LO21 - Système expert

Guillaume RUFF
Driss KIHAL

Automne 2020

1 Introduction

Un système expert est composé d'une base de conaissances, une base de faits et un moteur d'inférence. C'est un outil capable de reproduire les mécanismes cognitifs d'un expert, dans un domaine particulier.

2 Definition des composants du système expert

Nous allons définir les composants d'un système expert dans les prochaines sections.

2.1 Proposition

Une proposition est composée de son identifiant et de sa valeur, son identifiant l'identifie parmis d'autres propositions sa valeur est soit vraie soit fausse mais ne peut pas être les deux à la fois.
Son identifiant peut-être une chaînes de caractères ou autre en tout cas , il est unique.

2.2 Règle

On défini une règle comme une liste de propositions divisées en 2 parties la prémisse et la conclusion qui est toujours le dernier élément de cette liste.
Soit E l'ensemble des propositions de l'univers.

On peut aussi définir une règle Γ comme ceci :

$$\begin{aligned}\Gamma :&(E)^n \longrightarrow E\\ &(P_1, P_2, ..., P_n) \longmapsto T\end{aligned}$$

Soit n $\in$ N. On définit l'application Γ par

$$\forall (P_1, P_2, ..., P_n) \in (E)^n, \Gamma(P_1, P_2, ..., P_n) = P_1 \wedge P_2 \wedge ... \wedge P_n = T$$

Avec T la conclusion de la règle et $(P_1, P_2, ..., P_n)$ les propositions de sa prémisse.

Cette définition certes rasoir permet une meilleur vue de ce que peut-être une règle cependant elle n'est pas adaptée à l'algorihtmique et ses implémentations.
Nous utiliserons alors la définition suivante se rapprochant de celle du cours :

Constructeurs :

- créerRègleVide (Chaîne de caractères) $\longrightarrow$ (Règle)

Modificateurs :

- ajoutPremisse (Règle X Proposition) $\longrightarrow$ (Règle), ajoute une proposition à la prémisse
- créerConclusion (Règle X Proposition) $\longrightarrow$ (Règle) , ajoute la conclusion
- suppProposition (Règle X chaîne de caractères) $\longrightarrow$ (Règle) , supprime une proposition de la prémisse

Observateurs et méthodes d'accès

- estVideRègle (Règle) $\longrightarrow$ (Booléen), vérifie si la règle est vide
- estVidePrémisse (Règle) $\longrightarrow$ (Booléen), vérifie si la prémisse d'une règle est vide
- têteRègle (Règle) $\longrightarrow$ (Proposition) , renvoi la tête de la règle
- queueRègle (Règle) $\longrightarrow$ (Proposition) , renvoi la queue de la règle
- succ (Proposition) $\longrightarrow$ (Proposition) , renvoi l'élément suivant de l'élément courant
- TestPremisseR (Règle X Proposition)$\longrightarrow$ (Booléen), vérifie si propostion donnée en argument est contenu dans la premisse de la règle

2.3 Definition de la base de connaissances

Une base connaissances est une liste de règles on ne fait aucune supposition sur leurs véracité en dehors du moteur d'inférence et de la base de faits. Elle contient l'ensemble des règles du problème.
Voici la définition du type abstrait "Liste de règles" :
Une liste de règles est une suite de règles chacune ayant un successeur et un prédecesseur sauf respectivement pour la tête (la première règle) et la queue (la dernière règle).

2.4 Definition de la base de faits

Une base de faits est une liste de propositions logiques, elle reprend la même définition qu'une liste de règles à la seul différence que tout les élements de cette liste sont des propositions.

2.5 Definition d'un moteur d'inférence

Un moteur d'inférence a pour objectif de déduire de la base de connaissances et de la base de faits, des faits certains. C'est donc une fonction prenant en entrée une liste de règles et une liste de propositions vraie et sortant une liste de faits certains.

3 Algorithmes et raisonnements

Dans cette sections nous écrirons les algorithmes utiles au fonctionnement du système expert. Ils auront cette structure générique :

3.1 Algorithmes et Règles

Ici sont écrits tout les algorithmes régissants les règles du système expert (voir section 2.2).

3.1.1 Contructeurs

Algorithm 1: créerRègleVide

Result: R : [Règle]
R $\longleftarrow$ RègleVide
créerRègleVide $\longleftarrow$ R

3.1.2 Modificateurs

Algorithm 2: ajoutPremisse

```
Data: R : [Règle], P : [Proposition]
Result: R : Règle
  if non(estVide(R)) then
    if estVide(Premisse(R)) then
      Premisse(R) ← P
    else
      if estVide(succ(Premisse(R))) then
        succ(Premisse(R)) ← P
      else
        p ← PropositionVide
        p ← Premisse(R)
        while succ(succ(p)) != INDEFINI do
          p ← succ(p)
        end while
        succ(succ(p)) ← succ(p)
        succ(p) ← P
      end if
    end if
  end if
  ajoutPremisse ← R
```

Algorithm 3: suppProposition

```
Data: R : [Règle], id : [Chaîne de caractères]
Result: [Règle]
  p ← [Règle] : R
  if estVideRègle(R) then
    suppProposition ← R
  else
    if preVide(R) then
      suppProposition ← R
    else
      while id(succ(e)) ≠ id do
        e ← succ(e)
      end while
      succ(e) ← succ(succ(e))
      suppProposition ← R
    end if
  end if
```

3.1.3 Observateurs

Algorithm 4: estVideRègle

```
Data: R : [Règle]
Result: [Booléen]
  p ← [Règle] : R
  b :← [Booléen] : Vrai
  while succ(p) ≠ INDEFINI do
    if p ≠ INDEFINI then
      b ← Faux
    end if
    p ← succ(p)
  end while
  estVideRègle ← b
```

Algorithm 5: estVidePrémisse

```
Data: R : [Règle]
Result: [booléen]
  p ← [Règle] : R
  if estVideRègle(R) then
    estVidePrémisse ← Vrai
  else
    if succ(p) = INDEFINI then
      estVidePrémisse ← Vrai
    else
      estVidePrémisse ← Faux
    end if
  end if
```

Algorithm 6: têteRègle

```
Data: R : [Règle]
Result: [Proposition]
  if non(estVidePrémisse(R) then
    têteRègle ← R
  else
    têteRègle ← INDEFINI
  end if
```

Algorithm 7: queueRègle

```
Data: R : [Règle]
Result: [Proposition]
  p ← [Règle] : R
  if non(estVidePrémisse(R) then
    while succ(p) ≠ INDEFINI do
      p ← succ(p)
    end while
    queueRègle ← p
  else
    queueRègle ← R
  end if
```

Algorithm 8: TestPremisseR

Data: R : [Règle], P : [Proposition]
Result: B : Booléen
  if estVideRègle(R) OU estVide(P) **then**
    $TestPremisseR \leftarrow VRAI$
  end if
  if estVidePrémisse(Premisse(R)) **then**
    $TestPremisseR \leftarrow FAUX$
  else
    if Premisse(P)=P **then**
      $TestPremisseR \leftarrow VRAI$
    else
      $TestPremisseR \leftarrow TestPremisseR(Reste(R), P)$
    end if
  end if