**Bases de données relationnelles**

# TP8 : DATALOG

Le but de cette séance de TP est de vous faire decouvrir les DATALOG, et mieux comprendre PROLOG. Nous utiliserons DES (Datalog Educational System), un freeware de DATALOG du professeur Saenz-Perez d'une université de Madrid. Lui-même, DES repose sur SWI-PROLOG, freeware PROLOG de l'université d'Amsterdam. Vous pouvez récupérez des distributions (Linux, windows ou Max OS X) de DES librement sur le site http://des.sourceforge.net et de SWI-PROLOG sur le site http://www.swi-prolog.org.

## DES

- Le binaire DES (version 6.3.2) est disponible sur les machines du M5 dans */home/enseign/DES/des*. Vous devez saisir ce chemin manuellement. En combinant DES avec *rlwrap* (déjà vu en TP1 avec RA) vous aurez des avantages (correction sur la ligne courante et historique). La création d'un alias est conseillée (voir TP1 pour la syntaxe).
- Le signal d'invité "DES>" attend que vous saisissiez un *but* (requete) que le moteur DATALOG tentera de satisfaire.
- Dans l'interprète, vous pouvez obtenir de l' aide en tapant /help.
- Puisque vous n'avez pas encore chargé de base, vous ne pouvez pas encore faire grand chose. Une possibilité est de tester une comparaison 42=42. Observez le résultat. Comparez avec le résultat de 42=2015. L'opérateur d'inégalité vous servira, il est \=
- **Similairement a l'outil de normalisation, pour charger une base de connaissance (faits, regles), il faut charger le fichier qui les contient, avec les commandes suivates :**
- Dans l'interprète, vous pouvez changer de répertoire avec /cd , faire afficher le répertoire courant avec /pwd, et charger une base (nommée *mabase.dl*) avec /consult mabase.dl. Les commandes suivantes sont equivalentes :

```
DES> /c exemple
DES> /consult exemple
DES> /c exemple.dl
DES> /consult exemple.dl
```

- Lorsqu'un fichier est chargé (par /consult) la base chargée précédemment est "oubliée". Donc un "/consult" réinitialise la base de donnée extensionnelle dans le fichier "consulté".
- Si vous voulez accumuler plusieurs fichiers, utilisez /reconsult a partir du second.
- **Il n'est pas prevu d'alimenter l'interprete directement avec des regles, ou des definitions de predicats. En le faisant, vous n'ajoutez pas les predicats a la base de connaissance.**
- Avec la commande /listing vous pouvez afficher le contenu de la base de connaissance courante. Ce prédicat peut également prendre en argument le prédicat dont on souhaite connaître la définition courante. Par exemple dans l'exercice suivant :

```
DES> /listing articles
```

- Pour quitter DES, tapez /terminate. Vous vous retrouverez ensuite au niveau de SWI-PROLOG, donc l'interprète vous affiche "?-". Pour quitter SWI-PROLOG, tapez halt. (avec le "." !).

*Rappel : en* DATALOG*, tout identificateur commençant par une majuscule est une variable.*

# 1 La boutique

Le script datalog pour creer la boutique est disponible sur moodle. Chargez-le, testez des exemples du cours, egalement sur moodle.

Sachez que Datalog mémorise les résultats déjà calculés. Vous pouvez les afficher à tout moment avec la commande /list_et .

Dans un fichier *requetes1_boutique.dl* que vous rendrez, formulez des buts pour :

**Question 1** faire afficher tous les noms d'articles.

Nous rappelons qu'en Prolog ou Datalog, tout ce qui commence avec une majuscule est une variable. Lorsqu'on doit utiliser une variable mais qu'on ne désire pas connaître son instanciation on utilise une *variable anonyme*. Celle-ci est représentée par le symbole _ (tiret bas).

**Question 2** En utilisant un variable anonyme, interrogez DATALOG pour savoir si il existe un article de couleur bleue.

**Question 3** Interrogez DATALOG pour savoir si il existe un fournisseur pour un article de couleur bleue.

**Question 4** faire afficher tous les noms d'articles vendus par kiventout.

Dans un fichier *requetes2_boutique.dl* que vous rendrez, formulez des **predicats** :

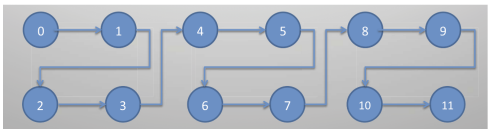
**Question 5** predicat *nomArtKiventout* : pour afficher les noms des articles que kiventtout vend, en tapant dans l'inteprete la requete $nomArtKiventout(X)$

**Question 6** predicat *couleurRare* : pour afficher les couleurs rares, pour lesquelles il n'y a qu'un seul article, avec la requete $couleurRare(X)$.

**Question 7** Définissez un prédicat *deuxrouges* tel que la requête $deuxrouges(X)$ rende les noms de fournisseurs d'au moins *deux* articles rouges.

**Question 8** Définissez un prédicat *bonmarche* tel que la requête $bonmarche(X)$ rende les noms de fournisseurs d'articles à moins de 10 euros.

# 2 Graphes




**Question 9** Saisissez le graphe de l'image dans un nouveau fichier *graph.dl* (que vous rendrez). Saisissez les assertions de cette base dans un fichier sous forme de *faits* DATALOG, qui définissent le prédicat $e/2$ (pour edge en anglais)

**Question 10** Utilisez le prédicat $p/2$ du cours pour afficher toutes les paires de sommets connectés, quelque soit la longueur du chemin (p pour path en anglais). Rendez votre requête, et son résultat.

**Question 11** Ajoutez un ou plusieurs cycles au graphe. Puis, retestez $p/2$.

**Question 12** Pouvez-vous determiner combien de fois un cycle est traversé ? Pensez-vous que Datalog reconnait la presence d'un cycle ?

**Question 13** Déclarez un prédicat $impair(X, Y)$ qui est vrai s'il existe un chemin de longueur impaire entre les sommets $X$ et $Y$. A l'aide d'une requête, affichez les paires de sommets, entre lesquels existe un chemin de longueur impaire.

**Question 14** Déclarez un prédicat $injoignable(X, Y)$ qui est vrai lorsqu'il n'existe aucun chemin entre les sommets $X$ et $Y$.

# 3 Résolution en SWI Prolog

A rendre : un fichier *crisefinanciere_reponse.dl*

Enregistrez le fichier *crisefinanciere.pl* disponible sur moodle, et lisez-le attentivement. Veillez à conserver le nom crisefinanciere.pl avec l'extension **.pl** pour prolog. Dans le répertoire où vous avez enregistré le fichier, lancez SWI prolog à partir du terminal (commande : swipl).

**Question 15** Posez une requête permettant de vous faire afficher les dettes.

**Question 16** Posez une requête pour afficher quelle fille évite quelle autre. Assurez-vous de faire afficher *toutes* les réponses. Qu'observez-vous ?

**Question 17** Comparez le résultat des mêmes requêtes en DES, et expliquez d'où viennent les différences entre les deux systèmes.

# 4 Analyse d'un programme non stratifiable

Nous revenos sur le paradoxe du barbier présenté en cours. Dans un village, deux catégories d'hommes existent : ceux qui se rasent eux-mêmes, et ceux qui ne se rasent pas eux-mêmes. Le barbier rase tous ceux qui ne se rasent pas eux-mêmes. En Datalog, cela se traduit en :

```
homme(barbier).
homme(maire).
rase(barbier,H) :- homme(H), not(rase(H,H)).
```

**Question 18** Après avoir saisi le code dans un fichier *barbier.dl*, activez en DES le mode *verbose* avec la commande */verboseon* chargez-le et observez le retour de DES. Quel message important obtenez-vous ? Que signifie-t-il ?
**Question 19** Analyser l'information memorisee pour $not(rase(maire, maire)$ et $not(rase(barbier, barbier)$ (qu'on obtient avec la commande /list_et ). Expliquer la negation dans la call table
**Question 20** Lancez la requête $rase(X, Y)$. Vous obtenez deux réponses, de types differenst : l'une a une valeur fiable (elle a été prouvé vraie), et l'autre une valeur logique *undefined*. Formulez en francais : quelle est l'infomation prouvée vraie, et quelle est problématique ?
**Question 21** Quels sont les faits qui ont été prouvés au cours de l'évaluation de votre requête précédente ? Comment les obtenir de DES ? Quelle est l'information négative prouvée par DES ? Quelle est l'information pour laquelle DES ne donne pas de valeur de vérité ?
**Question 22** Ajoutez un nouveau prédicat $est\_rase(X) : -rase(barbier, X)$. Qu'est DES capable de renseigner concernant l'état du barbier, pourquoi et comment ?

# 5 Arbre généalogique

A rendre : un fichier *famille.dl*.

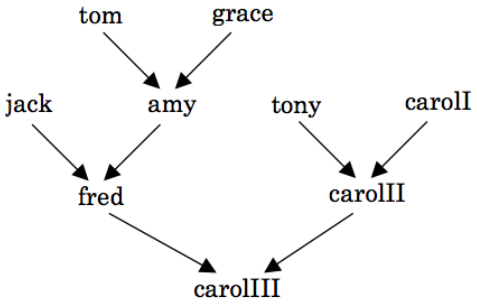
Dans cet exercice, vous travaillez la récursivité, à l'exemple d'un arbre généalogique. Le programme $P$ définit quatre prédicats, $pere/2$, $mere/2$, $parent/2$, et $ancetre/2$. La base contient uniquement des informations sur les relations immédiates entre parents et enfants. Par exemple, pour le fait que Tom est le père d'Amy, on retrouve $pere(tom, amy)$ dans la EDB. Attention à l'ordre de lecture, pour laquelle nous convenons que le premier arguement d'un prédicat est toujours le sujet, pour tous les predicats. On pourrait donc lire $pere(X, Y)$ comme $X$ *est le père de* $Y$, etc.

Les autre relations sont définies par des règles récursives. Prolog peut déduire du programme que la IDB Pour le contient par exemple $ancetre(grace, fred)$, correspondant au fait que Grace est l'ancêtre de Fred. Cette information implicite est obtenue avec une règle récursive, et une règle de base, non récursive. Comparez à la définition de lien direct dans un graphe, et atteignabilité par un chemin.

```
% la EDB
pere(tom,amy).
pere(jack,fred).
mere(grace,amy).
mere(amy,fred).
% le programme (les regles)
parent(X,Y) :- mere(X,Y).
parent(X,Y) :- pere(X,Y).

ancetre(X,Y) :- parent(X,Y).
ancetre(X,Y) :- parent(X,Z), ancetre(Z,Y).
```

Le code ci-haut ne donne qu'un fragment de la EDB. Voici l'arbre complet :




**Question 23** Dans un fichier *famille.dl*, saisissez la EDB complète, donc toutes les personnes qui apparaissent dans l'image, pour les prédicats $pere/2$ et $mere/2$. Remplacez les chiffres romains par des chiffres arabes.
**Question 24** Vérifiez (en définissant un *but* DATALOG) que *grace* et *amy* sont des meres, mais que *tony* n'est pas une mere.
**Question 25** Interrogez DATALOGpour connaitre tous les noms de meres.
**Question 26** Saisissez egalement le prédicat $ancetre/2$, et testez-le. Expliquez son fonctionnement, avec 1-2 phrases que vous rendez dans votre compte-rendu.
**Question 27** Calculez toutes les conséquences possibles du programme, à l'aide de DATALOG. Rendez les requêtes qui vous permettent de faire cela, ainsi que leur résultats, dans votre compte-rendu.

**Question 28** Programmer un prédicat *cousin/2*, qui est vrai pour deux personnes deux cousin(e)s.

Le but du reste de cet exercice est de programmer un nouveau prédicat récursif *mg/2*, à lire, *même génération* . Cette tache se décompose en deux questions :

**Question 29** L'idée du *cas de base* est simple : chaque personne est de la même génération qu'elle-même. Pourtant, puisqu'il n'est pas définit ce qu'est une personne, vous devrez penser a ce point. Lorsque votre définition du cas de base sera correcte, elle devrait produire :

```
DES> mg(X,Y)

{
  mg(amy,amy),
  mg(carol1,carol1),
  mg(carol2,carol2),
  mg(carol3,carol3),
  mg(fred,fred),
  mg(grace,grace),
  mg(jack,jack),
  mg(tom,tom),
  mg(tony,tony)
}
Info: 9 tuples computed.

DES>
```

**Question 30** Pour le cas récursif, sachez qu'il faut remonter de la racine de l'arbre aux feuilles. Avec la bonne définition, vous obtiendrez l'affichage suivant pour la requête $mg(X, Y)$ :

```
DES> mg(Y,M)
{
  mg(amy,amy),
  mg(amy,carol1),
  mg(amy,jack),
  mg(amy,tony),
  mg(carol1,amy),
  mg(carol1,carol1),
  mg(carol1,jack),
  mg(carol1,tony),
  mg(carol2,carol2),
  mg(carol2,fred),
  mg(carol3,carol3),
  mg(fred,carol2),
  mg(fred,fred),
  mg(grace,grace),
  mg(grace,tom),
  mg(jack,amy),
  mg(jack,carol1),
  mg(jack,jack),
  mg(jack,tony),
  mg(tom,grace),
  mg(tom,tom),
  mg(tony,amy),
  mg(tony,carol1),
  mg(tony,jack),
  mg(tony,tony)
}
Info: 25 tuples computed.
```