Talarmain Corentin
Voillemin Théo

Projet d'intelligence artificielle
Système expert

Présentation du projet :

Pour ce projet d'intelligence artificielle, il nous a été demandé de développer un système expert de niveau zéro plus ou supérieur ainsi que son moteur d'inférence.
Le moteur d'inférence développé devait gérer un chaînage avant et un chaînage arrière ainsi que de gérer une ou plusieurs incohérences présentent dans la base de faits.
C'est ce que vous retrouverez donc dans notre projet, le tout traiter sur un système expert de niveau un.

1) Contexte :

Nous avons développé notre système expert autour du contexte de la criminalité. Vous retrouverez lors de l'exécution de notre programme un ensemble de prédicat et de règle visant à déterminer des faits et des conclusions à partir de prémisse. C'est finalement ce contexte qui nous a poussé à développer un système expert de niveau un et non l'inverse.

2) Système expert :

Le système expert que nous avons développé est un système expert de niveau un. Nous l'avons limité par faute de temps donc il s'agit juste de l'implémentation de prédicat vérifiant la validité d'un ou plusieurs ensembles de variables, ainsi que de l'implémentation de règle consistant en la conjonction de prédicat en conditions, et l'instanciation d'ensemble de variables à un ou plusieurs prédicats en conclusion.

Il a été assez compliqué de développer un tel système en un temps assez limité notamment à cause de la gestion de la coordination des prédicats dans les différentes règles. En effet, à l'instanciation d'un prédicat, celui-ci est généraliste, c'est-à-dire que l'on connaît juste son arité et donc le nombre de variables qu'il doit vérifier. Cependant, dans une règle, une dépendance se forme entre les prédicats de conditions et de conclusions. Cette dépendance vient du fait qu'il faut prendre en compte que les variables sont liées entre les prédicats pour que la règle est du sens.

Regardons un exemple sur deux prédicats et une règle :

Prenons le prédicat P1 : X est plus petit que Y
le prédicat P2 : X est plus grand que Y

Jusqu'ici, nous pouvons remarquer que les deux prédicats sont indépendants, mais regardons la dépendance qui se forme lors de l'instanciation d'une règle :

Règle R1 : X est plus petit que Y (P1)
et Z est plus grand que Y (P2)
donc X est plus petit que Z (P1)

Ici on remarque qu'une dépendance s'est formé, il faut bien que les variables X,Y et Z soit considérées comme étant les mêmes variables entre les différents prédicats conditions et conclusions de la règle.
C'est cette dépendance qui complexifie énormément l'implémentation d'un système expert de niveau un ainsi que de son moteur d'inférence associé comparé à un système expert de niveau zéro ou zéro+

Ainsi, vous trouverez dans notre code, les classes Variable, Predicat et Regle qui représentent chacune les concepts du système expert de niveau un précédemment présentés.

2) Moteur d'inférence :

Le moteur d'inférence est essentiellement développé dans la classe Moteur, cette classe contient l'ensemble des prédicats, l'ensemble des règles ainsi que l'ensemble des conclusions que l'on souhaite atteindre. Regardons dès à présent comment fonctionnent nos différentes fonctions :

-Chaînage avant :

Il s'agit de la fonction ChainageAvant de la classe Moteur.

Cette fonction effectue un chaînage avant tant que les conclusions n'ont pas été toutes déterminées, ou bien lorsque plus aucune conclusion de règle n'a pu être développée. Dans le premier, le chaînage avant a permis la détermination des conclusions recherchées, dans le deuxième cas, la base de fait initial était trop vide et peu précise pour permettre la découverte des conclusions recherchées.

Le chaînage avant s'effectue de cette manière :

-À chaque itérations, on parcourt toutes les règles de la base de règles.

-Pour chaque règle, on exécute une fonction récursive (verifyCA dans la classe Regle) qui regarde toutes les possibilités de variables que les prédicats en condition ont en commun. Pour chacune de ces possibilités, on instancie aux prédicats conclusions un nouvel ensemble de variables correspondant. La récursivité vient du fait qu'un prédicat peut avoir plusieurs ensembles de variables le vérifiant ainsi que le fait que les différents prédicats en conditions des règles sont liés entre eux.

-La fonction verifyCA renvoie un vecteur contenant une paire de prédicat et un ensemble de variables nouvellement instancié à ce prédicat. La fonction ChainageAvant regarde ainsi, si parmi ce vecteur, il y a un prédicat et son ensemble de variable qui fasse partie des conclusions que l'on cherche, si c'est le cas, on l'enlève ainsi des conclusions.

-Chaînage arrière :

Il s'agit de la fonction ChainageArriere de la classe Moteur.

Dans le cas de cette fonction, et contrairement au chaînage avant, nous partons des conclusions. Ainsi, le but est de regarder uniquement les règles pouvant servir à la création des conclusions, mais aussi d'instancier de nouveaux prédicats uniquement pour certaines valeurs de variables concernant celles recherchées dans les conclusions. Si le chaînage avant développait toutes nouvelles instanciations possible pour toutes les variables. Le chaînage arrière cherche juste un chemin permettant le développement de la conclusion sans divaguer et instancier de nouvelles valeurs de vérités inutiles à un prédicat.

Le chaînage arrière s'effectue de cette manière :

-Pour chaque conclusion, on récupère le prédicat ainsi que l'ensemble des variables à lui vérifier nécessaire.

-On récupère toutes règles de la base de règles ayant pour conclusion ce prédicat.

-Tant qu'une de ces règles ne nous a pas permis de déterminer la conclusion, on exécute à chacune de ces règles la fonction CAregle. Cette fonction consiste en le développement de tous les prédicats de la règle. Pour chacun des prédicats, on appelle la fonction CApreg. Cette dernière vise à développer tous les prédicats. Soit il n'existe pas de règle dont les conclusions contiennent ce prédicat, dans ce cas on s'arrête, soit il en existe et on applique donc de nouveau la fonction CAregle à chacune de ces règles. Le but de cette exécution et de totalement dérouler les règles et les prédicats afin de se retrouver dans le cas où un prédicat n'est conclusion d'aucune règle. À partir de là, on remonte à la règle qui a développé ce prédicat à l'aide de CApreg et une fois que tous les prédicats de cette règle se retrouvent dans cette configuration, on lance la fonction verifyCA précédemment décrite, la seule subtilité consistant dans le fait que des variables des bases lui sont données, celles de la conclusion recherchée, afin de n'instancier que de nouvelles valeurs de vérités utiles à la détermination de la conclusion recherchée.

-Critère de choix :

Comme demandé, vous retrouverez la possibilité d'ordonnancer la base de règles selon plusieurs critères :

-En premier, vous retrouverez la fonction tri_nb_preds qui ordonnance la base de règles de telle sorte que les règles soient disposées de celle ayant le moins de prédicats conditions à vérifier, à celle qui en a le plus.

-En second, vous trouverez la fonction tri_comp qui, elle, ordonnance la base de règle de telle sorte qu'en premier l'on retrouve la règles dont l'ensemble des prédicats conditions à le moins de variables à vérifier, et en dernier celle qui en a le plus. Pour être plus clair, il s'agit de ranger les règles selon la somme des arités de ses prédicats conditions.

-Cohérence :

Vous retrouverez dans ce système expert la présence d'une incohérence qui vous sera signalée. Deux prédicats contradictoires sont présents :

X est coupable,

X ne peut pas être le coupable de cette affaire.

Ainsi si l'un est instancié alors que l'autre est vrai pour la même variable X, vous trouverez un message d'erreur vous indiquant l'incohérence présente.

3) Interface graphique:

Nous avons mis en place une interface graphique permettant d'utiliser notre moteur de niveau 1, celui-ci donne les outils nécessaire à la création de la base de fait et de buts nécessaire à l'exécution d'un chaînage avant ou un chaînage arrière. Pour cela il va falloir d'abord passer par les menus d'ajouts permettant d'ajouter soit un fait soit un but selon la case cocher dans celui-ci. Ensuite il va falloir attribuer des valeurs aux variables du prédicat sélectionné. Quand vous avez

terminé de créer votre base de fait et vos buts, vous pouvez allez dans la fenêtre de résolution et exécuter soit le chaînage avant, soit le chaînage arrière et de regarder le résultat.

Vous avez accès, à partir de la première fenêtre, à un menu d'aide affichant toutes les règles que nous avons crées pour vous permettre de tester notre système.

Conclusion :

Ce projet fut très intéressant, et nous aurions presque aimé avoir plus de temps pour compléter notre système expert.
Car, en effet, même s'il s'agit d'un niveau un, il est encore assez rudimentaire, voilà ce qu'il nous semblerait intéressant de continuer à développer pour l'améliorer :

-Rajouter des domaines pour les variables, pour l'instant il s'agit juste d'un mot, on peut ainsi se retrouver avec un objet comme étant la victime de l'affaire.

-Compléter les prédicats, comme rajouter les symboles « pour tout » et « il existe », ou bien la possibilité qu'un prédicat contienne lui-même un prédicat dans sa définition.

-Compléter la définition des règles pour permettre la disjonction ou la négation de prédicats, ainsi qu'aussi le rajout des symboles « pour tout » et « il existe ».

Enfin, ce projet a été assez difficile pour nous. Nous avons hésité plusieurs fois à redescendre à un système expert de niveau zéro+, notamment à cause de la difficulté de réfléchir les algorithmes et leurs récursivités de chaînage avant et chaînage arrière, mais nous avons préféré persévérer, et c'est pourquoi nous sommes tout de même assez fiers de vous présenter notre projet tel qu'il est en seulement trois semaines de travail.