Projet 1

Pour éviter la répétition les *rules* pour poser des questions, j'ai défini la règle *ask* qui prend une *formulation* et une *réponse*. La question prend la forme : "Est-ce que [formulation] [réponse] ?". Ceci permet de poser plusieurs questions différentes sans écrire pleins de version de *ask*. Une exemple serait "Est-ce que [la personne est de sexe] [homme]?".

Ensuite, la règle *countAnswer* a été écrit pour vérifier s'il y a une seule réponse possible pour l'ensemble de réponses obtenues en unifiant les différentes réponses avec la base de données. Si c'est le cas, cette règle permet de court-circuiter la recherche et de retourner la réponse. Il y a une version qui vérifie dans les données des personnes et une version qui utilise les données pour les objets.

Personne

La base de données des *facts* a été construit de sorte que chaque personne possède un nom, un sexe, un pays, une profession et une spécialité. Par exemple, il y a "*personnage(eugenie_bouchard, femme, canada, athlete, tennis).*"

Ainsi, la recherche se fait relativement facilement. Il suffit de demander dans l'ordre le sexe, le pays, la profession et puis la spécialité. À tout moment, s'il ne reste qu'une possibilité, le questionnement s'arrête et le programme affiche la personne devinée. Si pour une raison quelconque il ne reste aucune possibilité, le programme affiche "*Aucune personne ne correspond a ces criteres*" et retourne *false.*

Objet

La base de données a été construit pour que chaque objet possède un nom, si c'est électronique ou non, un lieu dans la maison, si c'est lourd ou léger et une spécialité. Par exemple, il y a "*chose(telephone, electronique, bureau, leger, communication).*"

La recherche se fait de façon similaire à la recherche de personnes, mais il faut demander dans l'ordre si c'est électronique ou non, un lieu dans la maison, si c'est lourd ou léger et une spécialité. Évidemment, la recherche s'arrête s'il n'y a un objet possible.

Projet 2

Pour commencer, il faut inclure la librairie *clpfd* pour utiliser la fonction *ins/2* proposée dans l'énoncé qui sert à vérifier le domaine d'une liste.

L'algorithme utilisée est l'approche proposé dans l'énoncé. C'est-à-dire qu'on vérifie d'abord ligne par ligne si les contraintes sont respectées. On vérifie ensuite chacune des colonnes. Si toutes les contraintes sont respectées, le nonogramme est valide et il est affiché à l'aide de la fonction *print_nonogram* fourni dans l'énoncé.

Plus en détail, pour l'implémentation de *valid_seq*, on a opté pour une règle qui prend la liste de contraintes, une séquence à vérifier et une valeur indiquant si on commence un nouveau groupement de cases pleines ou non. La dernière valeur sert à identifier si on s'attend à un espace vide (0) ou pas pendant la vérification des contraintes. En effet, si on est en train de compter un groupement de 3 cases pleines, on va mettre cette valeur à 1 et s'il y a une case vide avant la fin du compte de 3 cases pleines, c'est invalide. Pour *valid_lines*, il s'agit simplement de vérifier qu'il y a le bon nombre de lignes, vérifier que c'est des 0 et des 1 et puis vérifier tous les lignes récursivement avec la fonction *valid_seq*. Similairement pour *valid_columns*, on va vérifier chaque colonne avec *valid_seq*, mais il faut d'abord extraire la colonne à vérifier à l'aide de *extract* pour la k-ième colonne.

Pour aider à la compréhension et au développement du code, chaque règle a un commentaire pour expliquer le raisonnement en détail.

À la fin du fichier, il y a plusieurs cas de tests qui permettent de vérifier que le raisonnement fonctionne.