Intelligence Artificielle

TP2 : Résolveur de problèmes

Base de connaissances –test-

Le problème des cruches d'eau est le suivant :
On vous donne deux cruches d'eau, l'une de 4 litres, l'autre de 3 litres. Aucune de ces deux carafes n'est graduée. On vous donne une pompe d'eau qui peut être utilisée pour remplir les cruches. Comment remplir la cruche de 4 litres d'exactement 2 litres d'eau ?

Afin de répondre à cette question, le problème est modélisé et représenté à l'aide d'une base de connaissances d'ordre 1.
R1: si cruchesAetB(?x, ?y) et ?x <4 alors cruchesAetB (4, ?y)
R2. Si cruchesAetB(?x, ?y) et ?y<3 alors cruchesAetB (?x, 3)
R3 Si cruchesAetB (?x, ?y) et ?x > 0 alors cruchesAetB (0, ?y)
R4. Si cruchesAetB (?x, ?y) et ?y > 0 alors cruchesAetB (?x, 0)
R5. Si cruchesAetB (?x, ?y) et ?x + ?y ≥ 4 et ?y > 0 alors cruchesAetB (4, ?y – (4 – ?x))
R6. Si cruchesAetB (?x, ?y) et ?x + ?y ≥ 3 et ?x > 0 alors cruchesAetB (?x – (3 – ?y), 3)
R7. Si cruchesAetB (?x, ?y) et ?x + ?y ≤ 4 et ?y > 0 alors cruchesAetB (?x + ?y, 0)
R8. Si cruchesAetB (?x, ?y) et ?x + ?y ≤ 3, ?x > 0 alors cruchesAetB (0, ?x + ?y)
R9. Si cruchesAetB (0, 2) alors cruchesAetB (2, 0)
R10. Si cruchesAetB (2, ?y) alors cruchesAetB (0, ?y)
Dans la base de faits au départ on a cruchesAetB(0,0)
But1: cruchesAetB(2,0)
But2 : cruchesAetB(2, ?n)

A noter que ce projet résolveur doit pouvoir fonctionner avec n'importe quelle base de connaissances en entrée.

Etapes du projet.

1) Stocker la base de connaissances décrivant le problème dans un fichier texte .txt
2) Préparer la structure de données permettant de représenter le problème : base de règles, base de faits…
3) Préparer les fonctions permettant de charger la base de connaissances du problème en mémoire dans votre structure de données.
4) Développer une fonction génèreOperateursApplicables qui prend en entrée un état du problème et produit la liste de tous les règles déclenchables. Vous pouvez utiliser l'implémentation de la fonction d'unification utilisée en fin de TP.
   => génèreOperateursApplicables(Probleme p, Etat étatCourant): liste ReglesInstanciees
   Exemple:
   génèreOperateursApplicables (pbCruches, cruchesAetB(0, 0)) donne {R1(x/0,y/0), R2(x/0,y/0)}

5) Implémenter 2 algorithmes :
   a. Implémentation de l'algorithme de recherche en profondeur limitée itérative
   b. Implémentation de l'algorithme A* avec les heuristiques spécifiques au problème :
   - if ?x=2 alors h=0, si ?x+?y <2 alors h=7, si ?y>2 alors h=3 sinon h=1
   - h = abs(?x - 2)

6) Proposer 2 interfaces qui doivent permettre de choisir la base de connaissances, de choisir l'état initial, l'état final, de choisir l'algorithme de résolution et de montrer les étapes de la résolution du problème puis le résultat final, permettre de sauvegarde de ces informations dans un fichier algosRecherche.txt. (Afficher les nœuds développés, le chemin emprunté)
   a) Une version en ligne de commande : elle doit être fonctionnelle sans nécessiter l'utilisation de librairies graphiques.
   b) Une interface graphique : elle doit être agréable et facile à utiliser permettant d'intégrer l'ensemble des fonctionnalités du projet.

7) Optionnel
    a. Montrer le fonctionnement du résolveur sur d'autres problèmes (taquin, missionnaires,etc.)
    b. proposer d'implémenter d'autres algorithmes
    c. Comparaison des algorithmes

Vous pouvez utiliser les fonctions d'unification suivantes à compléter par 1a classe Expression ou créer vos propres classes/méthodes.

```
def unifier_atom(expr1:Expression,expr2:Expression):
    if(expr2.isAtom()): expr1,expr2=expr2,expr1
    if(expr1==expr2): return {}
    if(expr2.isVariable()): expr1,expr2 = expr2,expr1
    if(expr1.isVariable()):
        if(expr1 in expr2): return None
        if (expr2.isAtom()): return
{expr1.expression[0]:expr2.expression[0]}
        return {expr1.expression[0]:expr2.expression.__str__()}
    return None

def unifier(terms1:Expression,terms2:Expression):
    if(terms1.isAtom() or terms2.isAtom()): return
unifier_atom(terms1,terms2)
    F1,T1=terms1.separate()#return 2 lists : [first elt] [..rest..]
    F2,T2=terms2.separate()
    Z1=unifier(F1,F2)
    if(Z1==None): return None
    T1.substitute(Z1)
    T2.substitute(Z1)
    Z2=unifier(T1,T2)
    if(Z2==None): return None
    Z2.update(Z1)
    return Z2
```

Remise des travaux

Un dossier compressé contenant

I. Un rapport CONCIS comprenant la description des principales fonctions utilisées, avec des noms significatifs, un graphe des appels de fonctions, + la trace des exécutions des différentes parties. Préciser la version de python ainsi que des librairies utilisées.
II. Les fichiers de code et d'exemples nécessaires à l'exécution de votre projet.
III. Une démo de l'exécution

Ce dossier compressé est à envoyer par mail à :

-Mme Jarraya jarraya.amina.fst@gmail.com
-et à Mme Chater meriemchater@yahoo.fr

Critères d'évaluation

- Progression des travaux en séance de TP
- Qualité du code (reutilisabilité du code, documentation du code, maintenabilité du code, code exécutable)
- Qualité de la réalisation (fonctionnalités mises en œuvre, entrées-sorties claires et montrant explicitement les différentes étapes de réalisation des algorithmes)
- Qualité du rapport