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 parmi 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 chaîne de caractères ou autre, il doit lui être unique.

2.2 Règle

On définit une règle comme une liste de propositions divisées en 2 parties : la prémisse qui est la première partie de cette liste de proposition amputée de la conclusion. Conclusion qui est toujours le dernier élément de la Règle.
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 abstraite permet une meilleure vue de ce que peut-être une règle cependant elle n'est pas adaptée à l'algorithmique et ses implémentations.

Nous utiliserons alors la définition suivante se rapprochant de celle du cours :

Constructeurs :

- créerRègleVide () ⟶ (Règle)
- créerProposition (chaîne de caractères X booléen) ⟶ (Proposition)

Modificateurs :

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

Observateurs et méthodes d'accès

- estVideRègle (Règle) ⟶ (Booléen), vérifie si la règle est vide
- estVidePrémisse (Règle) ⟶ (Booléen), vérifie si la prémisse d'une règle est vide
- têteRègle (Règle) ⟶ (Proposition) , renvoie la tête de la règle
- queueRègle (Règle) ⟶ (Proposition) , renvoie la queue de la règle
- id(Proposition) ⟶ (Chaîne de caractère), renvoie l'identifiant d'une proposition
- succ (Proposition) ⟶ (Proposition) , renvoie l'élément suivant de l'élément courant
- TestPremisseR (Règle X Chaîne de caractère)⟶ (Booléen), vérifie si propostion dont l'id donnée en argument est contenu dans la premisse de la règle

2.3 Definition de la base de connaissances

Une base de connaissances est une liste de règles. On ne fait aucune supposition sur la véracité des propositions qui composent les règles 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 "BC" :
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).
Constructeurs :

- créerBCVide () ⟶ (BC)

Modificateurs :

- ajoutRègle (BC X Règle) ⟶ (BC)

Observateurs :

- têteBC (BC) ⟶ (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 seule différence qu'il n'y a pas de notions de prémisse ou conclusion .

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. On peut alors l'imaginer comme une fonction ayant comme paramètre une liste de règle c'est-à-dire une base de connaissances et une liste de proposition vraies, la base de faits. Cette dernière renvoie une nouvelle base de faits, résultat du parcourt des règles.

3 Algorithmes et raisonnements

Dans cette sections nous écrirons les algorithmes utiles au fonctionnement du système expert.

3.1 Algorithmes et Règles

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

3.1.1 Constructeurs

Algorithm 1: créerRègleVide

Result: R : [Règle]
$R \leftarrow [Règle]$
$créerRègleVide \leftarrow R$

Lexique :

- R : Un règle vide qui sera renvoyée

Algorithm 2: créerProposition

Result: R : [Règle]
Data: id : [Chaine de caractère]
$P \leftarrow [Proposition]$
$id(P) \leftarrow id$
$créerProposition \leftarrow P$

Lexique :

- P : Une nouvelle proposition qui sera renvoyée
- id : id de la proposition

3.1.2 Modificateurs

Algorithm 3: ajoutPremisse

Data: R : [Règle], P : [Proposition]
Result: R : Règle
  if non(estVide(R)) **then**
    $p \leftarrow [Proposition] : têteRègle(R)$
    while succ(p) != INDEFINI **do**
      $p \leftarrow succ(p)$
    end while
    $succ(p) \leftarrow p$
    $p \leftarrow P$
  end if
  $ajoutPremisse \leftarrow R$

Lexique :

- R : La règle où l'on souhaite ajouter une proposition
- P : Proposition que l'on souhaite ajouter
- p : Proposition qui prendra succesivement la valeur des propositions de la prémisse de R

Algorithm 4: créeConclusion

Data: R : [Règle], P : [Proposition]
Result: R : Règle
  if (estVide(R)) **then**
    $R \leftarrow P$
  else
    $p \leftarrow têteRègle(R)$
    while succ(p) != INDEFINI **do**
      $p \leftarrow succ(p)$
    end while
    $succ(p) \leftarrow P$
  end if
  $créConclusion \leftarrow R$

Lexique :

- R : La règle où l'on souhaite ajouter une conclusion
- P : Proposition que l'on souhaite ajouter en temps que conclusion
- p : Proposition qui prendra succesivement la valeur des propositions de la prémisse de R

Algorithm 5: suppProposition

Data: R : [Règle], id : [Chaîne de caractères]
Result: [Règle]
  $p \leftarrow [Proposition] : têteRègle(R)$
  if estVideRègle(R) **then**
    $suppProposition \leftarrow R$
  else
    if estVidePrémisse(R) **then**
      $suppProposition \leftarrow R$
    else
      while id(succ(e)) $\neq$ id **do**
        $e \leftarrow succ(e)$
      end while
      $succ(e) \leftarrow succ(succ(e))$
      $suppProposition \leftarrow R$
    end if
  end if

Lexique :

- R : La règle où l'on souhaite supprimer une proposition
- id : Nom de la proposition que l'on souhaite surpprimer
- e : Proposition qui prendra succesivement la valeur des propositions de la prémisse de R

3.1.3 Observateurs

Algorithm 6: estVideRègle

Data: R : [Règle]
Result: [Booléen]
  if R $\neq$ INDEFINI **then**
    $estVideRègle \leftarrow Faux$
  else
    $estVideRègle \leftarrow Vrai$
  end if

Lexique :

- R : La règle que l'on souhaite tester

Algorithm 7: estVidePrémisse

Data: R : [Règle]
Result: [booléen]
  $p \leftarrow [Proposition] : têteRègle(R)$
  if estVideRègle(R) **then**
    $estVidePrémisse \leftarrow Vrai$
  else
    if succ(p) = INDEFINI **then**
      $estVidePrémisse \leftarrow Vrai$
    else
      $estVidePrémisse \leftarrow Faux$
    end if
  end if

Lexique :

- R : La règle que l'on souhaite tester
- p : Proposition qui prend la valeur de la tête de la premisse de R

Algorithm 8: têteRègle

Data: R : [Règle]
Result: [Proposition]
  if non(estVideRègle(R)) **then**
    $têteRègle \leftarrow R$
  else
    $têteRègle \leftarrow INDEFINI$
  end if

Lexique :

- R : La règle dont on souhaite obtenir la tête de la premisse

Algorithm 9: queueRègle

Data: R : [Règle]
Result: [Proposition]
  $p \leftarrow [Proposition] : têteRègle(R)$
  if non(estVidePrémisse(R)) **then**
    while succ(p) $\neq$ INDEFINI **do**
      $p \leftarrow succ(p)$
    end while
    $queueRègle \leftarrow p$
  else
    $queueRègle \leftarrow \hat{t(e)}teRèegle(R)$
  end if

Lexique :

- R : La règle dont on souhaite obtenir la queue de la premisse

Algorithm 10: TestPremisseR

Data: R : [Règle], id : [chaine de caractère]
Result: B : Booléen
  if estVideRègle(R) OU id = INDEFINIT **then**
    $TestPremisseR \leftarrow FAUX$
  end if
  if estVidePrémisse(R) **then**
    $TestPremisseR \leftarrow FAUX$
  else
    if ID(têteRègle(R))=ID **then**
      $TestPremisseR \leftarrow VRAI$
    else
      $TestPremisseR \leftarrow TestPremisseR(Reste(R), id)$
    end if
  end if

Lexique :

- R : La règle où l'on souhaite chercher la proposition
- id : id de la proposition recherchée
- B : Un Booleen, Vrai si on a trouvé la premisse, faux sinon

3.2 Algorithmes et Bases de connaissances

Ici sont écrits les algorithmes en lien avec la base de connaissances du programme.

3.2.1 Constructeur

Algorithm 11: créerBCVide

Data:
Result: BC
$bc \leftarrow BCVide$
$créerBCVide \leftarrow bc$

Lexique :

- bc : Une base de connaissances vide

3.2.2 Modificateur

Algorithm 12: ajoutRègle

Data: bc : [BC], R : [Règle]
Result: [BC]
$pbc \leftarrow [Règle] : RègleVide$
if bc = INDEFINI **and** R $\neq$ INDEFINI **then**
  $bc \leftarrow R$
  $ajoutRègle \leftarrow bc$
else
  if bc $\neq$ INDEFINI **and** R $\neq$ INDEFINI **then**
    $pbc \leftarrow bc$
    while succ(pbc) $\neq$ INDEFINI **do**
      $pbc \leftarrow succ(pbc)$
    end while
    $succ(pbc) \leftarrow R$
    $succ(succ(pbc)) \leftarrow INDEFINI$
  end if
end if
$ajoutRègle \leftarrow bc$

Lexique :

- bc : La base de connaissances où l'on souhaite ajouter une règle
- R : La règle que l'on souhaite ajouter
- pbc : Règle vide qui prendra succesivement la valeur des élements de bc pour la parcourir

3.2.3 Observateur

Algorithm 13: têteBC

Data: bc : [BC]
Result: R : [Règle]
  if bc ≠ INDEFINI **then**
    $têteBC \leftarrow bc$
  else
    $têteBC \leftarrow INDEFINI$
  end if

Lexique :

- bc : La base de connaissances dont on souhaite obtenir la tête
- têteBC : La tête de BC

3.3 Algorithmes et Bases de faits

3.3.1 Constructeur

Algorithm 14: créerBFVide

Data:
Result: BF
  $bf \leftarrow BFVide$
  $créerBFVide \leftarrow bf$

Lexique :

- bf : Une base de connaissance vide

3.3.2 Modificateur

Algorithm 15: ajouterFait

Data: bf : [BF], P : [Proposition]
Result: BF
  if bf = INDEFINI **then**
    $bf \leftarrow P$
  else
    $pbf \leftarrow [Proposition] : bf$
    while succ(pbf) ≠ INDEFINI **do**
      $pbf \leftarrow succ(pbf)$
    end while
    $succ(pbf) \leftarrow P$
    $succ(succ(pbf)) \leftarrow INDEFINI$
  end if
  $ajouterFait \leftarrow bf$

Lexique :

- bf : La base de connaissances où l'on souhaite ajouter une proposition
- P : La proposition que l'on souhaite ajouter

3.4 Algorithmes et Moteur d'inférence

Voici les algoritmes nécessaires au fonctionnement du moteur d'inférence.

Algorithm 16: calculMoteur

```
Data: bf : [BF], bc : [BC]
Result: BF
  if bf ≠ INDEFINI and bc ≠ INDEFINI then
    pbf ← [Proposition] : bf
    while pbf ≠ INDEFINI do
      pbc ← [Régle] : bc
      while pbc ≠ INDEFINI do
        if id(pbf) ≠ id(queueRègle(pbc)) then
          if TestPremisseR(pbc,id(pbf)) = Vrai then
            suppProposition(pbc,ID(PBF))
            if estVidePremisse(pbc) then
              ajoutFait(bf,queueRègle(pbc))
            end if
          end if
        end if
        pbc ← succ(pbc)
      end while
      pbf ← succ(pbf)
    end while
  end if
  calculMoteur ← bf
```

Lexique :

- bf : La base de faits rentrée par l'utilisateur
- bc : La base de connaissances réunissant l'ensemble des règles du système
- pbf : Proposition, permettant le parcours de la base de faits
- pbc : Règle, permettant le parcours de la base de connaissances

3.5 Choix d'implémentation en langage C

Dans cette partie nous allons présenter et expliquer nos choix d'implémentations de nos algorithmes et de nos notions pour le langage C.

3.5.1 Propositions et Règles

Nous avons définit une proposition comme une structure de données composée de deux élements :
Une chaîne de caractères qui sera son identifiant et un booléen qui sera sa valeur (Vrai ou Faux).
On définit alors une liste de propositions comme une liste simplement chaînée, on appellera elementRègle un de ces élements. Ce dernier est composé d'un pointeur sur une Proposition et un pointeur sur l'élement suivant.

Une Règle est définie comme une structure de deux élement : un pointeur sur un élementRègle "tête" et un pointeur sur Proposition appelée "Conclusion".
Grâce à cette implémentation , nous avons un accès rapide à la conclusion tout en simplifiant
le type règle : La conclusion n'est plus en fin de prémisse, cela simplifie donc la manipulation et la conception de la prémisse.

3.5.2 Base de connaissances

On défini une base de connaissances comme une liste chaînée d'élements appelée elementBC, un elementBC est une structure composée d'un pointeur sur une règle et d'un pointeur sur un autre elementBC : le suivant.

3.5.3 Base de faits

On défini une base de faits comme une liste chaînée d'élements appelée elementBF, un elementBF est une structure composée d'un pointeur sur une proposition toujours vraie et d'un pointeur sur un autre elementBF : le suivant.

3.5.4 Moteur d'inférence

Le moteur d'inférence est implémenté en temps que fonction à deux paramètres : bc, un pointeur sur une structure BC(base de connaissances) ; bf, un pointeur sur une structure BF(base de faits). Cette fonction renvoie un pointeur sur la nouvelle base faits.
En vue de l'implémentation des propositions, le moteur d'inférence n'a pas besoin de parcourir la base de fait ou de supprimer des eléments de la prémisse d'une règle pour fonctioner, il se contente simplement de lire la valeur de la proposition afin de determiner si elle est vraie ou fausse.
On peut donc lancer le moteur d'inférence plusieurs fois, en rajoutant ou modifiant des règles dans la base de connaissances entre chaque essai sans problème.

4 Jeux d'essais

Dance cette partie nous allons voir le comportement de notre implémentation d'un systeme expert en C. Nous allons voir comment l'implémentation réagit à différentes règles, prémisses et bases de faits.

4.1 Essai n°1 : 1 règle

Pour cet essai la base de connaissances ne contient qu'une seule règle R.

$$\begin{aligned} R : (E)^2 &\longrightarrow E \\ (A, B) &\longmapsto C \end{aligned}$$

(Voir fichier src/Jeux d'essai/test1.c)
Observons le résultat du moteur d'inférence pour différentes bases de faits :

```
❯ ./a.out
A is true
B is true
C is true
```

Figure 1: Ici, en entrée seulement A et B sont vraies.

```
❯ ./a.out
A is true
```

Figure 2: Ici, en entrée seulement A est vrai.

On peut voir que le moteur d'inférence a le comportement attendu avec une règle :
Si tous les élements de la premisse sont dans la base de faits, alors la conclusion est vraie et rentre donc à son tour dans la base de faits, si ce n'est pas le cas, alors rien ne se passe.

4.2 Essai n°2 : 3 règles

Pour cette essai la base de connaissances contient trois règles : R1, R2 et R3.

$$\begin{aligned} R1 : (E)^2 &\longrightarrow E \\ (A, B) &\longmapsto C \end{aligned}$$

$$\begin{aligned} R2 : (E)^2 &\longrightarrow E \\ (C, D) &\longmapsto F \end{aligned}$$

$$\begin{aligned} R1 : (E)^2 &\longrightarrow E \\ (B, C) &\longmapsto D \end{aligned}$$

(Voir fichier src/Jeux d'essai/test2.c)

Observons le résultat du moteur d'inférence pour différentes bases de faits :

Figure 3: Ici, en entrée seulement A et B sont vraies.

Figure 4: Ici, en entrée seulement B et C sont vraies.

Figure 5: Ici, en entrée seulement A, C et D sont vraies.

On observe que le moteur d'inférence fonctionne aussi avec plusieurs règles, quelque soit la base de faits, la réponse du moteur d'inférence est toujours celle attendu.

5 Conclusion

Nous avons atteints l'objectif qui nous étais fixés : L'implémentation de notre système expert fonctionne en C sans soucis apparents. Toutefois, l'linterface pourrait être grandement améliorée : En effet, il n'y a pas moyen de revnir en arrière dans le menu, on peut donc se retrouver bloquer dans certains menu et contrains à relancer le programme. De plus, il est fort probable que l'interface aurait pu être simplifiée, de manière à réduire le code dans le main.
Même si nous n'avons pas découvert de nouveaux aspects du langage C avec ce projet, ce dernier à permis de renforcer nos bases et d'utiliser les compétences acquises le semestre dernier : L'utilisation d'objets et de pointeurs force à la rigeur d'écriture, les fautes de segmentations ou la perte d'une variable par sortie d'un contexte sont des erreurs simples à commettre.

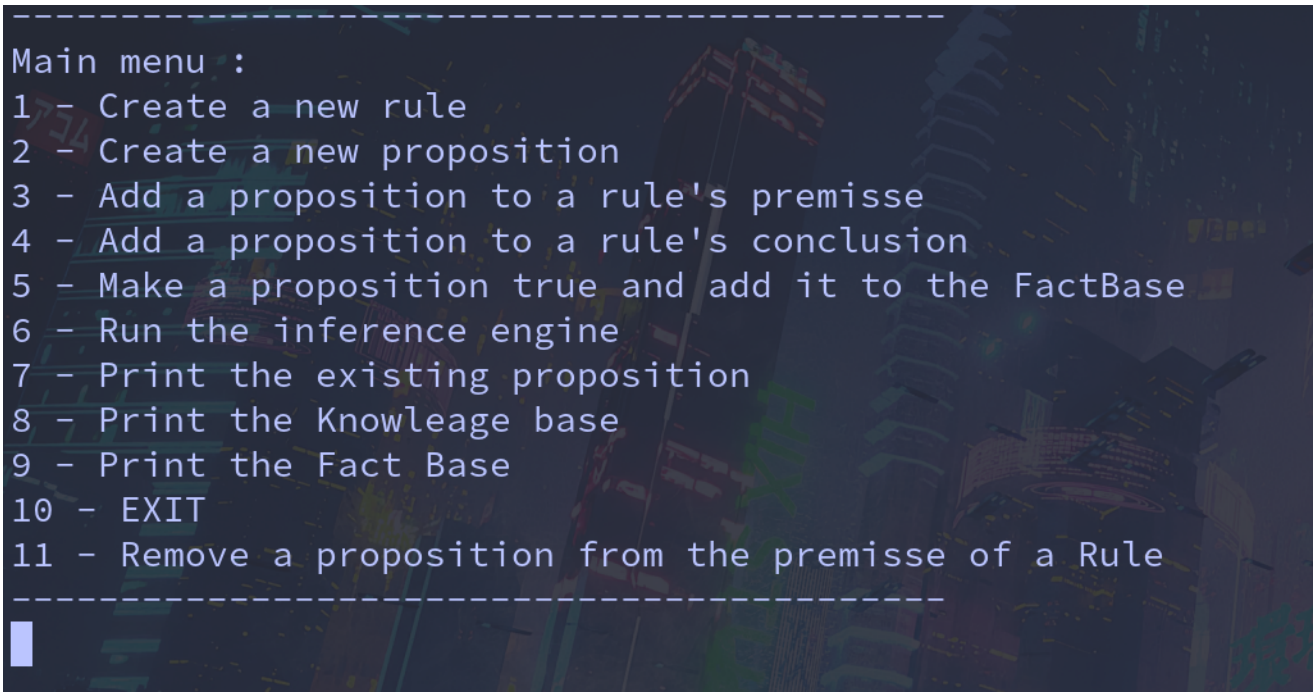

Figure 6: Menu principal du programme.