Louis Clouâtre Latraverse - 1720563
Zihui Zhong - 1687994
Pascal Desrochers - 1689838

INF8215 - Rapport TP2

23 mars 2017

École Polytechnique de Montréal

Projet 1

Nous avons réalisé cette partie en plusieures étapes distinctes. Commençons par la première, la définition de catégories pour pouvoir distinguer les divers objets/personnages. Pour faire cela, nous avons choisi une série de question binaires générales qui sépare ces objets en 2 groupes. À noter qu'une entité peut être dans les 2 groupes si la question est ambigu pour cet entité. Ces données sont sous forme d'un tableau Excel. Ensuite, avec un simple script python, nous avons transformé les données en une base de connaissances Prolog sous la forme: *catégorie(valeur, entité)*. La valeur est oui, ou non dépendamment de la réponse à la question. Ensuite, la fonction questions() est mis en place pour retourner une liste de [catégorie, question] et la fonction possibilities retourne la liste de tous les entités.

Par la suite, il était question de s'assurer que notre base de connaissance peut distinguer toute les entitées. Nous avons donc écrit un programme Prolog qui fait exactement cela. La fonction findCollision(X,Y,Z) essaye de trouver un X, une liste de réponse aux questions, qui match avec 2 différentes entités Y et Z. Pour ce faire, il appelle findMatch, qui avec iteration() extrait les questions de la liste et trouve un match. Avec cette information, nous avons donc ajusté nos questions pour s'assurer qu'aucune collision ne peut arriver.

Finalement, avec une base de connaissance fiable, nous avons effectué le code lui-même. La fonction Objet(X) et Personnage(X), servent à initialiser iteration avec les bon paramètres. Il y a 4 versions de la fonction iteration, la première, iteration(X,[],Pos) sert quand la liste de question est fini, dans ce cas, qui est seulement présent pour la robustesse du programme, on retourne tous les entités retenues. Le deuxième cas est le cas ou Pos, les possibilités restantes à une longueur de 1. Dans ce cas ci, nous avons trouvé la réponse. Le 3e cas est celui où Pos est vide. Dans ce cas ci, l'objet pensé est inexistant dans la base de connaissance. Finalement on arrive au cas général. Dans le cas général, on obtient avec getFirstQuestion la question la plus pertinente, la pose avec askQuestion et continue dans l'itération suivante.

La fonction getFirstQuestion retourne la question la plus pertinente, définie en tant que la question qui divise les possibilités restantes en groupes de tailles les plus similaire. Cette condition est calculé dans la fonction getRatio, qui retourne le ratio entre les deux sous-groups. Ce ratio est défini comme Min(taille A, taille B) / Max(taille A, taille B). La fonction retourne donc la question avec le plus grand ratio et la liste de question en retirant cette dernière question. La fonction askQuestion fait 3 choses : elle affiche le texte de la question, récupère la réponse de l'usager et effectue un filtre selon cette réponse sur les possibilités restantes.

Projet 2

La première fonction implémentée est celle de validation de séquence. Elle fonctionne en vérifiant premièrement que la séquence ne commence pas par un 0. Si elle commence par un 0 on le retire de la séquence et on repasse les contraintes et la nouvelle séquence à la fonction. Si elle commence par un 1, on prend notre première contrainte, la soustrait par 1 et on retire le premier chiffre de la séquence, puis on renvoie le résultat à la fonction valid_seq. Si le premier élément de la contrainte égale 0 mais que le premier élément de la séquence égale 1, la séquence est invalide. On continue jusqu'à ce qu'on ai une liste de contrainte contenant qu'un 0 et une séquence vide. Si on n'obtient pas ce résultat la séquence ne fonctionne pas.

La deuxième fonction implémentée est valid lines. Elle commence par vérifier que la longueur de la ligne est bien égale à la longueur de la liste de contrainte de colonne, puis appelle la fonction list_between qui vérifie récursivement que chaque élément de la ligne est un integer entre 0 et 1. Ensuite, elle appelle la fonction valid_seq avec la première contrainte de la liste de contrainte ainsi que la première ligne de la liste de ligne. Finalement, elle appelle récursivement valid_lines avec les liste de contraintes et de lignes sans leurs premier éléments. Cela continue jusqu'à ce que les deux listes soient bien vides.

La troisième fonction implémentée est extract. La fonction s'appelle récursivement pour créer une liste dans result composé de la valeur indexé à index pour chaque ligne. Ce processus nous permet de créer une colonne.

La quatrième fonction implémentée est valid_columns. Cette fonction appelle la fonction valid_column qui extrait la colonne à l'index, puis valide la séquence sur cette colonne. Ensuite, l'index est incrémenté de 1 et la fonction valid_columns est appelé récursivement sur le reste des spécifications de colonnes jusqu'à épuisement des stocks.

Enfin, la fonction nonogram est implémentée. Celle-ci vérifie la validité des lignes et colonnes puis utilise la fonction print_nonogram donné dans le TP pour donner la réponse.