilisation. En voici quelques uns :

- Wikipédia : Liste des langages de programmations par type : `https://en.wikipedia.org/wiki/List_of_programming_languages_by_type` ;
- Learneroo : The Different Programming Languages : `https://www.learneroo.com/modules/12/nodes/94` ;
- WhoIsHostingThis : What Code Should You Learn ? `http://www.whoishostingthis.com/blog/2014/09/04/learn-to-code/`.

2 Architecture

2.1 Rappels sur l'architecture

Rappelons rapidement l'architecture d'un système expert. Un système expert est constitué de trois parties principales dissociées les unes des autres : une *base de faits*, une *base de règles*, et un *moteur d'inférences*.

La base de faits est une base d'informations qui comprend les faits initiaux et déduits au cours du programme.

La base de règles contient les différentes règles (connaissances implicites de l'expert rendues explicites pour être représentées informatiquement) utilisées pour déduire d'autres faits.

Les inférences au cours du processus sont réalisées par le moteur d'inférences. C'est lui qui fait le lien entre les deux précédentes bases. Il exécute les règles contenues dans la base de règles au regard des faits présents dans la base de faits ; les règles étant déclenchables en fonction des faits avérés. À la fin de l'exécution d'une règle, le résultat retourné qui est aussi un fait est stocké dans la base de faits.

Il existe de type de fonctionnement pour les moteurs d'inférences : le *chaînage avant* et le *chaînage arrière*.

Le chaînage avant consiste à regarder les faits présents et à choisir une règle qui peut être exécutée : on cherche les résultats que l'on peut obtenir en se basant sur les résultats déjà obtenus.

Le chaînage arrière examine les règles à exécuter pour arriver à un certain fait : on cherche un moyen d'arriver à un certain résultat.

Il s'agit ici de construire un système expert d'ordre 0+ – que nous avons choisi d'appeler *Cactus* –, c'est à dire un système expert manipulant des faits qui ne sont non pas des propositions booléennes mais des *triplets* comportant trois parties :

1. un attribut, qui est le nom du concept que l'on veut modéliser dans le fait ;
2. une valeur, qui permet de quantifier l'attribut ;
3. un opérateur, qui permet de préciser la valeur de l'attribut

Ainsi `(temperature >= 30)` et `(saison EQ été)` sont des faits vus sous l'angle d'un système-expert d'ordre 0+ dont les attributs, les opérateurs et les valeurs sont respectivement `temperature` et `saison`, `>=` et `EQ`, et `30` et `été`.

2.2 Base de faits

Puisqu'il s'agit de concevoir un système expert d'ordre 0+, nous avons choisi d'implémenter nos faits selon la forme suivante :

```
(attribut EQ valeur)
```

La base de faits est stockée dans une variable globale `*faits*` initialement vide : elle se remplira au cours de l'exécution du système. Il s'agira d'une liste de triplets.

Voici quelques attributs que nous utiliserons pour modéliser les faits :

Attributs	Signification
`Application`	le type d'application à développer
`Machine`	le type d'OS utilisé pour développer
`Cible`	le type d'OS visé pour l'application
`Budget`	le budget du développeur
`Precision`	pour préciser l'utilisation
`Utilisation`	précise le type d'utilisation

TABLE 1 – Exemple d'attributs pour les faits

`Propositions` sera l'attribut du faits utilisé pour stocker les différentes propositions inférées par *Cactus*.

2.3 Base de règles

Nous avons décidé d'implémenter notre base de règle de cette façon :

```
(((Premisse1 operateur valeur)...(PremisseN operateur valeur))
    ((Resultat1 operateur valeur)...(Resultat1M operateur valeur)))
```

où `operateur` $\in \{$`=`, `<`, `>`, `EQ`$\}$.

```
(defparameter *regles* '(

  ; Site web
  (((Application EQ Site-Web) (PrecisionSite EQ Simple) (Interaction-Dynamique EQ OUI))
    ((Propositions EQ (HTML PHP MySQL JavaScript))))
  (((Application EQ Site-Web) (PrecisionSite EQ Simple) (Interaction-Dynamique EQ NON))
    ((Propositions EQ (HTML PHP MySQL))))
  (((Application EQ Site-Web) (PrecisionSite EQ Responsive))
    ((Propositions EQ (HTML PHP MySQL BootStrap JavaScript))))
  (((Application EQ Site-Web) (PrecisionSite EQ Efficace) (RoRvsDjango EQ Configurable))
    ((Propositions EQ (HTML Django))))
  (((Application EQ Site-Web) (PrecisionSite EQ Efficace) (RoRvsDjango EQ Populaire))
    ((Propositions EQ (HTML Ruby-on-Rails))))

  ; Application Mobile
  (((Application EQ Mobile)(Machine EQ Mac)(Cible EQ iPhone) (Budget > 100))
    ((Propositions EQ (Swift))))
  (((Application EQ Mobile) (Cible EQ Android))
    ((Propositions EQ (JAVA Android-Studio SDK-Android))))
  (((Application EQ Mobile) (Cible EQ Android))
    ((Propositions EQ (JAVA Android-Studio SDK-Android))))

  ; Logiciel
  (((Application EQ Logiciel) (PrecisionLogiciel EQ Complexe))
    ((Propositions EQ (C++ JAVA Git))))
  (((Application EQ Logiciel) (PrecisionLogiciel EQ Complexe) (Machine EQ Windows))
    ((Propositions EQ (C# Git))))

  ; Calcul-Numerique
  (((Application EQ Calcul-Numerique) (Usage EQ Individuel) (Budget > 100))
    ((Propositions EQ (Matlab))))
  (((Application EQ Calcul-Numerique) (Budget < 100) (ManipulationMatrice EQ OUI))
    ((Propositions EQ (Octave Scilab Julia Fortran))))
  (((Application EQ Calcul-Numerique) (Budget < 100) (ManipulationMatrice EQ NON))
    ((Propositions EQ (Python MathPlotLib Numpy))))

  ; Machine-Learning
  (((Application EQ Machine-Learning) (PrecisionML EQ Prototypage-Rapide) (Budget < 101) (ManipulationMatrice EQ OUI))
    ((Propositions EQ (Octave))))
  (((Application EQ Machine-Learning) (PrecisionML EQ Prototypage-Rapide) (Budget < 101) (ManipulationMatrice EQ NON))
    ((Propositions EQ (Python Sci-kit MathPlotLib Numpy))))
  (((Application EQ Machine-Learning) (PrecisionML EQ Prototypage-Rapide) (Budget > 100))
    ((Propositions EQ (Matlab))))
  (((Application EQ Machine-Learning) (PrecisionML EQ Modele-Complexe) (Budget
```

```
51      ((Propositions EQ (LISP))))
52    (((Application EQ Systeme-Expert) (Parenthese EQ Non-Supportee))
53      ((Propositions EQ (Prolog))))
54
55    ; Applet
56    (((Application EQ Mobile) (Machine EQ Mac) (Cible EQ Mac))
57      ((Propositions EQ (Swift))))
58    (((Application EQ Applet) (Usage EQ Personnel))
59      ((Propositions EQ (Pygame Tkinter))))
60
61    ; Jeu-Video
62    (((Application EQ Jeu-Video) (PrecisionJeu EQ 3D))
63      ((Propositions EQ (C++ Unity3D OpenGL))))
64    (((Application EQ Jeu-Video) (PrecisionJeu EQ RPG-2D))
65      ((Propositions EQ (RPG-Maker))))
66    (((Application EQ Jeu-Video) (PrecisionJeu EQ 2D) (Cible EQ Mac) (Budget <
    ↪  35))
67      ((Propositions EQ (SpriteKit Swift Pygame))))
68    (((Application EQ Jeu-Video) (PrecisionJeu EQ 2D) (Cible EQ Mac) (Budget >
    ↪  34))
69      ((Propositions EQ (C++ Unity3D OpenGL))))
70    (((Application EQ Jeu-Video) (PrecisionJeu EQ 2D) (Cible EQ Windows) (Budget
    ↪  < 35))
71      ((Propositions EQ (Pygame))))
72    (((Application EQ Jeu-Video) (PrecisionJeu EQ 2D) (Cible EQ Windows) (Budget
    ↪  > 34))
73      ((Propositions EQ (Pygame))))
74    (((Application EQ Jeu-Video) (PrecisionJeu EQ 2D) (Cible EQ Linux) (Budget <
    ↪  35))
75      ((Propositions EQ (C++ Unity3D OpenGL))))
76    (((Application EQ Jeu-Video) (PrecisionJeu EQ 2D) (Cible EQ Linux) (Budget >
    ↪  34))
77      ((Propositions EQ (C++ Unity3D OpenGL))))
78
79    ; DIY
80    (((Application EQ DIY) (AccesAInternet EQ OUI))
81      ((Propositions EQ (Arduino))))
82    (((Application EQ DIY) (AccesAInternet EQ OUI))
83      ((Propositions EQ (Raspberry-Pi))))
84
85    ;Système Embarqué
86    (((Application EQ Systeme) (Machine EQ Linux) (PrecisionSysteme EQ
    ↪  Interaction))
87      ((Propositions EQ (C Shell Tkinter))))
88    (((Application EQ Systeme-Embarque))
89      ((Propositions EQ (Assembleur Shell C))))
90
91    ;Système Multi-agents
92    (((Application EQ SMA) (PrecisionSMA EQ SimulationDeFoule))
93      ((Propositions EQ (MASSIVE))))
94    (((Application EQ SMA) (PrecisionSMA EQ Trading))
95      ((Propositions EQ (MetaTrader4))))
96    (((Application EQ SMA) (PrecisionSMA EQ Autre))
97      ((Propositions EQ (JADE Java))))
```

2.4 Base de connaissances

Pour donner les informations concernant les technologies choisies par *Cactus* nous avons décider d'implémenter une base de connaissances. Celle-ci contient pour chaque technologie une brève description de celle-ci.

Les éléments des cette base sont représentés sous la forme de *liste pointée* ainsi :

```
(technologie . "La description de la technologie")
```

```
(defparameter *technologies*
'(

 (C . "Un des langages les plus populaires et très bien structuré ; utilisé pour accéder à la mémoire de la machine, pour la création de système d'exploitation et pour de nombreuses autres choses en programmation système.")
 (Shell . "Interpréteur de commandes : l'interface entre l'OS et l'utilisateur ; il existe différente famille, le plus utilisé restant 'bash'.")

 (Python . "Langage interprété de prototypage. Efficace et très simple d'utilisation ; de nombreuses bibliothèques.")
   (Tkinter . "Bibliothèque Python : permet de créer des interfaces graphiques.")
   (PyGame . "Bibliothèque Python : permet de créer des petits jeux et applications graphiques.")
   (Django . "FrameWork de développement web Python : très bien construit et extremement efficace une fois maîtrisé 'Django : the web framework for perfectionists with deadlines'.")
   (Numpy . "Bibliothèque python : boîte à outils scientifiques.")
   (MathPlotLib . "Bibliothèque python : utile pour la visualisation de données tracé de courbes, de surfaces, de nuages de points....")
   (Sci-kit . "Bibliothèque python : dispose de beaucoup d'implémentation d'algorithme de machine learning.")

 (R . "Utilisé dans les domaines scientifiques (particulièrement en statistiques et data-mining).")
 (MatLab . "Le langage reconnu pour ces fonctionalité en analyse numérique et calcul scientifique. Possède de très nombreuses fonctionalités mais est payant.")
 (Octave . "Logiciel et langage de programmation de calcul numérique, alternative libre et gratuite à Matlab.")
 (Scilab . "Logiciel et langage de programmation de calcul numérique, alternative libre et gratuite à Matlab.")
 (Julia . "Logiciel et langage de programmation de calcul numérique, alternative récente et libre à Matlab. Performant.")

 (Maple . "Logiciel de calcul formel ; propriétaire et payant.")
 (Sage . "Se veut être ''une alternative viable libre et open source à Magma, Maple, Mathematica et Matlab''. C'est un langage de calcul formel.")

 (LISP . "Le langage de programmation pour intellience artificielle symbolique le plus populaire de tous. Il est minimaliste et permet d'implémenter de nombreuses choses.")
 (Prolog . "Un langage de programmation français (Cocorico !), il est utilisé en Intelligence Artificielle.")
 (OWL-Lite . "Langage de représentation des connaissances construit sur le modèle de données de RDF. Basé sur la logique de description, Web Ontology Language (OWL) permet de définir des ontologies web structurées. OWL-Lite en est sa version la plus simple, pour les utilisateurs ayant besoin d'une hiérarchie de classification et de contraintes simples.")
```


Programme 2: Base de connaissances *technologies*

```
27   (OWL-DL . "Langage de représentation des connaissances construit sur le
     ↪ modèle de données de RDF. Basé sur la logique de description, Web Ontology
     ↪ Language (OWL) permet de définir des ontologies web structurées. OWL-DL en
     ↪ est une version intermédiaire décidable qui permet une expressivité
     ↪ maximale avec une garantie de calcul.")
28   (OWL-Full . "Langage de représentation des connaissances construit sur le
     ↪ modèle de données de RDF. Basé sur la logique de description, Web Ontology
     ↪ Language (OWL) permet de définir des ontologies web structurées. OWL-Full
     ↪ en est sa version la plus libre, indécidable, permettant une expressivité
     ↪ maximale mais sans garantie de calcul.")
29
30   (MASSIVE . "Logiciel utilisant les Systèmes Multi-agents (SMA) dans le but
     ↪ de simuler des foules. Il est très utilisé dans les industries du cinéma
     ↪ et du jeu vidéo, notamment dans la trilogie Seigneur des Anneaux ainsi que
     ↪ dans la série Games of Thrones.")
31   (MetaTrader4 . "Logiciel utilisant les Systèmes Multi-agents (SMA) dans le
     ↪ domaine de la finance afin d'effectuer du trading automatique.")
32   (JADE . "JAVA Agent DEvelopment (JADE) est une plateforme de programmation
     ↪ multi-agent implémentée en Java.")
33
34   (SQL . "Le standard des bases de données relationnelles : c'est à la fois un
     ↪ langage et une technologies d'implémenation de BDD ; connait plusieurs
     ↪ variantes d'implémentations.")
35     (PLSQL . "Variante propriétaire d'Oracle ; permet d'implémenter des bases
     ↪ de données relationnelles-objet et dispose de fonctions et fonctionnalités
     ↪ supplémentaires.
36     Les bases de données relationnelles-objet sont très efficaces lorsqu'il
     ↪ s'agit de construire des bases de données dont l'accès en lecture est très
     ↪ fréquent.")
37     (PostGreSQL . "Variantes libre d'implémentation de SQL la plus populaire.
     ↪ Elle reste la plus plebiscitée car elle est très robuste en terme
     ↪ d'implémentation et veille à suivre de très près les standard SQL.")
38     (MySQL . "Variantes libre d'implémentation de SQL, simple d'utilisation
     ↪ mais aussi moins bien structurée.")
39
40   (VBA . "Permet de créer des applications légères dans Excel pour traiter des
     ↪ petits ensemble de données ; solution accessible aux non informaticiens
     ↪ car elle permet de créer facilement des petits formulaires et
     ↪ applications.
41    Cependant c'est une solution payante.")
42
43   (Arduino . "Cartes électroniques de 'hacking' en license libre. Elles se
     ↪ programment généralement dans un formalisme proche du C et du C++")
44
45   (HTML . "Formalisme de représentation de données utilisée par les pages web.
     ↪ La base du développement web depuis toujours. Il peut être aussi utilisé
     ↪ pour créer des documents.")
46     (JavaScript . "Langage de programmation permettant de faire fonctionner
     ↪ des applications web côté client.")
47       (NodeJS . "Plate-forme construite sur le moteur d'exécution JavaScript
     ↪ de Google Chrome. Permet de créer facilement des applications réseau
     ↪ rapides, évolutives et scalables.")
48       (AJAX . ", pour 'Asynchronous JAvascript and Xml'. Architecture qui
     ↪ permet de créer des applications dynamiques. Est utilisé pour les sites
```

Nous utiliserons cette base de connaissances dans la fonction `afficherPropositions()` que nous détaillerons plus bas : l'idée est d'avoir un petit descriptif des technologies proposées pour comprendre en quoi elles sont pertinentes.

3 Fonctionnement du système

Comme nous l'avons évoqué au début, il existe plusieurs moyens de réaliser le moteur d'inférence : le chaînage avant et le chaînage arrière. De même pour chacune de ces façons de procéder, on peut choisir de parcourir l'arbre de déduction en profondeur ou en largeur.

3.1 Chaînage avant

Le chaînage avant a pour avantage d'être facilement implémentable et ne repose pas sur la recherche d'une réponse particulière contrairement au chaînage arrière dont l'algorithme peut se construire sur la recherche d'un but spécifique (comme nous avons pu le voir en TD dans le cas du chaînage arrière en profondeur d'abord avec les fonctions `verifier()` et `verifierET()`).

Voici un algorithme itératif pour le chaînage avant.

Algorithme 1 : Chaînage avant

```
début
    faire
        pour chaque règle r de la base de règle BR faire
            si r est déclenchable alors
                EC ← EC ∪ {r};
                BR ← BR − {r};
            si EC ≠ ∅ alors
                BF ← BF ∪ conclusions(r);
            sinon
                poser une question;
    tant que il n'y a pas de propositions dans BF;
    afficher les propositions ;
```

Cet algorithme permet de poser des questions à l'utilisateur lorsque le système ne peut plus inférer. L'utilisateur sera amené a préciser des valeurs d'attributs pour que le moteur puisse ainsi construire de nouveaux faits. On remarquera que le système s'arrêtera dès que des propositions auront été inférées par le moteur.

3.1.1 En largeur

Nous avons tout d'abord réalisé un moteur d'inférence en chaînage avant et en profondeur d'abord.

Nous l'avons implémenté sous LISP de cette façon :

```
1  (defun chainageAvantLarg () ; Moteur chaînage avant en largeur
2    (let (EC regleCourante)
3      (loop ; on boucle
4        (if (valeur (assoc 'Propositions *faits*)) ; si le but est présent dans
   ↪  la base de faits avec une valeur non nulle
5          (progn
6            (funcall 'afficherPropositions) ; on affiche les propositions
7            (return nil)) ; et on arrête
8          (dolist (r *regles*) ; sinon on parcourt les règles dans la base de
   ↪  règles
9            (when (declenchable? r); si une règle est déclenchable
10             (setq EC (append EC (list r))) ; on l'ajoute à l'ensemble
   ↪  contraint EN FIN
11             (setq *regles* (remove r *regles* :test 'equal))))) ; on l'enlève
   ↪  de la base de règles
12       (if EC ; si on peut encore déclencher des règles
13         (progn
14           (setq regleCourante (pop EC)) ; on choisit la dernière obtenue
15           (ajouter (conclusion regleCourante))) ; on ajoute son résultat à la
   ↪  base de faits
16         (askQuestion))))) ; sinon on pose une question
```

Programme 4: Chainage avant – Parcours en largeur

On remarquera l'importance des lignes 10 et 14 pour le parcours en largeur : l'ajout ligne 10 de `r` en fin de l'ensemble contraint `EC` et l'exécution ligne 14 de la règle en tête de `EC` permettent d'ordonner les règles afin que les premières ajoutées soient les premières utilisées. On utilise ainsi *une structure de file* (aussi appelée FIFO pour "*First In, First Out*").

3.1.2 En profondeur

En modifiant légèrement le programme, on passe facilement du parcours en largeur au parcours en largeur.

```
1  (defun chainageAvantProf () ; Moteur chaînage avant en profondeur
2    (let (EC regleCourante)
3      (loop ; on boucle
4        (if (valeur (assoc 'Propositions *faits*)) ; si le but est présent dans
   ↪  la base de faits avec une valeur non nulle
5          (progn
6            (funcall 'afficherPropositions) ; on affiche les propositions
7            (return nil)) ; et on arrête
8          (dolist (r *regles*) ; sinon on parcourt les règles dans la base de
   ↪  règles
9            (when (declenchable? r); si une règle est déclenchable
10             (push r EC) ; on l'ajoute à l'ensemble contraint EN TÊTE
11             (setq *regles* (remove r *regles* :test 'equal))))) ; on l'enlève
   ↪  de la base de règles
12       (if EC ; si on peut encore déclencher des règles
13         (progn
14           (setq regleCourante (pop EC)) ; on choisit la dernière obtenue
15           (ajouter (conclusion regleCourante))) ; on ajoute son résultat à la
   ↪  base de faits
16         (askQuestion))))) ; sinon on pose une question
```

Programme 5: Chainage avant – Parcours en profondeur

Ce qui fait la différence ici se trouve à la ligne 10 : on ajoute cette fois-ci `r` en tête de `EC`. On passe alors d'une structure de file à une structure de pile (aussi appelée LIFO pour "*Last In, First Out*".

3.2 Fonctions outils

Afin d'abstraire les raisonnements nous avons mis au point des fonctions outils. `conclusion()`, `declenchable?()` et `ajouter()` sont celles mises en place pour les règles.

```lisp
;;Fonctions outils pour les règles
(defun conclusion (r)
  ; Renvoie les conclusions d'une règle
  (cadr r))

(defun declenchable? (r)
  ; Renvoie t si la règle est déclenchable ; nil sinon
  (let (
      (OK t)
      (premisses (car r)))
    (dolist (p premisses OK) ; pour chaque premisses de r
       ; si l'attribut a une valeur numérique mais n'est pas présent dans les faits
      (if (and (numberp (valeur p)) (not (valeur (assoc (attribut p) *faits*))))
          (setq OK nil)
           ; s'il est présent mais si sa valeur ne vérifie pas la prémisse p
          (if (not (funcall (operateur p) (valeur (assoc (attribut p) *faits*)) (valeur p)))
           (setq OK nil))))))

(defun ajouter (resultats)
  ; Ajoute un résultat à la base de faits *faits*
  (dolist (triplet resultats)
    (if (assoc (attribut triplet) *faits*) ; si l'attribut est déjà présent dans la base
      (setf (valeur (assoc (attribut triplet) *faits*)) (valeur triplet)) ; on remplace sa valeur
      (push triplet *faits*)))) ; sinon on rajoute triplet à la base de faits
```

Programme 6: Fonctions outils pour les règles

`attribut()`, `operateur()` et `valeur()` sont celles mises en place pour les faits.

```lisp
;; Fonctions outils pour les faits (triplets)
(defun attribut (triplet)
  (car triplet))

(defun operateur (triplet)
  (cadr triplet))

(defun valeur (triplet)
  (caddr triplet))
```

Programme 7: Fonctions outils pour les faits

3.3 Poser une question : fonction `askQuestion()`

Une possibilité lorsque le système n'arrive plus à tirer de conclusions et de poser des questions à l'utilisateur pour apporter de nouveaux faits. C'est ici le rôle réalisé par la fonction

```
askQuestion().

(defun questionAssociee (attribut)
  (if (cdr (assoc attribut *questions*))
    (cdr (assoc attribut *questions*))
    attribut))

(defun askQuestion ()
  (let ((attribut (car (set-difference (listeAttRegles) (listeAttFaits)))
  ↪ valeur))
    ; "attribut" est le premier élément de la différence entre :
    ; - la liste des attributs dans la base de faits
    ; - la liste des attributs dans la base de règles (prémisses)
    ; C'est-à-dire un attribut dont la valeur est inconnue.
    (if attribut
      (if (numberp (car (AttValues attribut))) ; la valeur de celui-ci
  ↪ est-elle un nombre ?
          (until
            (AND
              (not (format t "------~&~S~%------~%Votre choix (nombre) : "
  ↪ (questionAssociee attribut)))
              (numberp (setq valeur (read))))) ; Redemande tant que son choix
  ↪ n'est pas valide
          (until
            (AND
              ; liste les valeurs possibles de l'attribut et fait lire un choix
  ↪ à l'utilisateur
              (not (format t "------~&~S~%------~%~S~%~%Votre choix : "
  ↪ (questionAssociee attribut) (delete-duplicates (AttValues attribut))))
              (member (setq valeur (read)) (delete-duplicates (AttValues
  ↪ attribut))))))
              ; Redemande tant que son choix n'est pas valide
      (error "Nous n'avons rien pu trouver."))
    (sleep 1)
    (pushnew (list attribut 'EQ valeur) *faits*)))
```

Programme 8: Fonction `askQuestion()` permettant de récupérer des informations

Nous utilisons d'autres fonctions pour réaliser certaines tâches.

```lisp
(defun listeAttFaits ()
  (loop for fait in *faits*
    collect (attribut fait)))

(defun listeAttRegles ()
  (loop for regle in *regles*
    append (loop for premisse in (car regle)
      collect (attribut premisse))))

(defun AttValues (attribut)
  (loop for regle in *regles*
    if (assoc attribut (car regle))
    collect (valeur (assoc attribut (car regle)))))
```

Programme 9: Fonctions outils pour `askQuestion()`

`listAttFaits()` récupère la liste des attributs . `askQuestion()`. `askQuestion()`.

3.4 Afficher les propositions du système : fonction `afficherPropositions()`

C'est dans cette fonction que nous utilisons la base de connaissances `**technologies**`.

```lisp
(defun technologiesAssociee (attribut)
  (if (cdr (assoc attribut *technologies*))
    (cdr (assoc attribut *technologies*))
    attribut))

(defun afficherPropositions ()
  (let ((prop (caddr (assoc 'Propositions *faits*))) (overlay
  ↪ '------------------))
    (format t "~&~A~&Voici les différentes technologies que je vous propose :
  ↪ " overlay)
    (dolist (x prop)
      (format t "~& -» ~A : ~S" x (technologiesAssociee x)))
    (format t "~&~A~&" overlay)))
```

Programme 10: Fonction `afficherPropositions()` qui affiche les propositions du système expert

⋆ ⋆ ⋆

Conduite d'expertise d'un SE d'ordre 0+

Dates de remise :

- **Lundi 28 novembre 2016 à 18H pour la réponse à la question 1.**
- **Lundi 9 janvier 2017 à 18H pour la réponse aux questions 2 et 3.**
- **Démonstration et présentation orale lors du dernier TD.**

L'objet du TP03 est de réaliser le développement d'un SE de sa phase d'expertise à sa phase d'utilisation. A cette fin, vous devez :

1 - Formalisez une problématique d'un domaine au choix (un qui vous passionne) qui puisse être traitée par un SE d'ordre 0+. Justifiez votre choix et faites-le valider par votre chargé de TD.
2 - Déterminez les connaissances nécessaires au SE : explicitez votre base de règles (donnez vos sources). Présentez l'arbre de déduction associé et donnez des jeux d'essais.
3 - Programmez votre SE
    a. Justifiez la représentation Lisp choisie pour exploiter les faits et la base de règles.
    b. Développez, justifiez et commentez le moteur d'inférences choisi : chaînage avant (ou arrière) en profondeur d'abord, chaînage avant (ou arrière) en largeur d'abord.
    c. Testez votre moteur et commentez les résultats. Une comparaison avec un deuxième moteur développé serait un plus.

Documents à produire :

- Un rapport écrit comportant les réponses aux points précédents et présentant des scénarios d'utilisation.
- Un fichier comportant le code lisp de votre SE avec les scénarios d'utilisation (à envoyer par courriel).
- Une courte présentation orale s'appuyant sur des transparents et une démonstration sont attendues au cours du dernier TD.