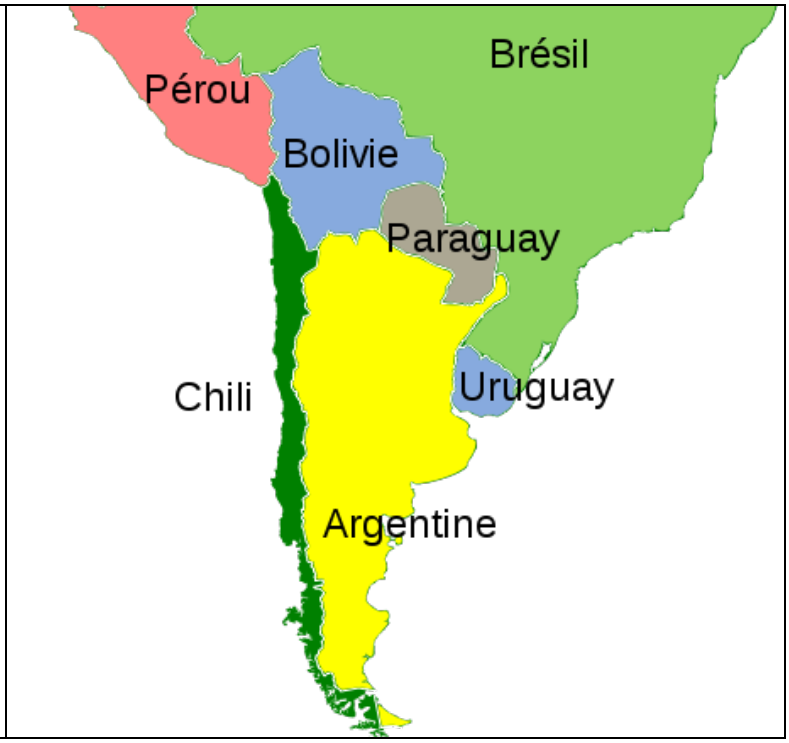
Université Montpellier II – Master Informatique 2015
Intelligence Artificielle (GMIN 101)

Examen

Durée : 2 heures
Seul document autorisé : une feuille A4
Barème indicatif : 6,5 à 7 points par exercice

Exercice 1. Modélisation en différents problèmes

On considère la carte ci-dessous (portion de l'Amérique du Sud) que l'on veut colorier avec trois couleurs de façon à ce que deux pays adjacents aient des couleurs différentes. Les couleurs sont notées 1, 2 et 3, les 7 pays de la façon suivante :

Pe : Pérou Bo : Bolivie Br : Brésil C : Chili Pa : Paraguay A : Argentine U : Uruguay	

1) Représentez ce problème sous la forme d'un problème de satisfaction de contraintes (X,D,C), où X est l'ensemble des variables, D l'union l'ensemble des domaines de chaque variable et C l'ensemble des contraintes. Vous indiquerez le nombre total de variables et de contraintes. Vous pouvez définir le contenu des contraintes en extension ou en intension (dans tous les cas, essayez d'être synthétique et clair dans votre présentation).

2) Représentez le même problème sous la forme d'un problème de calcul des réponses à une requête conjonctive Q dans une base de données B, où Q et B sont vus comme des ensembles d'atomes en logique du premier ordre, et chaque homomorphisme de Q dans B correspond à une solution. Pour décrire la représentation obtenue vous pouvez

soit indiquer comment vous passez du réseau de contraintes (X,D,C) à Q et B, soit partir à nouveau du problème de coloration initial.

3) Représentez le même problème sous la forme d'un problème SAT (où chaque modèle de l'ensemble de clauses construit correspond à une solution).

Exercice 2. Modélisation / Résolution de CSP

Un architecte pense pouvoir construire une maison en respectant les contraintes suivantes des différents prestataires :

1. La maçonnerie ne peut débuter qu'entre les jours 0 et 14 (c'est-à-dire y compris 0 et 14), et dure 7 jours
2. La charpente ne peut débuter qu'entre les jours 7 et 11, ne peut se faire qu'après la maçonnerie et dure 3 jours
3. Le toit ne peut débuter qu'entre les jours 12 et 14, ne peut se faire qu'après la charpente et dure 1 jour
4. La plomberie ne peut débuter qu'entre les jours 12 et 19, ne peut se faire qu'après la maçonnerie et dure 8 jours
5. Le sol ne peut débuter qu'entre les jours 11 et 17, ne peut se faire qu'après la maçonnerie et dure 3 jours
6. Le fenêtrage ne peut débuter qu'entre les jours 11 et 19, ne peut se faire qu'après le toit et dure 1 jour
7. La façade ne peut débuter qu'entre les jours 17 et 22, ne peut se faire qu'après le toit et la plomberie et dure 2 jours
8. Le jardin intérieur ne peut débuter qu'entre les jours 17 et 23, ne peut se faire qu'après le toit et la plomberie et dure 1 jour
9. La peinture ne peut débuter qu'entre les jours 8 et 18, ne peut se faire qu'après le sol et dure 2 jours
10. L'aménagement intérieur ne peut débuter qu'entre les jours 12 et 23, ne peut se faire qu'après le fenêtrage, la façade, le jardin et la peinture et dure 1 jour

a) Donnez une modélisation de ce problème sous forme de CSP.

b) Donnez le graphe (ou l'hypergraphe) des contraintes.

c) Dans l'objectif d'appliquer l'algorithme de backtrack, calculer l'ordre d'affectation des variables en appliquant l'heuristique qui consiste à privilégier la variable la plus contrainte (celle qui permet de vérifier à l'étape courante le plus de contraintes), et en cas d'égalité la variable la plus contraignante (celle qui partage le plus de contraintes avec les variables restant à assigner).

d) Appliquez l'algorithme de backtrack en utilisant cet ordre pour les variables et en privilégiant toujours la date au plus tôt.

e) Finalement donnez si cela est possible un planning pour la réalisation des différentes tâches.

Exercice 3. Règles positives en logique du premier ordre

On considère la base de connaissances suivante :

- Règles

  R1: flat(x1,y1) → sg(x1,y1)

  R2 : up(x2,y2) ∧ sg(y2,z2) ∧ up(t2,z2) → sg(x2,t2)

- Faits (où a,b,c,d,e,f,g sont des constantes)

  flat(a,b) flat(b,c) flat(a,c)

  up(d,a) up(d,b) up(e,c) up(f,d) up(g,e)

1) **Saturez** la base de faits avec les règles, en procédant **en largeur** (cf. algorithme FC rappelé en annexe). A chaque étape, on ne considère que les **nouveaux** homomorphismes. On rappelle qu'une application de règle est dite utile si elle produit un fait qui n'appartient pas à la base de faits courante.

Présentez les résultats dans un tableau selon le **format suivant** :

Etape	**Règle applicable**	**Homomorphisme**	**Fait produit**	**Application utile ?**
n° étape	n° règle	…	…	oui/non
…	…	…	…	…

2)

a. Comment reconnaît-on qu'un homomorphisme est *nouveau* ?

b. On dit qu'un prédicat est *calculé* s'il apparaît au moins une fois en conclusion de règle : ici, *sg* est un prédicat calculé, et c'est le seul. L'ensemble de règles ci-dessus a une particularité : la condition (l'hypothèse) de chaque règle contient au plus un prédicat calculé. Un tel ensemble de règles est appelé *linéaire*. Comment exploiter le fait qu'un ensemble de règles soit linéaire pour ne calculer que les homomorphismes nouveaux à chaque étape ?

3)

a. Soit la requête qui demande si « il existe x, y et z tels que sg(x,y) ∧ up(y,z) ∧ flat(z,c) » où *c* est une constante. La base de connaissances répond-elle positivement à cette requête ? Justifiez votre réponse en vous basant sur le mécanisme de *chaînage avant*.

b. Traitez cette requête par un mécanisme de *chaînage arrière*. Vous considérerez les faits avant les règles, et les règles dans l'ordre R1 puis R2. En ce qui concerne les atomes des buts, vous indiquerez quelles heuristiques vous font choisir un atome plutôt qu'un autre. Vous ferez la trace du chaînage arrière en indiquant les substitutions trouvées à chaque pas.

Annexe

```
Algorithme FC(K) // saturation de la base K
// Données : K = (BF, BR)
// Résultat : BF saturée par application des règles de BR
Début
fin ← faux
Tant que non fin
  new ← ∅ // ensemble des nouveaux faits obtenus à cette étape
  Pour toute règle R = H → C de BR
    Pour tout nouvel homomorphisme S de H dans BF
      Si S(C) n'est ni dans BF ni dans new
           Ajouter S(C) dans new
    FinPour
  FinPour
  Si new = ∅
    fin ← vrai
  Sinon ajouter tous les éléments de new à BF
FinTantQue
Fin
```