POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

Département de génie informatique et de génie logiciel

INF8215

TP2 : Prolog, programmation logique et programmation par contraintes

Rapport de laboratoire

Bilal Itani, [1743175]

Xiangyi Zhang, [1924288]

Mohammed Seddik Ben-Yahia, [1862313]

10 Novembre 2018.

Sommaire

Exercice 1 : Détective

Nous avons approché l'exercice 1 du laboratoire en identifiant le domaine de chaque variable dans notre problème. Pour modéliser la position d'une maison, nous avons pris un nombre entier entre 1 et le nombre total de maisons, soit 5. La valeur 1 représente la maison la plus à gauche et la valeur 5 représente la maison la plus à droite. Nous avons ensuite déterminé les différentes variables du problème. L'énoncé du problème énumère des variables sous cinq grande catégories, soit la nationalité, la boisson, la couleur de la maison, l'animal de compagnie du propriétaire et la profession de ce dernier. Le tableau 1 suivant énumère les différentes variables que nous avons considérées.

Tableau 1 : Énumération des variables de chacune des catégories

	Variable 1	Variable 2	Variable 3	Variable 4	Variable 5
Nationalité	Anglais	Espagnol	Ukrainien	Norvégien	Japonais
Boisson	Café	Thé	Lait	Jus Orange	Eau
Couleur	Rouge	Vert	Blanche	Jaune	Bleue
Animal	Chien	Escargots	Zèbre	Cheval	Renard
Profession	Sculpteur	Diplomate	Acrobate	Violoniste	Médecin

Le domaine de chacune de ces variables est entre 1 et 5, car une seule variable de chacune de ces catégories peut être associée à une maison. Ceci nous amène à une contrainte importante dans notre solution. Chaque variable d'une même catégorie est différente. Autrement dit, l'Anglais et le Japonais ne peuvent pas avoir la même valeur, puisqu'ils ne peuvent pas habiter dans la même maison. Le même raisonnement a été appliqué pour les autres catégories de variables. Les contraintes ont été écrites en suivant mot pour mot la liste des affirmations dans l'énoncé. La représentation de notre problème se trouve dans le fichier **q1.mzn**.

Avec cette approche, nous avons déterminé que **le norvégien** buvait de **l'eau** et que **le zèbre** était l'animal de compagnie de **l'ukrainien**.

Exercice 2 : Round-Robin

Modèle du CSP

Concernant le tournoi Round-Robin et afin de mettre en place un calendrier des matches de chacun des N-1 tours, nous avons utilisé un modèle basé sur un tableau (appelé **calendar**) à deux dimension, dont les lignes représentent les équipes et les colonnes représentent les tours. Nous avons pris comme variables : **nbTeams**, qui représente le nombre d'équipes, **nbRounds** qui représente le nombre de tours et **pv** qui est un tableau de dimension **nbTeams * nbTeams** et qui détermine le lieu des matches (à domicile ou à l'extérieur).

Par la suite, nous avons ajouté à ce modèle les différentes contraintes à considérer, et qui sont comme suit :

1. Toutes les équipes d'une ligne sont différentes, car chaque équipe joue contre toutes les autres une seule fois exactement.
2. Toutes les équipes d'une colonne sont différentes, car chaque tour est marqué par la confrontation de toutes les équipes.
3. Une équipe ne peut pas jouer contre elle-même et si une équipe *i* joue contre une équipe *j* dans le tour *k*, alors *j* joue contre *i* dans le même tour.
4. Une équipe ne peut jouer 4 matchs successifs (ou plus) à domicile ni 4 matchs successifs (ou plus) à l'extérieur.

La symétrie

La symétrie dans ce problème découle du fait que les tours sont indistincts.

Afin de briser la symétrie, il convient d'utiliser des contraintes redondantes pour filtrer les états équivalents de l'espace de recherche initial afin que l'algorithme de recherche fonctionne plus efficacement. Ainsi, nous avons introduit une contrainte redondante qui ajoutent le fait qu'une équipe *i* doit avoir une paire de tours adjacents *j* et *j+1* tels que le numéro de l'équipe à affronter dans le tour *j* est inférieur au numéro de l'équipe à affronter dans le tour *j+1*. Cette contrainte est évidemment redondante et fait que les tours ne soient plus totalement équivalents.

Temps de résolution

L'ajoute de la contrainte redondante a diminué le temps de résolution de **98%** environ, soit un passage de **2mn 18s** avant l'ajout de la contrainte redondante à **2s 222ms** après. L'efficacité de l'algorithme a été énormément augmentée. Une si grande différence peut être expliquée par le fait qu'avant l'ajoute de la contrainte redondante et à cause de la symétrie, il existait un certain nombre d'états équivalents lors de la recherche, ce qui a alourdi la charge de calcul de l'algorithme de recherche.

Bonus

La contrainte globale

À afin de faire respecter l'exigence du nombre de matchs à domicile/à l'extérieur, nous avons utilisé la contrainte globale "**at_least**". Cette dernière, prend trois arguments et se lit comme suit : *au moins* n *variables* dans un *tableau* x prennent une *valeur* v. Pour notre cas spécifique, nous avons ajouté une contrainte globale qui exige pour chaque équipe d'avoir un match à domicile (au moins) et un match à l'extérieur (au moins) dans quatre tours successifs.

La raison de son utilisation

L'avantage d'une contrainte globale est sa capacité de prendre en compte un ensemble de contraintes d'un point de vue global et de les propager plus efficacement. Ce qui augmente considérablement l'efficacité du CSP.

Exercice 3 : Cours à prendre

Afin de résoudre l'exercice 3, nous avons d'abord identifié les différentes relations qu'un cours peut avoir avec un autre dans notre base de connaissance. Il existe ici deux relations qu'il faut représenter. La relation **requis**, soit une relation où un cours B ne peut pas être pris que si on a suivi un cours A dans le passé ou que ce cours A est suivi en même temps que B. Ensuite, il y a la relation de **corequis**, soit une relation symétrique, selon l'énoncé du problème, où un cours B ne peut être suivi à une même session que si un cours A est aussi suivi à la même session. Ici, la relation **corequis** est comprise dans la relation **requis**. Avec le schéma fourni dans l'exercice 3, nous avons réussi identifié les faits dans l'exercice pour peupler notre base de connaissance. Ensuite, nous avons dû représenter six prédicats, trois prédicats pour la relation **corequis** et trois autres pour la relation **requis**. Dans notre solution, nous faisons la distinction entre un fait et une relation. En effet, tout prédicat commençant par "f_" représente un fait. Ici, la relation de prérequis est représenté uniquement au niveau de la base de connaissance, puisque nous avons jugé redondant de représenté la relation de prérequis dans notre solution.

Relation corequis

1) f_corequisite(B,A) → corequisite(A,B)
2) f_corequisite(A, X), corequisite(X,B) → corequisite(A,B)
3) requisite(A,X), corequisite(X,B) → corequisite(A,B)

La première relation est la relation de symétrie pour les corequis. Si un cours B est corequis à un cours A dans la base de connaissance, alors A est aussi corequis à B. La deuxième relation est une relation de transitivité. Elle est aussi récursive. Si un cours A est corequis à un autre cours et que cet autre cours est corequis à B. Alors A est corequis à B. Finalement, la troisième relation est aussi une relation de transitivité, elle est aussi récursive. Elle tient en compte la notion de requis. Si un cours A est requis à un autre cours et que cet autre cours est corequis à B. Alors A est aussi vue comme étant corequis à B.

Relation requis

4) f_prerequisite(A,B) ; f_corequisite(A,B) → requisite(A,B)
5) f_prerequisite(A, X), prerequisite(X,B) → requisite(A,B)
6) f_corequisite(A,X), prerequisite(X,B) → requisite(A,B)

Dans la quatrième relation, on juge qu'un cours A est un requis à un cours B si le cours A est un prérequis de B ou que le cours A est un corequis de B. La cinquième relation est une relation de transitivité. Si un cours A est prérequis à un autre cours et que cet autre requis est un prérequis à un cours B, alors A est un cours requis pour suivre le cours B. La sixième relation est similaire à la cinquième. Si un cours A est corequis à un autre cours et que cet autre cours est un prérequis à un cours B, alors A est un requis pour B.

Avec ces six relations, on risque d'avoir des doublons dans la liste que l'on retourne à l'utilisateur. En effet, c'est pour ça que nous utilisons la fonction prolog "setof" qui permet d'éliminer les doublons. De plus, dans le cas de la recherche d'un cours qui est un corequis avec plusieurs autre à un autre cours (e.g le cours INF2205 qui a plusieurs autres corequis au cours INF1900), le cours lui-même va apparaître dans la liste des résultats en raison de la relation de symétrie (1). Ainsi, nous éliminons ce cours de la liste à l'aide de la fonction prolog "delete". Notre solution se trouve dans le fichier **q3.pl**. Afin de l'exécuter, il suffit de lancer la commande suivante : *coursAPrendreComplet('NomDeCours',X)*.

Exercice 4 : Akinator

Partie A - Personnes

Pour la première partie de l'exercice 4, nous avons d'abord identifié trois à quatre questions permettant de filtrer la liste des personnes. Afin d'identifier une personne, nous avons procédé en posant des questions sur le sexe de la personne, sa profession, son pays d'origine et, dans le cas où c'est un président, le numéro du président depuis la fondation du pays. (e.g 34e président vs 37e président des É-U). Le tableau suivant présente les caractéristiques que nous avons identifié pour chacune des personnes :

Tableau 2 : Caractéristiques des personnes

	Sexe	Profession	Pays	# Président USA
Michael Jackson	M	singer	usa	-
Mikhail Gorb.	M	resident	russia	-
Jennifer Lawr.	F	actor	usa	-
Hideo Kojima	M	producer	japan	-
Banksy	M	artist	england	-
Lara Croft	F	video game char.	usa	-
Mario	M	video game char.	italy	-
J.K Rowling	F	writer	england	-
Lady Gaga	F	singer	usa	-
Quentin Tarant.	M	director	usa	-
Joseph Staline	M	president	soviet union	-
Dwight D. Eis.	M	president	usa	34th
Cleopatre	F	queen	egypt	-
Victor Hugo	M	writer	france	-
Jesus	M	prophet	palestine	-
Ayrton Senna	M	racer	brazil	-
Moise	M	prophet	egypt	-
Fernando Alons.	M	racer	spain	-
Pape Francois	M	pope	argentina	-
James Bond	M	secret agent	usa	-
Denzel Wash.	M	producer	usa	-
Richard Nixon	M	president	usa	37th

Avec ce tableau, nous avons construit notre base de connaissance, nous avons créé des hypothèses pour chaque personne à l'aide de relations avec des ET logique. Dans notre programme, la variable X représente la personne et la variable Y représente son pays d'origine. Le programme vérifie d'abord le sexe de la personne (homme/femme), son métier puis son pays d'origine. Une fois avoir trouvé un X et un Y qui est valide avec la base de connaissance, il pose des questions en relation avec la personne X qu'il a identifié pour confirmer que c'est bien la personne que l'utilisateur recherche. Pour le cas de Michael Jackson, le programme sait initialement que Michael Jackson est un homme, chanteur originaire des États-Unis. Afin de valider que Michael Jackson est bien la personne que l'utilisateur recherche, il doit avoir la

confirmation de l'utilisateur. Ainsi, il valide auprès de l'utilisateur que le personnage est bien un homme, chanteur, originaire des États-Unis. Une fois confirmé par l'utilisateur, Michael Jackson est déterminé avec certitude, à l'aide de la base de connaissance et de cette confirmation.

Aussi, nous avons jugé redondant d'utiliser plusieurs fois une méthode ask spécifique pour chaque question. Ainsi, nous avons utilisé une seule méthode générale pour poser des questions à l'utilisateur. La solution pour la partie sur les personnes de cet exercice se trouve dans le fichier **q4_partA.pl**. Afin d'exécuter notre solution, il suffit de lancer la commande *personne(X)*. Il suffit de répondre avec « y. » pour un oui et avec « n. » pour un non.

Partie B - Objets

Concernant la partie B, sur les objets, nous avons procédé de façon similaire à la précédente partie. Cette fois, afin d'alléger le code, car nous avons compris le principe, nous avons choisi de catégoriser les objets en fonctions de deux catégories, afin de limiter le nombre de question à poser à l'utilisateur. Afin d'identifier un objet, nous avons choisi de les classer en deux catégories. D'abord, en fonction du fait si l'objet est un objet électrique ou non. Ensuite en fonction de la fonctionnalité de l'objet. Par exemple, une cafetière a pour fonction de faire du café. Ainsi, le tableau suivant contient les catégories de chaque objet ainsi que la valeur associé à chacune de ces catégories.

Tableau 3 : Catégories des objets

Objet	Électrique	Fonction
aspirateur	oui	nettoyer
ordinateur	oui	programmer
téléphone	oui	communiquer
fourchette	non	ustensile
balai	non	nettoyer
cactus	non	enjoliver
assiette	non	contenir de la nourriture
four	oui	cuisson
cuisinière	oui	cuisiner
cafetière	oui	faire du café
grille-pain	oui	griller le pain
table	non	socialiser
casserole	non	cuisiner
shampooing	non	laver les cheveux
Détergent à vaisselle	non	laver la vaisselle
lit	non	dormir
clé	non	déverrouiller
portefeuille	non	transporter des cartes
sac à dos	non	transporter des livres
piano	non	jouer de la musique
lampe	oui	éclairer
papier	non	écrire

C'est à l'aide de ce tableau que nous avons construit nos prédicats et notre base de connaissances, de la même façon qu'expliqué dans la partie A de cet exercice. La solution de la partie B se trouve dans le fichier **q4_partB.pl**. Afin d'exécuter notre programme, il suffit de lancer la commande *objet(X).* Tout comme la partie précédente, il suffit de répondre avec « y. » pour un oui et avec « n. » pour un non.