TP Logique

L2 Informatique

Préliminaires

- Le but du TP est de coder en `Prolog`. Il est divisé en 3 parties plus ou moins dures. La première se penche sur la modélisation via la logique. La seconde parle de récursivité et de listes. La dernière partie est réduite à un seul exercice : coder un 2-SAT solver (via la méthode de résolution).
- Les *règles du jeu* sont les suivantes :
  - La présence est *obligatoire*,
  - À côté du code, qui doit être *commenté*, il faut rendre une documentation `Markdown` qui décrit le fonctionnement d'un prédicat dès que celle-ci repose sur plus d'une ou 2 règles.
  - Il faudra soumettre code et documentation dans une archive sur l'ent *1 semaine* après votre dernier TP.
  - vous pouvez vous mettre par groupe de 1 à 3 *maximum*.

PS 1 : n'oubliez pas que vous disposez d'une aide en PDF pour prendre en main le langage, ça peut servir.
PS 2 : l'éditeur s'appelle `SWI-Prolog`, et pour écrire du `Markdown` on utilise un éditeur de texte classique, mais si vous lisez ça c'est que vous avez bien sur du *LIRE LA DOC*.
PS 3 : amusez vous bien.

1 Énigmes et modélisation (niveau promenade dominicale)

Exercice 1 : la fin de la solitude

Vous travaillez pour le site de rencontres `www.mitoc.gov` qui propose des rencontres entre les gens inscrits sur la base des prédicats suivants :

- `personne(I, N, T, C, A)` décrit un(e) inscrit(e). `N` est son nom, `T` sa taille, `C` la couleur de ses cheveux et `A` son âge. `I` est un identifiant (un nombre entier) ;
- `gout(I, M, L, S)` : la personne dont l'identifiant est `I` aime le genre de musique `M`, le genre de littérature `L`, et pratique le sport `S` ;
- `recherche(I, T, C, A)` : la personne `I` recherche un partenaire de taille `T`, ayant des cheveux de couleur `C` et dont l'âge est `A`.

On considère que deux personnes X et Y sont assorties si X convient à Y et si Y convient à X. On dira que X convient à Y si d'une part X convient physiquement à Y (la taille, l'âge et la couleur des cheveux de X sont ceux que Y recherche) et si d'autre part les goûts de X et Y en matière de musique, littérature et sport sont identiques.

1. Donner un ensemble d'assertions représentant le fichier des candidats.
2. Écrire les règles définissant `convient-physiquement(X, Y)` et `ont-memes-gouts(X, Y)`.
3. En déduire le programme qui détermine les couples assortis.

Remarque : donner des noms suffisamment rigolos aux personnes peut amener des points bonus.

Exercice 2 : attention à ne pas dépasser !

On se propose de définir un prédicat permettant de colorier la carte de la Figure 1. Les règles sont les suivantes :

- on dispose de trois couleurs qui sont : vert, jaune et rouge ;
- deux zones contiguës doivent avoir des couleurs différentes.

1. Écrivez un prédicat `coloriage(C1, C2, C3, C4)` qui comportera deux parties. La première partie génère toutes les valeurs possibles de `C1`, `C2`, `C3` et `C4`. La seconde vérifie si les colorations obtenues sont conformes à la carte par l'utilisation du prédicat `X \== Y` sur les couleurs des zones contiguës.
2. Reprenez ce prédicat, et modifiez le programme en déplaçant les tests de différence de couleurs le plus tôt possible dans l'écriture du prédicat, c'est-à-dire en vérifiant les différences de couleurs dès que celles-ci sont instanciées. Quelle en est la conséquence ?

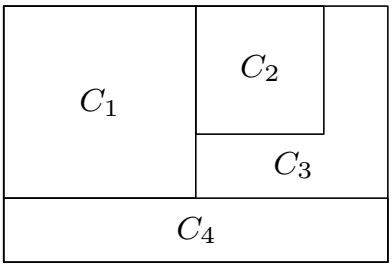

Figure 1

Exercice 3 : globe-trotter

Une agence de voyages propose à ses clients des séjours de une ou deux semaines à Rome, Londres ou Tunis. Le catalogue de l'agence contient, pour chaque destination, le prix du transport (indépendant de la durée) et le prix d'une semaine de séjour qui varie selon la destination et le niveau de confort choisi : hôtel, chambre chez l'habitant ou camping.

1. Écrire l'ensemble des assertions (règles) qui décrivent ce catalogue (les prix sont laissés à votre appréciation).
2. Exprimer la relation `voyage(V, D, H, S)` : le voyage dans la ville `V` pendant `D` semaines avec l'hébergement `H` coûte `S` euros.
3. Compléter par `voyage_economique(V, D, H, S, SMAX)` qui exprime que le coût de ce voyage est inférieur à `SMAX` euros.

2 Listes, ensembles et détectives (niveau randonnée montagnarde)

Exercice 1 : listes

Pour commencer cette section, on s'intéresse aux listes et à quelques opérations basiques. Le but ici est de se familiariser avec des algorithmes récursifs sur les listes donc. On veut développer les fonctionnalités suivantes :

1. renvoyer le premier élément d'une liste,
2. renvoyer le dernier élément,
3. insérer un élément en début de liste,
4. insérer un élément en fin,
5. supprimer un élément d'une liste (et toutes ses occurrences),
6. inverser les éléments d'une liste,
7. vérifier qu'une liste est miroir,
8. fusionner 2 listes,
9. sélectionner les éléments d'indice pair d'une liste,
10. pour un $n \in \mathbb{N}$ donné, générer la liste des n premiers termes de la suite de Fibonacci.

Remarque : ce n'est pas forcément l'exercice le plus fun, mais il est important pour se familiariser avec l'outil liste qui est capital pour tout le reste du TP !

Exemple : voici pour chacune des questions, ce que l'on peut attendre comme réponse (et la structure des prédicats sur les listes !) :

```
?- head(X, [a, b, c, d]).
X = a.

?- last(X, [a, b, c, d]).
X = d.

?- addhead(a, L, [b, c, d]).
L = [a, b, c, d].

?- addlast(d, L, [a, b, c]).
L = [a, b, c, d].

?- remv(a, L, [a, a, b, c, a, d, a, e, a, f]).
L = [b, c, d, e, f].

?- reverse(L, [a, b, c, d]).
L = [b, c, d, a].

?- mirror([a, b, c, c, b, a]).
true.

?- mirror([a, b, c, d, a]).
false.

?- fusion(F, [1, 2, 3], [a, b, c]).
F = [1, a, 2, b, 3, c].

?- even(X, [pair, impair, pair, impair, pair, impair]).
X = [pair, pair, pair].

?- fibo(8, F).
F = [21, 13, 8, 5, 3, 2, 1, 1].
```

Exercice 2 : tous ensemble pour les ensembles

Dans cet exercice on se penche sur des ensembles que l'on va arbitrairement appeler « simples ». On rappelle d'abord qu'un ensemble est une collection non ordonnée d'éléments ne pouvant apparaître qu'une fois chacun. Mais que diable signifie « simple » ? Ce sont des ensembles dont les éléments eux-mêmes *ne sont pas* des ensembles. Illustrons un peu :

- $\{a, b, c, 4, 5\}$ est un ensemble simple,
- $\{\{a, 5, c\}, \{\{a\}, \{b\}\}, \{1, 4, b\}\}$ est un ensemble d'ensembles, il n'est pas simple.

Pour la suite on va se donner des ensembles E, F. Il faut coder chacune des fonctions suivantes au moyen de prédicats `Prolog` :

1. l'appartenance $x \in E$,
2. transformer une liste en ensemble,
3. la relation de sous-ensemble $E \subseteq F$,
4. l'égalité entre deux ensembles,
5. l'intersection $E \cap F = \{x \mid x \in E \wedge x \in F\}$,

6. l'union $E \cup F = \{x \mid x \in E \vee x \in F\}$,
7. la différence $E \setminus F = \{x \mid x \in E \wedge x \notin F\}$,
8. en déduire le complémentaire $\overline{E}$ de E par rapport à un ensemble Q, (on suppose $E \subseteq Q$),
9. la différence symétrique $E \Delta F$ (dont la définition est laissée à votre recherche assidue).

Remarque : l'ensemble vide $\emptyset$ se code par la liste vide `[ ]`.

Exercice 3 : ensembles d'ensembles d'ensembles d'ensembles d'ensembles

En réalité, on va pas aller si loin. On s'intéresse plutôt à une *famille d'ensembles* $\mathcal{F}$ sur un univers donné Q. $\mathcal{F}$ est alors un sous-ensemble de l'ensemble des parties 2^Q de Q. Autrement dit, $2^Q = \{E \mid E \subseteq Q\}$ et $\mathcal{F} \subseteq 2^Q$. On donne un exemple pour clarifier.

Exemple : Soit $Q = \{1, 2, 3\}$. On a alors :

$$2^Q = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

À présent, une famille d'ensembles $\mathcal{F}$ sur Q peut être par exemple donnée par $\mathcal{F} = \{\{1\}, \{2, 3\}, \{1, 3\}\}$. On a bien $\mathcal{F} \subseteq 2^Q$ puisque chaque ensemble de $\mathcal{F}$ est une partie de Q. On remarque que $\mathcal{F}$ n'est pas fermée par intersection : il faudrait lui ajouter $\{2, 3\} \cap \{1, 3\} = \{3\}$ ainsi que $\{1\} \cap \{3\} = \emptyset$.

Pour simplifier la tâche, on pose un ordre total implicite sur Q. Ainsi, l'ordre d'apparition des éléments de Q dans un sous-ensemble F de Q sera toujours le même. Si par exemple $Q = \{a, b, c\}$ on « fixe » $a < b < c$ et ainsi on considère que cet ordre est toujours respecté : on peut avoir $\{a, c\}$ mais pas $\{c, a\}$. On veut coder chacune des fonctions suivantes :

1. calculer 2^Q à partir de Q (sans répétitions),
2. pour $\mathcal{F} \subseteq 2^Q$, calculer la *fermeture par intersection* (voir exemple) de $\mathcal{F}$, c'est-à-dire, si F_1 et F_2 sont dans F, alors il faut aussi y ajouter $F_1 \cap F_2$,
3. en utilisant les opérations de complémentaire et de fermeture par intersection, calculer la fermeture par union de $\mathcal{F}$.

Remarque : il est fort probable qu'il faille coder plus d'un prédicat par question pour résoudre cet exercice.

Exercice 4 : si je tombe, tu tombes avec moi

Bravo, vous venez d'être embauché(e) comme détective à l'université. Votre tâche est de coffrer sans pitié les bandes d'enseignants et étudiants corrompus. L'avantage, c'est que suivant la structure de la bande, il peut suffire de capturer un petit groupe de personnes pour aussi embarquer tout le reste, par effet « avalanche ». Cependant, votre carrière débute seulement et vous risquez d'avoir de nombreux groupes de personnes malhonnêtes à mettre derrière les barreaux. Aussi aimeriez vous avoir un programme qui fasse la sale besogne à votre place.

Heureusement, vous avez suivi un cours de méthodes discrètes. Vous pouvez donc modéliser ces mafias comme suit. Appelons M un ensemble de personnes mal intentionnées. Afin de capturer la structure du groupe on va introduire une relation binaire A (comme « Avalanche ») qui associe deux personnes p_1, p_2 de M. On notera $p_1 A p_2$ pour dire *« si on attrape p_1, p_2 tombe aussi »*. Attention ! A n'est pas symétrique (les lois des malfrats sont parfois impénétrables).

Les énigmes à résoudre sont les suivantes :

1. Etant donné (M, A) et $F \subseteq M$, déterminer toutes les personnes que l'on capturera par la relation A partant de F.
2. Pour une paire (M, A) donnée, trouver les groupes de personnes minimaux par inclusion qui permettent de capturer tout le monde. Autrement dit, on veut trouver les plus petits F dont la réponse serait M dans la question précédente.

3. Calculer tous les ensembles de personnes « capturables », c'est-à-dire les $F \subseteq M$ qui ne permettent pas d'embarquer d'autres personnes que celles de F. Autrement dit, on cherche les $F \subseteq M$ pour lesquels la première question renvoie F.

Exemple : Votre première affaire concerne 5 personnes : Vincent, Aurélie, Lucas, Oscar, Yannick. On pose donc $M = \{v, a, l, o, y\}$. La relation A est illustrée dans la Figure 2. Une paire (M, A) peut être représentée par un graphe dirigé où M sont les sommets et un arc (u, v) signifie uAv. Par exemple dans la figure, on a aAo, c'est-à-dire *« Si Aurélie tombe, Oscar aussi. »*. Quand on choisit $F = \{l, y\}$ et qu'on suit tous les chemins possibles (en vert), on capture au final $\{l, y, v, o\}$ mais pas a. Un ensemble qui permette de capturer tout le monde est par exemple $F = \{a, y\}$. Il est en plus minimal car on ne peut ni récupérer y par a, ni a par y : on ne peut enlever ni l'un ni l'autre de F.

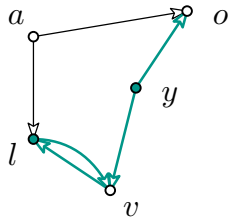

FIGURE 2 – Exemple de relation A sur $M = \{v, a, l, o, y\}$

3 2-SAT Solver (plus jamais ça)

Le problème SAT pour (SATisfiability) est très célèbre en informatique. Et pour cause, c'est le premier problème qui a été montré NP-complet. Les années 70 venaient de commencer. La question est simple : on se donne une formule booléenne $\phi(x_1, \ldots, x_n)$ et on veut savoir en un temps raisonnable (par rapport à la taille de ϕ) si, oui ou non, il existe une assignation $I(x_1, \ldots, x_n)$ des variables $x_1, \ldots, x_n$ de ϕ qui la rende vraie. Bientôt 50 ans ont passé, et la question résiste toujours … Néanmoins, il existe des versions spécifiques du problème. C'est le cas par exemple de 3-SAT, qui pose la même question que SAT mais sous la contrainte que ϕ soit une CNF avec des clauses de taille au plus 3 littéraux. Malheureusement, cette version du problème est aussi dure que la première.

En revanche, le problème 2-SAT est, lui au moins, soluble en un temps raisonnable. C'est exactement à lui que l'on s'attaque ici. Notre question donc est la suivante : nous est donnée une formule booléenne $\phi(x_1, \ldots, x_n)$ sous forme de 2-CNF. Est-elle satisfiable ? Avant de continuer, faisons un exemple.

Exemple : Pour rappel, une CNF est une *conjonction* ($\wedge$) de clauses *disjonctives* ($\vee$). Par exemple

$$\phi = (p \vee \neg q) \wedge (q \vee r) \wedge (\neg r)$$

est une conjonction de 3 clauses (entre parenthèses). C'est une CNF. De plus, chacune des clauses ne contient au plus que deux littéraux. C'est donc une 2-CNF. Peut-on trouver une assignation de p, q, r qui satisfasse la formule ? Premièrement, puisque $\neg r$ est une clause, il faut prendre $I(r) = 0$. Ensuite, on a $(q \vee r)$ mais on a déjà choisi $I(r) = 0$, il faut donc assigner 1 à q. De même pour $p \vee \neg q$, puisque $I(q) = 1$, on doit choisir $I(p) = 1$. Donc, ϕ est satisfiable, notamment quand $I(p) = I(q) = 1$ et $I(r) = 0$, on a bien $I(\phi) = 1$.

Pour savoir si une 2-CNF est satisfiable, on va utiliser le principe de *résolution* : partant de clauses de la forme $(p \vee q)$ et $(\neg q \vee r)$, on peut déduire la clause $(p \vee r)$. Formellement

$$\{(p \vee q), (\neg q \vee r)\} \vdash (p \vee r)$$

Intuitivement, on peut se représenter $(p \vee q)$ et $(\neg q \vee r)$ comme les implications $\neg p \to q$ et $q \to r$ respectivement. De ce point de vue, $\neg p \to r$ (équivalent à $p \vee r$) est direct par transitivité.

L'algorithme pour 2-SAT est donc le suivant : étant donné ϕ, calculer toutes les résolutions possibles et les ajouter. Quand ϕ est saturée, i.e. on ne peut plus ajouter de nouvelles clauses par résolution, il est nécessaire et

suffisant (Théorème de Krom) de vérifier qu'il n'y ait pas de clauses unitaires contradictoires présentes (de la forme (p) et $(\neg p)$).

Exemple : Reprenons $\phi = (p \vee \neg q) \wedge (q \vee r) \wedge (\neg r)$ et appliquons l'algorithme. Premièrement on résout toutes les paires possibles et on ajoute les clauses produites à la formule :

- $(p \vee \neg q)$ et $(q \vee r)$ donnent $p \vee r$,
- $(p \vee \neg q)$ et $(\neg r)$ ne résolvent pas,
- $(q \vee r)$ et $(\neg r)$ donnent q.

Nous avons maintenant $\phi = (p \vee \neg q) \wedge (q \vee r) \wedge (\neg r) \wedge (p \vee r) \wedge (q)$. Il se trouve que l'on peut encore résoudre des clauses, on réitère donc la précédente étape. En fait, une seule résolution est possible ici : $(p \vee r)$ avec $(\neg r)$ donnant p. On obtient donc $\phi = (p \vee \neg q) \wedge (q \vee r) \wedge (\neg r) \wedge (p \vee r) \wedge (q) \wedge (p)$ qui est saturée. Aucun littéral n'est contradictoire, ϕ est satisfiable.

Exemple : Prenons maintenant $\phi = (a \vee b) \wedge (\neg b \vee a) \wedge (\neg a)$. On remarque que $(a \vee b)$ et $(\neg b \vee a)$ peuvent se résoudre en a que l'on ajoutera à ϕ. Cependant, on aura $\phi = (a \vee b) \wedge (\neg b \vee a) \wedge (\neg a) \wedge (a)$ qui est insatisfiable puisqu'il faudrait valider à la fois a ET $\neg a$.

Exercice (enfin !) : Donner en `Prolog` un prédicat `sat` qui prend en paramètres une 2-CNF, et qui renvoie `true` si celle-ci est satisfiable, et `false` sinon. Utiliser l'algorithme de résolution ! On suppose que :

- une 2-CNF est representée comme une liste de listes de longueur 2,
- si a est un littéral, alors sa negation $\neg a$ est codée par `not(a)`,
- si il n'y a qu'un seul littéral a dans une clause, on le double, c.-à-d. on écrit `[a, a]`, puique $a \leftrightarrow a \vee a$.

Par exemple, la formule $\phi = (a \vee b) \wedge (\neg b \vee a) \wedge (\neg a)$ s'écrira :

```
F = [[a, b], [not(b), a], [not(a), not(a)]]
```

Liens utiles

- *Learn Prolog Now*, Blackburn, Bos et Striegnitz : `http://www.learnprolognow.org/`,
- *Documentation Prolog* : `http://www.swi-prolog.org/pldoc/doc_for?object=manual`,
- *Markdown guide* : `https://www.markdownguide.org/`,
- *Markdown cheatsheet* : `https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet`