Système Expert

I- Introduction

Depuis le milieu des années 70 un certain nombre de systèmes issus de recherches en Intelligence Artificielle sont désignés comme des *systèmes experts* ou *systèmes à base de connaissances*. Ces logiciels suscitent un important engouement dans certains milieux professionnels parce qu'ils abordent, avec un relatif succès, certaines tâches concrètes, jusqu'ici faiblement informatisées, de type diagnostic, prévision, conception, planification d'actions…

Ces systèmes travaillent sur des domaines bien définis et sont conçus à base de *connaissances déclaratives* appelées *règles*. Un des grands principes de ces systèmes est que les connaissances utilisées sont bien **séparées** de leurs procédures d'utilisation, appelées *moteurs d'inférences.*

Exemple de règle d'un système expert de diagnostic médical (MYCIN) :

si la culture est le sang
et l'organisme est à Gram négatif
et l'organisme est sous forme de bâtonnet
alors il est probable (0.6) que l'organisme est le pseudomonia aeruginosa

Exemple de règle pour l'analyse de mesure de pendage (DIPMETER ADVISOR) :

si la faille est de couleur rouge
et la direction est perpendiculaire à la faille
et la longueur est supérieure à 60 mètres
alors il s'agit d'une faille synsédimentaire

Un système expert est, au sens strict, un logiciel dont l'ambition est de reproduire le raisonnement d'un expert dans un domaine de compétence spécialisé. Les premiers exemples de tels systèmes ont ainsi été développés au tout début des années 70 pour l'aide au diagnostic, qu'il soit médical ou technique.

Mais on trouvera aussi bien des systèmes experts dans le domaine de la prospection minière, l'archéologie, la généralisation cartographique, l'automatisation de procédures, les contrats d'assurance, l'orientation professionnelle, etc.

1. Définitions

Dans ce travail, nous nous limiterons à des règles simples d'ordre 0 fondées sur le calcul propositionnel.

- Une *proposition* est une expression quelconque qui peut être évaluée à vrai ou à faux (ex : « le patient a un écoulement nasal »).
- Un *fait* est une proposition évaluée à vrai.
- Une *règle d'inférence* se présente sous la forme *prémisse* → *conclusion*
  où *prémisse* = ensemble de propositions
  et *conclusion* = une conclusion à déduire de la prémisse
- Une *base de connaissances* est composée d'un ensemble de *faits* et de *règles d'inférence*.
- Un *moteur d'inférence* est un programme qui permet, à partir d'une *base de règles* et d'une *base de faits*, de déduire de nouveaux faits.
- Un *système expert* est composé d'une *base de connaissances* et d'un *moteur d'inférence*.

2. Les moteurs d'inférences d'un système expert

Une règle de la forme :

si condition 1
et condition 2
et condition 3
…
alors conclusion

doit être interprétée comme :

si la condition_1 ***et*** *la condition_2* ***et*** *la condition_3 … sont vraies*
alors on peut ***inférer*** *la conclusion*

Trois modes généraux d'enchaînement des règles peuvent être utilisés :

- le chaînage avant : on part des faits pour arriver au but
- le chaînage arrière : on part d'un but et on tente de remonter la chaîne afin de déterminer les faits qui permettent de le déduire
- Le chaînage mixte

Dans le cadre de ce projet, seul le chaînage avant est étudié.

II- Version de base

1. La base de connaissances

1.1 Les propositions

Les propositions sont constituées de chaînes de caractères, sans espace (on peut substituer à l'espace le souligné "_") : *vole, mammifere, feuilles, fièvre, couleur_rouge, etc.*

1.2 La base de règles

Règle simple

La base de règles est fournie dans un fichier texte sous la forme :

> *nombre_règles*
> *# commentaire*
> *si condition_1*
> *et condition_2*
> *et condition_3*
> *…*
> ***alors*** *conclusion*

Règle finale

Parmi les règles, on distingue les règles qui apportent une conclusion finale :

> *# conclusion finale*
> *si condition_1*
> *et condition_2*
> *…*
> ***solution*** *conclusion*

Les conditions et conclusions sont des propositions.
Le fichier peut contenir des lignes blanches et des lignes de commentaires qui débutent par un "#"

1.3 Le chaînage avant

Dans le mode de chaînage avant, le moteur d'inférence part des faits pour arriver au but. L'algorithme qu'on propose de mettre en œuvre est décrit ci-dessous.

```
début
liste des règles actives ← base des regles initiale fournie dans un fichier
liste des faits ← fournie par l'utilisateur
tant qu'il existe des règles actives:
 liste des regles declenchables ← filtrage des règles déclenchables parmi les règles actives
 si aucune règle n'est déclenchable alors retourner la liste des faits, arrêter l'inférence
 # résolution des conflits
 sélectionner une des règles déclenchables R (par exemple la première)
      declencher la règle R
      si la règle est une règle finale alors retourner la conclusion, arrêter l'inférence
      supprimer de la liste des règles actives les règles qui amènent à la même conclusion que R
fin tant que
retourner la liste des faits
fin
```

La *résolution des conflits* qui permet de sélectionner une des règles déclenchables à appliquer peut être plus élaborée et sélectionner par exemple la règle avec le plus de conditions, ou appliquer toutes celles qui sont déclenchables (stratégie en largeur d'abord), etc.

2. Travail demandé

La classe Regle

- ✓ Définir la classe *Regle* qui permet de modéliser une règle de la base
  - Attributs
    - *premisse_ (liste des conditions)*
    - *conclusion_*
  - Méthodes principales
    - la méthode *__str__* qui permet d'afficher une règle à l'aide du *print*
    - l'accesseur *getConclusion* qui retourne la conclusion de la règle
    - *est_declenchable* : vérifie si les conditions de la règle sont toutes vérifiées par rapport à une liste de faits passée en paramètre
    - *declenche* : déclenche la règle, c'est à dire ajoute la conclusion dans la liste des faits et enlève la règle de la liste des règles actives
    - *est_regle_finale* : renvoie *False*
- La sous-classe *RegleFinale* en adaptant si besoin les différentes méthodes de la classe *Regle*

Lecture du fichier des règles

- ✓ Ecrire une fonction (hors de la classe) *charge_fichier_regles* qui retourne une liste de règles à partir du nom du fichier passé en paramètre.

La classe SystemeExpert

Définir la classe *SystemeExpert* qui permet de modéliser un système expert.

- Attributs :
  - *base de règles* (liste d'instances de la classe *Regle*)
- Ecrire l'ensemble des méthodes permettant de mettre en œuvre le chaînage avant.
- Tester votre programme à partir de la base de règles fournie sur Arche.

III- Version améliorée

Dans une base de règles, on peut disposer de plusieurs règles qui apportent la même conclusion. Lorsque cette conclusion est inférée, on peut donc considérer que ces autres règles ne sont plus utiles. Dans l'objectif d'améliorer le programme en évitant d'examiner des règles inutiles, on introduit un dictionnaire *dico_conclusions* tel que *dico_conclusions[proposition] = liste des règles qui apportent cette conclusion.*

Par exemple les deux règles suivantes concluent à une roche métamorphique, s'il l'une est déclenchée, alors il est inutile d'examiner ultérieurement la seconde.

```
si non_mineraux_visibles_oeil_nu
et non_effervescence_acide
et debit_en_feuillets
alors metamorphique
```

```
si mineraux_visibles_oeil_nu
et pas_de_pate
et mineraux_jointifs
et foliation
alors metamorphique
```

Ce dictionnaire est construit à partir de la base de règles de départ et mis à jour à chaque fois qu'une conclusion est inférée.

Compléter le programme précédent afin d'introduire un tel dictionnaire.

IV- Version avancée

Lorsqu'on manipule de grosses bases de règles, on structure l'ensemble des règles en *familles*, par exemple la famille de règles sur les roches métamorphiques, celle sur les roches magmatiques, etc.

Afin de ne charger ces règles qu'en cas de besoin, on distingue une nouvelle sous-classe de règles qui vont permettre de charger de nouvelles familles de règles. Ces règles apparaissent dans les fichiers sous la forme :

> *si condition_1*
> *et condition_2*
> *et condition_3*
> *…*
> ***famille*** *conclusion*

Les règles de famille

- Décrire la sous-classe *RegleFamille*
- Adapter la méthode *declenche* pour qu'elle charge en plus le nouveau fichier de règles
- et donc la fonction *charge_fichier_regles* pour qu'elle reconnaisse également les règles de ce nouveau type
- ainsi que la méthode *chainage_avant* afin qu'elle mette à jour le dictionnaire des conclusions dans ce cas
- Tester le programme sur les fichiers de règles fournis sur Arche.

1. Courte webographie

http://masig.ensg.eu/PDF/Systemes%20Experts.pdf
http://www.iro.umontreal.ca/~pift6802/pdf/05b_si4.pdf
http://philippe.morignot.free.fr/Articles/SYBR2.pdf
http://www-igm.univ-mlv.fr/~dr/XPOSE2004/jvaldes/index.html#1
http://www.lamsade.dauphine.fr/~maudet/cours/md2/md2-java-projet2.pdf
http://www.persee.fr/doc/bspf_0249-7638_1986_hos_83_10_8714
http://fr.akinator.com/
https://tel.archives-ouvertes.fr/tel-00529718/document