Système expert

Caillier Paul

Trombini Quentin

SOMMAIRE :

Sommaire…………………………………………………………………………………………………………………………………….page 1.

Choix de conception…………………………………………………………………………………………………………………...page 2.

Algorithmes…………………………………………………………………………………………………………………………………page 3.

Jeux d’essais………………………………………………………………………………………………………………………………page 18.

Commentaires ………………………………………………………………………………………………………………………….page 19.

Choix de conception

Nous avons choisi de représenter notre base de connaissance par le produit cartésien d’une règle et de son suivant, les règles sont-elles mêmes les produits cartésiens d’une prémisse et d’une conclusion : la conclusion est une chaine de caractère et n’est vrai que lorsque la prémisse est vraie, une prémisse est le produit cartésien d’une proposition et de son suivant. Lorsque toutes les propositions de la prémisse d’une règle sont vraies alors sa conclusion l’est aussi. La base de faits quant à elle est constitué uniquement d’une prémisse car c’est une liste de proposition qui sont toutes vraies alors autant les stocker toutes dans la même liste. Avec ce système nous n’avons besoin que de la tête de liste de la base de connaissance et de la tête de liste de la base de faits pour pouvoir effectuer toutes les actions demandées : cela nous éviter de faire passer plus de variable a travers tout le code.

Nous avons ensuite séparé le programme en trois partis distincts :

* Le menu principal permettant d’accéder aux deux autres menus et de lancer le moteur d’inférence.
* Le menu BC qui permet d’appeler toutes les actions utiles avec la base de connaissance, les règles et leurs conclusions.
* Le menu BF qui permet d’appeler toutes les actions utiles avec la base de fait et son affichage.

Algorithmes

Proposition -> Liste de caractère

Prémisse -> produit cartésien d’une proposition et de son suivant

Règle -> Produit cartésien d'une Prémisse et d'une Proposition nommée conclusion

BaseDeConnaissance -> produit cartésien d’une règle et de son suivant

BaseDeFaits -> produit cartésien d’une proposition et de son suivant

**But de la fonction** : créer une règle vide

**Lexique :** Règle -> Produit cartésien d'une Prémisse et d'une Proposition nommée conclusion

**Algorithme :** *Données* : Ø ; *Sortie :* Règle

Fonction createRule(): Règle

DEBUT:

createRule <- nouveau(Règle)

FIN.

**But de la fonction** : Ajouter une proposition à la queue d’une prémisse

**Lexique :** Prémisse -> produit cartésien d’une proposition et de son suivant, Proposition-> Liste de caractère

**Algorithme :** *Données* : prem(une Prémisse), str(une proposition) ; *Sortie :* prem une prémisse

Fonction addProp(prem:<Prémisse>,str:<Proposition>)

DEBUT:

newElem=nouveau(Premisse)

newElem(proposition)=str

newElem(suivant)=NULL

SI estNULL(prem) ALORS

Prem=newelement

SINON

Prémisse buffer=prem

TANT QUE estNonNULL(suivant(buffer))ALORS

Buffer=suivant(buffer)

FIN TANT QUE

Suivant(buffer)newElem

FIN SINON

addProp()<-prem

FIN.

**But de la fonction** : créer la conclusion d’une règle

**Lexique :** Règle -> Produit cartésien d'une Prémisse et d'une Proposition nommée conclusion, conclusion -> Liste de caractère

**Algorithme :** *Données* : rule: (une Règle) ; *Sortie :* rule: (une Règle)

Fonction createConclusion(rule:<Règle>)

Début:

Conclusion(rule) <- nouveau(Proposition)

createConclusion<- rule

Fin.

**But de la fonction** : tester si une proposition appartient à la prémisse d’une règle(récursif)

**Lexique :** Prémisse -> produit cartésien d’une proposition et de son suivant, Proposition -> Liste de caractère

**Algorithme :** *Données* : rulePrem(une prémisse),prop(une proposition) ; *Sortie :* un booléen

Fonction testProp(rulePrem:<Premisse>, prop:<Proposition>):Booléen

DEBUT:

SI estVide(rulePrem) ALORS

testProp <- faux

SINON:

SI rulePrem = prop:ALORS

testProp <- vrai

SINON:

testProp <- testProp(suivant(rulePrem),prop)

FIN SI.

FIN SI.

FIN.

**But de la fonction** : Supprimer une proposition de la prémisse d’une régle

**Lexique :** Prémisse -> produit cartésien d’une proposition et de son suivant, Proposition-> Liste de caractère, Règle -> Produit cartésien d'une Prémisse et d'une Proposition nommée conclusion

**Algorithme :** *Données* : rulePrem(une Prémisse), prop(une proposition) ; *Sortie :* prem une prémisse

Fonction deleteProp(rulePrem:<Premisse>,prop:<Proposition>)<Premisse>

DEBUT

prem<Premisse>

teteListe<Premisse>

teteListe=rulePrem

SI estVide(rulePrem) ALORS

deleteProp<-NULL

SINON

SI proposition(rulePrem)=prop ALORS

prem=suivant(ruleprem)

free(ruleprem)

deleteprop<-prem

FIN SI

FIN SI

SI suivant(rulePrem)=prop ALORS

prem=suivant(ruleprem)

suivant(ruleprem)=suivant(suivant(ruleprem))

free(p)

deleteProp<-save

SINON

ruleprem=suivant(ruleprem)

FIN SI

TANT QUE !estVide(rulePrem) ALORS

SI suivant(rulePrem)=prop ALORS

p=suivant(ruleprem)

suivant(ruleprem)=suivant(suivant(ruleprem))

free(p)

deleteProp<-save

SINON

ruleprem=suivant(ruleprem)

FIN SI

FIN TANT QUE

FIN

**But de la fonction** : tester si la prémisse d’une règle est vide

**Lexique**Règle -> Produit cartésien d'une Prémisse et d'une Proposition nommée conclusion, Prémisse -> produit cartésien d’une proposition et de son suivant

**Algorithme :** *Données* : rule(une Régle) ; *Sortie :* un booléen

Fonction isPremisseEmpty(rule:<Règle>): Booléen

DEBUT:

SI (premisse(rule)=NULL ALORS

isPremisseEmpty <- vrai

SINON:

isPremisseEmpty <- faux

FIN SI.

FIN.

**But de la fonction** : supprimer une proposition précise dans une prémisse

**Lexique**Prémisse -> produit cartésien d’une proposition et de son suivant, Proposition -> Liste de caractère

**Algorithme :** *Données* : head(une prémisse),prop(une proposition) ; *Sortie :* une prémisse

Fonction deleteProp(head:<Prémisse>,prop :<proposition>):<Prémisse>

DEBUT

<Prémisse> :p

<Prémisse> : save

SI estNULL(head) ALORS

deleteProp<-NULL

SINON

SI proposition(head)=prop ALORS

p=suivant(head)

Free(head)

deleteProp<-p

FINSI

FINSI

FAIRE

SI proposition(suivant(head))=prop ALORS

p=suivant(head)

suivant(head)=suivant(suivant(head)

free(p)

deleteprop<-save

SINON

Head=suivant(head)

FINSI

TANT QUE estNonNULL(head)

deleteProp<-save

FIN

**But de la fonction** : accéder a la proposition se trouvant en tête d’une prémisse

**Lexique**Règle -> Produit cartésien d'une Prémisse et d'une Proposition nommée conclusion, Prémisse -> produit cartésien d’une proposition et de son suivant

**Algorithme :** *Données* : rule(une Régle) ; *Sortie :* une proposition

Fonction premisseHead(rule:<Règle>):proposition

DEBUT

premisseHead <-proposition(premisse(rule))

FIN

**But de la fonction** : accéder à la conclusion d’une regle

**Lexique**Règle -> Produit cartésien d'une Prémisse et d'une Proposition nommée conclusion, Proposition -> Liste de caractère

**Algorithme :** *Données* : rule(une Régle) ; *Sortie :* une proposition

Fonction RuleConclusion(rule:<Règle>):Proposition

DEBUT

RuleConclusion <-conclusion(rule)

FIN

**But de la fonction** : créer une base de connaissance vide

**Lexique**Règle -> Produit cartésien d'une Prémisse et d'une Proposition nommée conclusion ;BaseDeConnaissance -> produit cartésien d’une règle et de son suivant

**Algorithme :** *Données* : Ø ; *Sortie :* BC une BaseDeConnaissance

createBasis() :une BaseDeConnaissance

DEBUT

createBasis<- nouveau(BaseDeConnaissance)

FIN

**But de la fonction** : ajouter une règle à une base de connaissance

**Lexique**Règle -> Produit cartésien d'une Prémisse et d'une Proposition nommée conclusion, BaseDeConnaissance -> produit cartésien d’une règle et de son suivant

**Algorithme :** *Données* : BC (une BaseDeConnaissance) ; *Sortie :* BC une BaseDeConnaissance

addRule(BC :<BaseDeConnaissance>) :BaseDeConnaissance

DEBUT

<Régle> pass=régle(BC)

TANT QUE estNonNULL(suivant(pass)) ALORS

Pass=suivant(pass)

FIN TANT QUE

Suivant(pass)=nouveau(régle)

FIN

Ci-dessus les algorithmes des deux premières parties qui était demandé ci-dessous vous trouverez les algorithmes de la partie 3

**But de la fonction** : afficher la base de connaissance

**Lexique**Règle -> Produit cartésien d'une Prémisse et d'une Proposition nommée conclusion, Proposition -> Liste de caractère

**Algorithme :** *Données* : Bc(une BaseDeConnaissance) ; *Sortie :* Ø

Fonction afficheBC(BC :<BaseDeConnaisance>)

DEBUT

<Entier> :1

TANT QUE estNonNULL(BC) ALORS

Ecrire « ‘i’> »

<Prémisse> : prem=PremisseHead(régle(BC))

<Proposition> :conc=RuleConclusion(régle(BC))

TANT QUE estNonNULL(prem) ALORS

Ecrire «’proposition(prem)’ - »

FIN TANT QUE

Ecrire « >»

Ecrire «’conc’\n »

BC=suivant(BC)

i=i+1 ;

FIN TANT QUE

FIN

**But de la fonction** : ajouter et remplir une règle en queue de la base de connaissance

**Lexique**Règle -> Produit cartésien d'une Prémisse et d'une Proposition nommée conclusion, BaseDeConnaissance -> produit cartésien d’une règle et de son suivant.

**Algorithme :** *Données* : BC(une BaseDeConnaissance) ; *Sortie :* Ø

Fonction addRuleToBC(BC<BaseDeConnaissance>)

DEBUT

<régle> :rule=createRule()

SI estNULL(régle(BC)) ALORS

Régle(BC)=rule

Suivant(BC)=NULL

SINON

TANT QUE estNonNull(suivant(BC)) ALORS

BC=suivant(BC)

FIN TANT QUE

Newel=nouveau(BC)

Régle(newel)=rule

Suivant(BC)=newel

Suivant(newel)=NULL

FIN SINON

<Entier> :menu

<chaine de caractere> : strbuffer[255]

FAIRE

ECRIRE « Entrez une proposition»

Strbuffer=LIRE(input)

addProp(Prémisse(rule),strbuffer)

ECRIRE « Que souhaitez-vous faire avec cette nouvelle règle?\n\n »

ECRIRE « 1 – Entrez une nouvelle proposition \n\n»

ECRIRE « 2 – Entrez une conclusion et retourner au menu principal\n\n »

TANT QUE menu !=2

CreateConclusion(rule)

ECRIRE « Entrez une conclusion \n»

Strbuffer=LIRE(input)

Conclusion(rule)=strbuffer

FIN

**But de la fonction** : créer un menu pour accéder à toutes les fonctions utilisant la base de connaissance

**Lexique**BaseDeConnaissance -> produit cartésien d’une règle et de son suivant.

**Algorithme :** *Données* : BC(une BaseDeConnaissance),BF(une BaseDeFaits) ; *Sortie :* Ø

Fonction menuBC(BC <BaseDeConnaissance>, BF <BaseDeFaits>)

DEBUT

<Entier> :wait

<Entier> :quit=0

TANT QUE quit !=1 ALORS

ECRIRE « \n \n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n »

SI estNULL(régle(BC)) ALORS

ECRIRE « il n’y a aps de règles dans la base de connaissance\n »

SINON

ECRIRE « La base de connaissacne est \n»

Affiche(BC)

FIN SI

ECRIRE « Entrez le numéro correspondant à l’action que vous souhaitez effectuer »

ECRIRE « 1 - Ajouter une régle à la base de connaissance »

ECRIRE « 2 – Supprimer une régle »

ECRIRE « 3 – Quitter »

Wait=LIRE(input)

SI wait=1 ALORS

addRuleToBC(BC)

SINON

SI wait =2 ALORS

ECRIRE « Entrez le numéro de la règle à supprimer »

<Entier> : in=LIRE(input)

BC=deleteRule(BC,in)

SINON

SI wait=3 ALORS

Quit=1

FINSI

FINSI

FINSI

FIN TANT QUE

FIN

**But de la fonction** : ajouter une nouvelle proposition à la base de faits

**Lexique**BaseDeFaits -> produit cartésien d’une proposition et de son suivant

**Algorithme :** *Données* : BC(une BaseDeConnaissance),BF(une BaseDeFaits) ; *Sortie :* Ø

Fonction addBF(BC <BaseDeConnaissance>, BF <BaseDeFaits>)

DEBUT

<Entier> :menu

<chaine de caractere> : strbuffer[255]

FAIRE

ECRIRE « Entrez une proposition »

Strbuffer=LIRE(input)

addProp(BF,strbuffer)

ECRIRE  « QUE souhaitez vous faire »

ECRIRE «1 - Entrez une nouvelle proposition »

ECRIRE « 2 – retour au menu »

Menu=LIRE(input)

TANT QUE menu !=2

FIN

**But de la fonction** : afficher la base de faits

**Lexique**BaseDeConnaissance -> produit cartésien d’une règle et de son suivant, BaseDeFaits -> produit cartésien d’une proposition et de son suivant.

**Algorithme :** *Données* : BC(une BaseDeConnaissance),BF(une BaseDeFaits) ; *Sortie :* Ø

Fonction afficheBF(BC <BaseDeConnaissance>, BF <BaseDeFaits>)

DEBUT

SI estNonNULL(BF) ALORS

ECRIRE « La base de fait contient »

ECRIRE « ‘proposition(BF)’

TANT QUE estNonNULL(suivant(BF)) ALORS

BF=suivant(BF)

ECRIRE «  - »

ECRIRE « ‘proposition(BF)’ »

FIN TANT QUE

SINON

ECRIRE « La base de fait est vide »

FINSI

FIN

**But de la fonction** : créer un menu pour accéder à toutes les fonctions utilisant la base de faits

**Lexique**BaseDeFaits -> produit cartésien d’une proposition et de son suivant

**Algorithme :** *Données* : BC(une BaseDeConnaissance),BF(une BaseDeFaits) ; *Sortie :* Ø

Fonction menuBF(BC <BaseDeConnaissance>, BF <BaseDeFaits>)

DEBUT

<Entier> :quit=0

TANT QUE quit !=1 ALORS

ECRIRE « \n \n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n »

afficheBF(BC,BF)

<Entier> : wait

ECRIRE « Entrez le numéro correspondant à l’action que vous souhaitez effectuer »

ECRIRE « 1 - Ajouter une proposition à la base de faits »

ECRIRE « 2 – Supprimer une Proposition »

ECRIRE « 3 – Exécuter le système expert »

ECRIRE « 4 – Quitter »

Wait=LIRE(input)

SI wait=1 ALORS

addBF (BC)

SINON

SI wait =2 ALORS

ECRIRE « Entrez la proposition que vous souhaitez supprimer »

<chaine de caractere>:strbuffer[255]=LIRE(input)

BC=deleteProp(BF,strbuffer)

SINON

SI wait=3 ALORS

Moteur(BC,BF)

SINON

SI wait=4 ALORS

Quit=1

FINSI

FINSI

FINSI

FINSI

FIN TANT QUE

FIN

**But de la fonction** : lancer un moteur d’inférence afin de déduire de la base de faits ce qui est vrai

**Lexique**BaseDeFaits -> produit cartésien d’une proposition et de son suivant, BaseDeConnaissance -> produit cartésien d’une règle et de son suivant

**Algorithme :** *Données* : BC(une BaseDeConnaissance),BF(une BaseDeFaits) ; *Sortie :* Ø

Fonction moteur (BC <BaseDeConnaissance>, BF <BaseDeFaits>)

DEBUT

SI estNULL(proposition(BF)) ALORS

ECRIRE « La base de fait est vide »

SINON

SI estNULL(règle(BC)) ALORS

ECRIRE « La base de connaissance est vide »

SINON

<BaseDeConnaissances> : save=BC

<Entier> :buffer

<Prémisse> : end=BF

TANT QUE estNonNULL(suivant(end)) ALORS

End=suivant(end)

FIN TANT QUE

FAIRE

FAIRE

Buffer=testProp(prémisse(règle(BC)),proposition(BF))

SI buffer=1 ALORS

Prémisse(règle(BC))=deleteProp(prémisse(règle(BC)),proposition(BF))

SI estNULL(prémisse(règle(BC)) ALORS

<prémisse> : newel=nouveau(prémisse)

Proposition(newel)=conclusion(règle(BC))

Suivant(end)=newel

End=newel

ECRIRE « ‘proposition(end)’ »

FINSI

FINSI

BC=suivant(BC)

TANT QUE estNonNULL(BC)

BC=save

BF=suivant(BF)

TANT QUE estNonNULL(BF)

FINSI

FINSI

FIN

Objectif: Supprimer une règle dans une base de connaissance

Lexique:

BC: La base de connaissance

ToDelete: l'élément de la liste à supprimer

ToConnecter: l'élément de la liste à connecter au reste de la liste

i: le numéro de la règle a supprimer

j: une variable d'incrémentation

Fonction deleteRule(BC:<BaseDeConnaissance>,i:<Entier>):Entier

DEBUT

Si Règle(BC) = INDEFINI Alors: // Cas ou la BC est nulle

deleteRule <- -1

FinSi

j <- 1

toDelete = BC

toConnect = BC

Tant que j != i faire:

Si next(toDelete) = INDEFINI alors://Cas ou l'index i est trop grand

deleteRule <- -1

Sinon:

toConnect <- toDelete

toDelete <- next(toDelete)

FinSi

j <- j+1

FinTantQue

Si toDelete==toConnect alors: //cas ou le premier élément de la liste est suprimer

BC <- next(BC)

Sinon Si next(toDelete) != INDEFINI alors://Cas en millieu de liste

toConnect <- next(next(toConnect));

Sinon://Cas en fin de liste

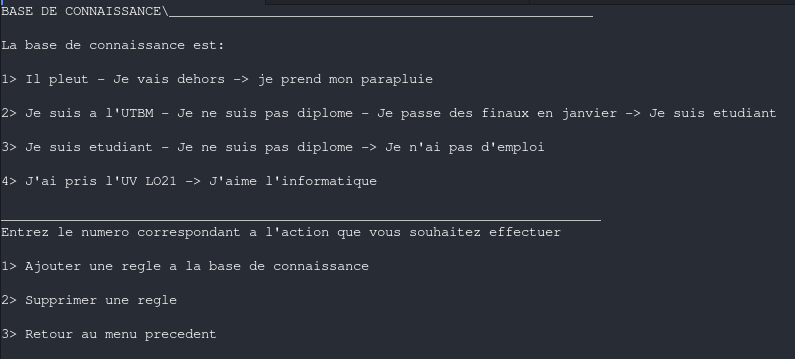
next(toConnect) <- INDEFINI

FinSi

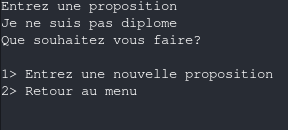
suppr(toDelete)

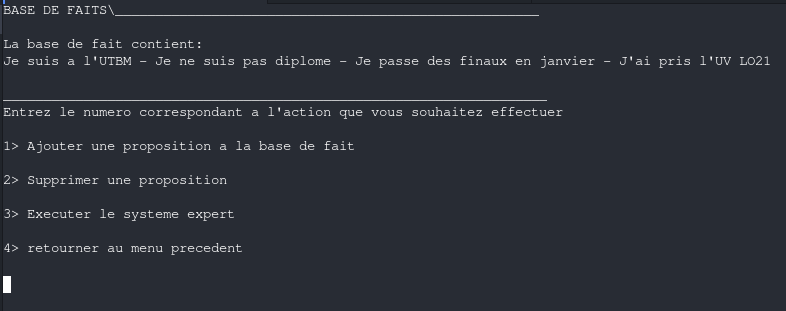
FIN

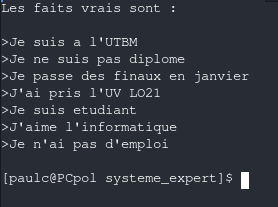
Jeux d’essais

Début du programme avec la base de connaissance pré-enregistrée dans un fichier .txt

Depuis le menu toute la base de connaissances peut être modifié

Ci-dessous le menu permettant d’ajouter des propositions à la base de fait

Le menu permettant d’effectuer toutes les actions sur la base de faits et de lancer le moteur d’inférence, ainsi que d’afficher la base de faits

 Le résultat du système expert qui à déduit de la base de fait et de la base de connaissance toutes les propositions qui sont vrai.

Commentaires

Comme le montre le graphique ci-dessous nous avons effectué le projet sur une durée d’environ un mois et grâce à un travail assez régulier nous avons réussir à finir le projet à temps.

Cependant nous aurions aimé y ajouter d’autre features comme par exemple le fait que certaine conclusion en interdise d’autre (exemple : si « il fait beau est vrai » alors « il pleut » est forcément faux). Mais malgré tout nous somme heureux d’avoir réussi à mener le projet à bien et d’avoir réussi à organiser correctement notre travail pour finir dans les temps.

Vous pourrez d’ailleurs accéder au GitHub en suivant le lien suivant : https://github.com/Clamorae/systeme\_expert