PROJET LO21
SYSTEME EXPERT

KACIMI Abdelhamid

MONTANGE Aimeric

TRONC COMMUN – 3e semestre

Notre projet était de concevoir un système expert. C'est-à-dire que l'utilisateur devait entrer des faits, dans notre cas des lettres (propositions) et à partir de ses faits nous devions en déduire d'autres faits grâce à des règles décrites dans la base de connaissances. Tout ceci fonctionne grâce au moteur d'inférence qui permet d'exécuter le schéma décrit ci-dessus.

Les règles sont des suites de proposition logique pouvant prendre la valeur « vrai » ou « faux ».

Afin de bien concevoir notre système, nous avons modélisé la base de connaissances et la base de faits grâce à des structures et des listes.

Tout d'abord nous avons choisi d'utiliser des listes pour pouvoir modéliser la base de connaissances. La base de connaissances est une liste de règles.

Par la suite nous avons défini la règle comme une structure simple composée d'une liste appelée « prémisse » contenant une proposition (lettre) et un booléen utilisé par la suite, et composée d'un proposition (lettre) pour la conclusion.

Pour la base de fait, nous l'avons modélisée par une simple liste contenant des faits (lettre).

A l'exécution de notre programme, l'utilisateur devra entrer les faits afin de construire sa base de fait. Par la suite le programme lui demandera de construire les règles de la base de connaissances, donc de construire les prémisses et les conclusions de chacune.

Une fois ceci fait, le moteur d'inférence travaillera et à partir des règles inscrites en déduira des faits qu'il ajoutera à la base de fait.

Ce projet nous a permis de travailler en équipe de deux sur un sujet précis. Notre répartition fut très bien construite afin que le projet soit mené à bien. Nous avons tous les deux appris beaucoup lors de la conception de ce projet, ce qui a enrichi notre savoir dans le domaine de la programmation et de la documentation en langage C.

Algorithme

element : STRUCT

CARACTERE lettre

ENTIER present

elem* suivant

FIN STRUCT

element*⇔liste

regle : STRUCT

liste premisse

CARACTERE conclusion

FIN STRUCT

ELEMENT : STRUCT

regle reglebc

ELEM* suivant

FIN STRUCT

ELEMENT*⇔basec

El : STRUCT

CARACTERE fait

E* suivant

FIN STRUCT

El*⇔basef

Lexique : R : règle à créer	Créer_regle regle R R.premisse←INDEFINI R.conclusion←INDEFINI RETOURNER R
Données : - liste pre - Caractère conclu	
Résultat : la regle R	

Lexique : temp : liste permettant de créer l'élément de la liste contenant le caractère voulu i : liste identique à la prémisse, permet de se placer à la fin de la prémisse Is_present : fonction permettant de savoir si le caractère est dans la liste	Ajouterq_premisse temp←Allocation(element*) temp->lettre←c temp->suivant←INDEFINI temp->present←FAUX SI (Is_present(c,pre)=VRAI) ALORS ECRIRE (Erreur : cette proposition est déjà presente dans la premisse) RETOURNER pre FIN SI SI (pre=INDEFINI) ALORS RETOURNER temp SINON i←pre TANT QUE (i->suivant≠INDEFINI) i←i->suivant FIN TANT QUE i->suivant←temp RETOURNER pre FIN SI
Données : - CARACTERE c - liste pre	
Résultat : liste temp ou liste pre	

Lexique : temp : liste permettant de stocker la prémisse de la règle à laquelle nous voulons ajouter une conclusion Is_present : fonction permettant de savoir si le caractère est dans la liste	Creer_conclusion temp←Allocation(element*) temp←R.premisse SI (Is_present(conclu,temp)=VRAI ECRIRE(Erreur : c'est une des prémisses) RETOURNER R.conclusion FIN SI SI (R.conclusion=INDEFINI) ALORS RETOURNER conclu SINON ECRIRE (Erreur : la règle a déjà une conclusion) RETOURNER R.conclusion FIN SI
Donnees : - Regle R - CARACTERE conclu	
Resultat : CARACTERE	

Donnees : - CARACTERE c - liste a_tester	Is_present SI(a_tester=INDEFINI) ALORS RETOURNER FAUX SINON SI (a_tester->lettre=c) ALORS RETOURNER VRAI SINON RETOURNER Is_present(c,a_tester->suivant) FIN SI FIN SI
Resultat : Boolean	

Lexique : c : proposition à supprimer R : Regle dans laquelle la proposition est à supprimer temp : liste permettant d'accéder au caractère de la prémisse à supprimer l : liste permettant de stocker l'élément à supprimer et de libérer la place mémoire correspondante	Remove SI (Is_present(c,R.premisse) = FAUX) ALORS ECRIRE (Cette proposition n'est pas dans cette regle) SINON temp ← Allocation(element*) temp ← R.premisse TANT QUE (c ≠ temp->suivant->lettre) FAIRE temp ←temp->suivant FIN TANT QUE l←temp->suivant temp->suivant←temp->suivant->suivant LIBERER(l) FIN SI RETOURNER R
Donnees : - regle R - CARACTERE c	
Resultat : regle R sans le caractere c	

Lexique : R : règle vide ou non vide	Is_empty (regle R) SI (R.premisse=NULL) ALORS ECRIRE (vide) RETOURNER VRAI SINON ECRIRE (remplie) RETOURNER FAUX FIN SI
Données : regle R	
Résultat : Boolean : VRAI (si la regle est vide) ou FAUX (si la regle n'est pas vide)	

Lexique : R : Règle à partir de laquelle la prémisse est retournée	Head_Premisse RETOURNER R.premisse->lettre
Données : règle R	
Résultat : Première proposition de la prémisse de la règle R	

Lexique : R : Règle à partir de laquelle la conclusion est retournée	Conclusion RETOURNER R.conclusion
Données : règle R	
Résultat : Conclusion de la règle R	

Lexique : B : base de connaissance crée R : règle vide présente dans la base	Create_basec B←Allocation(ELEMENT*) R←Creer_regle_vide() B->reglebc←R B->suivant←NULL RETOURNER B
Données : Aucune	
Résultat : basec B	

Lexique : R : règle à ajouter dans la base de connaissance BC : base de connaissance dans laquelle est ajoutée la règle i : pointeur permettant de parcourir BC temp : base de connaissance temporaire dans laquelle la règle est ajoutée	ajouter_reglebc temp ← Allocation(ELEMENT*) temp->reglebc ← R temp->suivant ← NULL SI (BC=NULL) ALORS RETOURNER temp SINON I←BC TANT QUE (i->suivant ≠ NULL) FAIRE I←i->suivant FIN TANT QUE i->suivant←temp RETOURNER BC FIN SI
Données : - basec BC - règle R	
Résultat : Basec BC	

Lexique : B : base depuis laquelle la règle est retrouvée	Head_basec RETOURNER B->reglebc
Données : basec B	
Résultat : Première règle de la base de connaissance	

JEUX D'ESSAIS :

- Exécuter le programme

- Entrer dans la base de fait : {A-B-F}

- Entrer dans la base de connaissance : Les règles suivantes

  - A+B+C+D=E
  - B+F=K
  - A+B+K=H
  - B+X=Y

- Afficher la base de fait

Résultat attendu dans l'affichage de la base de fait : {A-B-F-K-H}

- Ajouter une règle : A+B+Z=P
- Afficher les règles :

Résultat attendu :

A+B+C+D=E

B+F=K

A+B+K=H

B+X=Y

A+B+Z=P

- Ajouter un fait à la base de fait : ajouter {Z}

- Afficher la base de fait

Résultat attendu dans l'affichage de la base de fait : {A-B-F-K-H-Z-P}

Commentaires :

Lorsque nous entrons dans la base de faits, chacun des éléments {A-B-F} sont ajoutés en queue.

Ensuite à la création des règles, le moteur d'inférence étudie les prémisses des règles afin de savoir si des éléments de celles-ci sont dans la base de faits, si tous les éléments de la prémisse sont présents dans la base de faits alors la conclusion est ajoutée dans la base de faits, et les règles sont parcourues avec la nouvelle base de faits, avec le nouvel élément ajouté.

C'est pourquoi la règle numéro 2 permet d'ajouter {K} à la base de faits, ensuite la règle numéro 3 permet d'ajouter {H} à la base de faits.

La base de faits est alors constituée de 5 éléments : {A-B-F-K-H}

L'ajout d'une règle permet de compléter la base de connaissances avec une nouvelle règle qui sera ajouté en queue de la base de connaissances.

Par la suite, l'ajout à la base de faits de l'élément {Z} permet grâce à la règle numéro 5 d'ajouter l'élément {P} à la base de faits.

La base de faits est alors constituée de 7 éléments : {A-B-F-K-H-Z-P}