

# Programmation Logique – TP noté

## Les sorites de Lewis Carroll

Narendra Jussien

Mars 2002

### Modalités pratiques

Ce TP est à rendre avant le **lundi 15 avril 2002 18 heures**. Vous m'enverrez, par mail, le code commenté de votre application Prolog. N'oubliez pas de consulter l'annexe A concernant le détail du code à rendre.

## 1 Introduction

Le révérend Dodgson plus connu sous le pseudonyme de Lewis Carroll, écrivain et logicien britannique du siècle dernier (1832 – 1898) a proposé des raisonnements appelés *sorites* dont il faut trouver la conclusion. Voici un exemple (que vous connaissez déjà)<sup>1</sup> de ces raisonnements :

- Quand je résous un exercice de logique sans ronchonner, vous pouvez être sûr que c'en est un que je comprends.
- Ces sorites ne sont pas disposés régulièrement, comme les exercices auxquels je suis habitué.
- Aucun exercice facile ne me donne jamais mal à la tête.
- Je ne comprends pas les exercices qui ne sont pas disposés régulièrement, comme ceux auxquels je suis habitué.
- Je ne ronchonne jamais devant un exercice, à moins qu'il ne me donne mal à la tête.
- Donc ...

Le but du TP est de définir en Prolog un certain nombre d'outils permettant de résoudre ces sorites. À la fin du TP, on pourra simplement faire :

```
| ?-   init,  
      premisses("les exercices qui sont resolus sans ronchonner sont ceux que je comprends"),  
      premisses("cet exercice n'est pas dispose regulierement"),  
      premisses("aucun exercice qui est facile n'est migraineux"),  
      premisses("les exercices qui ne sont pas disposes regulierement ne sont pas parmi ceux que je comprends"),  
      premisses("les exercices sont resolus sans ronchonner a moins qu'ils ne soient migraineux"),  
      solution(X).  
X = 'cet exercice n'est pas facile'
```

<sup>1</sup>. Ces explications proviennent de [L'Hospitalier, 1998] et les sorites elles-mêmes proviennent de [Carroll, 1966].

La particularité des sorites de Lewis Carroll est qu'elles peuvent être résolues en utilisant le principe de résolution car chaque prémisse ne doit servir qu'une et une seule fois. En modélisant donc chaque prémisse sous forme de clause et, partant d'une quelconque de ces prémisses, en appliquant le principe de résolution en combinaison avec les autres clauses, on obtient la solution de la sorite.

Nous allons écrire notre outil en plusieurs étapes. Tout d'abord, nous allons nous attacher à la résolution symbolique de la sorite *i.e.* en supposant les prémisses déjà modélisées. Puis nous allons nous occuper de la reconnaissance de la structure de chaque phrase afin de modéliser automatiquement chaque prémisse. Il s'agit en fait d'une sorte de traitement du langage naturel.

Vous trouverez en annexe B de ce sujet différentes sorites que vous pouvez utiliser pour tester votre programme et que vous retrouvez dans le fichier `~jussien/Fi3/tp-note.pl`. Dans ce même fichier, vous trouverez un prédicat `init/0` qui est à appeler avant chacun des tests que vous ferez : il permet de nettoyer correctement la base de faits.

## 2 Résolution symbolique

Une clause  $p(x) \vee \neg q(y) \vee r(a)$  sera représentée par la liste : `[p(X), non(q(Y)), r(a)]`. Les variables libres sont donc les variables quantifiées universellement et les constantes conservent leur expression.

1. Donner les clauses obtenues en modélisant l'exemple.
2. Écrire le prédicat `resolution(Clause1,Clause2,Resu)` qui applique un pas du principe de résolution entre les clauses `Clause1` et `Clause2`, le résultat étant renvoyé dans `Resu`.

```
| ?- resolution([non(bebe(X)),illogique(X)],[non(illogique(Y)),meprise(Y)],R).
R = [non(bebe(X)),meprise(X)],
Y = X
| ?- resolution([a(X), b(X)], [c(X), d(X)], R).
no
| ?- resolution([a(X), b(X)], [non(b(Y)), non(a(Y))], R).
R = [b(X), non(b(X))],
X = Y
```

3. Écrire le prédicat `rsorite(Clauses, Cl, Resultat)` qui utilise toutes les clauses<sup>2</sup> de `Clauses` par résolution au départ de `Cl` renvoyant le résultat dans `Resultat`.

```
| ?- rsorite([[non(crocodile(X)),non(meprise(X))],[non(illogique(Y)),meprise(Y)]],
             [non(bebe(Z)),illogique(Z)], R).
R = [non(bebe(_A)),non(crocodile(_A))]
```

4. Sachant que les clauses provenant de la modélisation sont stockées dans des faits Prolog `clauseLogique(Cl)` écrire le prédicat `sorite(L)` qui renvoie `L` : réponse obtenue en appliquant le principe de résolution sur l'ensemble des clauses modélisées.

```
%% Programme
clauseLogique([non(bebe(X)),illogique(X)]).
clauseLogique([non(crocodile(X)),non(meprise(X))]).
clauseLogique([non(illogique(Y)),meprise(Y)]).
```

---

2. Attention, elles ne sont peut être pas placées dans le bon ordre dans la liste.

```

%% But
| ?- sorite(L).
L = [non(bebe(_A)),non(crocodile(_A))]

```

### 3 Traitement des expressions en langage naturel

Pour le traitement des expressions représentant les prémisses, nous allons adopter un traitement en trois temps :

1. Décomposition de la chaîne de caractères en une liste d'atomes (mots). Les séparateurs d'atomes (mots) seront les espaces (code 32).
2. Reconnaissance d'un *modèle* (pattern) dans la liste représentant la chaîne de caractères. Ce modèle correspondra à une règle de modélisation.
3. Enregistrement des propriétés – correspondance entre une série de mots et un prédicat au sens logique.

#### 3.1 Outils de manipulation de chaînes

- À l'aide du prédicat prédéfini `name/2`, écrire un prédicat `traduit/2` prenant en entrée une chaîne de caractères<sup>3</sup> et fournissant en sortie une liste d'atomes composant cette chaîne.

```

| ?- traduit("tous les coups sont permis",L).
L = [tous,les,coups,sont,permis]
| ?- traduit("Les poissons n'ont pas d'os",L).
L = ['Les',poissons,'n\'ont',pas,'d\'os']

```

- Pour faciliter l'écriture des prémisses, nous allons systématiquement passer les mots rencontrés au singulier. Notre outil de mise au singulier ne va pas être très perfectionné, il se contente d'enlever un `s` final lorsqu'il y en a un. Nous conserverons quand même quelques exceptions : gens, dans, moins, vos (dont le singulier est votre), gris, sans, vous, tres, ...

Écrire le prédicat `singulier/2` qui place dans le deuxième paramètre le singulier de l'atome passé en premier paramètre.

```

| ?- singulier(chats,S).
S = chat
| ?- singulier(vos,S).
S = votre
| ?- singulier(mieux,S).
S = mieux

```

- On écrira ensuite le prédicat `tousSingulier/2` qui passera au singulier tous les atomes apparaissant dans le premier paramètre (une liste d'atomes) pour les mettre dans le deuxième paramètre (une autre liste).
- On cherchera aussi à supprimer les mots non significatifs dans la chaîne passée par l'utilisateur. Ainsi, *il, elle, ils, elles, un, une, des, le, la, les, mon, ma, mes, ton, ta, tes, son, sa, ses* sont non significatifs dans une phrase. Écrire le

---

3. Rappel : en Prolog, une chaîne de caractères, c'est une *liste* de codes ASCII.

prédicat `supprimeNS/2` qui supprime dans la liste d'atomes passée en paramètre les mots non significatifs. Le résultat est alors renvoyé dans le deuxième paramètre.

```
| ?- supprimeNS([le, chat, de, la, voisine],L).
L = [chat,de,voisine] ?
yes
```

- Les concepts (propriétés) que nous manipulons dans la résolution symbolique ne sont formés que d'un unique atome. Écrire le prédicat `unSeulAtome/2` qui forme un unique atome à partir d'une liste d'atomes.

```
| ?- unSeulAtome([le,chat,de,la,voisine],A).
A = 'le chat de la voisine'
```

- Pour faciliter l'écriture des prémisses et permettre à l'utilisateur d'utiliser différentes expressions pour représenter la même propriété, nous allons définir le prédicat `tresProches(L1,L2)` qui réussit si deux listes d'atomes sont *assez proches* pour décider qu'elles représentent la même expression. La règle sera la suivante: soit  $N$  le minimum des longueurs des deux listes,  $L_1$  et  $L_2$  seront considérées comme *assez proches* si le nombre d'atomes en commun est supérieur strictement à  $N/2$ .

```
| ?- tresProches([les,gens,illogiques],[illogiques]).
yes
| ?- tresProches([les,bebes],[les,gens,illogiques]).
no
```

## 3.2 Outils de manipulation des concepts

- Nous choisirons de stocker les nouvelles propriétés par l'intermédiaire de l'ajout dans l'univers Prolog (prédicat `assert/1`) de nouveaux faits de la forme: `propriete(Atome,ListeAtomes)` où `Atome` est l'atome représentant la propriété considérée et `ListeAtomes` est la liste d'atomes l'ayant généré. Nous procéderons de même pour les constantes apparaissant dans les prémisses, avec le prédicat `constante/2`.

Vous trouverez dans le fichier `~jussien/Fi3/tp-note.pl` le prédicat `ajoutePropriete(Atome,ListeAtomes,AtomeR)` où `Atome` est le nouvel atome que l'on souhaite créer (à partir de la liste d'atomes `ListeAtomes`) et renvoyant l'atome à utiliser réellement (`AtomeR`). Il s'agit d'`Atome` s'il n'existe pas de propriété déjà rentrée assez proche; sinon il s'agit de l'atome déjà créé. Ce prédicat affiche en plus une trace de son comportement.

```
| ?- ajoutePropriete('peut tuer un crocodile', [peut,tuer,un,crocodile], P).
Nouvelle propriete : peut tuer un crocodile
P = 'peut tuer un crocodile'
| ?- ajoutePropriete('peut massacrer un crocodile', [peut,massacrer,un,crocodile], P).
Propriete : peut tuer un crocodile
P = 'peut tuer un crocodile'
```

- On écrira de la même façon le prédicat `ajouteConstante/3`.

- Écrire le prédicat `proprieteAssociee(A,Ta,X)` qui à partir de l'atome `X` et de l'élément `X` construit le terme structuré `Ta` correspondant à `A(X)` et ajoute la propriété traitée à l'ensemble des propriétés (en utilisant `ajoutePropriete/3`).

```
| ?- proprieteAssociee([gentil,professeur],T,Jussien).
Nouvelle propriete : gentil professeur
T = 'gentil professeur'(Jussien) ?
yes
```

- Vous trouverez dans le fichier `~jussien/Fi3/tp-note.pl` le prédicat `constanteAssociee/2` qui fait la même chose pour les constantes.

### 3.3 Reconnaissance de *patterns*

Nous allons maintenant nous intéresser à la reconnaissance de *patterns*. Pour cela, nous allons passer chaque prémisses rentrée par un *filtre* qui nous permettra de la modéliser facilement.

Nous allons reconnaître les modèles suivants: *les A sont B, tous les A sont B, aucun A n'est B, les non A sont B, seuls les A sont B<sup>4</sup>, les A ne sont pas B, nul n'est A quand B, A est B, A n'est pas B*. `A` et `B` représente des variables contenant les parties de phrase permettant d'unifier phrase et modèle. Pour chacun des modèles précédents, **donner** la modélisation symbolique (sous forme de clause) associée.

#### 3.3.1 Filtrage avec joker

Avant de prendre en compte les variables ayant un nom nous allons considérer un modèle simple de filtrage permettant d'accepter des jokers (`?` pour un seul mot et `*` pour un ensemble éventuellement vide de mots).

Le prédicat `filtrage/2` permet de filtrer une liste passée en premier paramètre par un modèle passé en deuxième paramètre.

```
filtrage([], []).
filtrage([], [*]).           % une étoile remplace du vide !
filtrage([X|Xs],[?|Ys]) :- % un seul caractère
    filtrage(Xs,Ys).
filtrage([X|Xs],[*|Ys]) :- % premier cas pour * : remplace vide
    filtrage([X|Xs],Ys).
filtrage([X|Xs],[*|Ys]) :- % deuxieme cas, remplace au moins X
    filtrage(Xs,[*|Ys]).
filtrage([X|Xs],[X|Ys]) :-
    filtrage(Xs,Ys).
```

#### 3.3.2 Filtrage avec variable

1. En vous inspirant de la version de `filtrage/2` avec joker, écrivez une version du prédicat `filtrage/2` qui accepte des variables libres à la place des `*` et qui les unifie avec la liste des atomes remplacés en cas de succès.

```
| ?- filtrage([les,chats,de,la,voisine,sont,gris],[les,A,sont,B]).
A = [chats,de,la,voisine], B = [gris]
```

2. Écrire le prédicat `pattern/1` qui prend en paramètre une liste d'atomes, reconnaît un des *patterns* à reconnaître, crée les propriétés apparaissant et éventuellement les constantes apparaissant (on utilisera les prédicats `constanteAssociee`

---

4. Cette expression qui normalement est une équivalence sera ici considérée comme *Les A sont B*.

et `proprieteAssociee`), puis ajoute la clause logique modélisant le pattern considéré à la base de faits Prolog.

## 4 Résolution du problème

À l'aide de tout ce que vous avez écrit jusqu'à présent, écrire les prédicats :

- `premise/1` qui prend en entrée une chaîne de caractères, la met sous forme de liste d'atomes, la simplifie (suppression des mots non significatifs et mise au singulier), reconnaît un *pattern* connu, crée les propriétés à créer, et crée la clause correspondante.

```
| ?- premise("les bebes sont illogiques").
Nouvelle propriete : bebe
Nouvelle propriete: illogique
Modelisation : [non(bebe(_969)),illogique(_969)]
yes
| ?- premise("nul n'est meprise quand il peut venir a bout d'un
crocodile").
Nouvelle propriete : peut venir a bout d'un crocodile
Propriete : meprise
Modelisation : [non(peut venir a bout d'un
crocodile(_2103)),non(meprise(_2103))]
yes
| ?- premise("les gens illogiques sont meprises").
Propriete : illogique
Nouvelle propriete: meprise
Modelisation : [non(illogique(_1225)),meprise(_1225)]
yes
```

- Écrire le prédicat `solution(X)` qui collecte les clauses logiques, appelle le prédicat `sorte/1` pour récupérer la clause conclusion de l'ensemble des prémisses et la met sous une forme un peu plus lisible (on pourra utiliser le prédicat `expression/2` fourni dans le fichier `~jussien/Fi3/tp-note.pl`).

```
| ?- solution(X).
X = 'les helleniste sont cheveux roux' ?
yes
```

## 5 Améliorations

Pour traiter le plus de sorites possibles :

1. Améliorer le programme en traitant aussi les prémisses de la forme : *les U qui sont A sont B*. Autrement dit, les prémisses dans lesquels l'univers de calcul est exprimé explicitement. On sauvegardera cette information sous la forme d'un fait Prolog et on vérifiera pour les prémisses suivantes que l'univers ne change pas.

```
| ?- premise("les exercices qui sont resolus sans ronchonner sont ceux que je comprends").
Univers : exercice
Nouvelle propriete : resolu sans ronchonner
Nouvelle propriete : ceux que je comprend
Modelisation : [non(resolu sans ronchonner(_3564)),ceux que je comprend(_3564)]
yes
| ?- premise("cet exercice n'est pas dispose regulierement").
Nouvelle constante : cet exercice
Nouvelle propriete : dispose regulierement
```

```

Modelisation : [non(dispose regulierement(cet exercice))]
yes
| ?- premisses("aucun exercice qui est facile n'est migraineux").
Nouvelle propriete : facile
Nouvelle propriete : migraineux
Modelisation : [non(facile(_2244)),non(migraineux(_2244))]
yes
| ?- premisses("les exercices qui ne sont pas disposes regulierement ne sont pas parmi ceux que je comprends").
Propriete : ceux que je comprend
Propriete : dispose regulierement
Modelisation : [non(ceux que je comprend(_4191)),dispose regulierement(_4191)]
yes
| ?- premisses("les exercices sont resolus sans ronchonner a moins qu'ils ne soient migraineux").
Propriete : resolu sans ronchonner
Propriete : migraineux
Modelisation : [resolu sans ronchonner(_3648),migraineux(_3648)]
yes
| ?- solution(X).
X = 'cet exercice n'est pas facile' ?
yes

```

2. Proposer une gestion des termes antonymes (comme internes/externes) plus naturelle que d'être obligé de donner explicitement la modélisation de cette information permettant de résoudre la sorite(*cf.* exemple `ecole/1`)<sup>5</sup>.
3. Traiter les *équivalences i.e.* ne plus tenir compte de la note de la page 5. Pour cela, on pourra par exemple, essayer de résoudre la sorite avec la totalité des clauses modélisées puis si ce n'est pas possible (ce qui arrive lorsqu'on utilise une équivalence) essayer avec une clause de moins, et ainsi de suite, jusqu'à ce qu'on arrive à prouver quelque chose.

## A Liste des réponses à fournir dans le code commenté

Vous fournirez deux fichiers : le premier contenant la version de base du système de résolution de sorites; le seconde la version modifiée avec toutes les améliorations que vous aurez intégrées. Ce dernier fichier devra comporter un entête de commentaires décrivant vos améliorations et la stratégie mise en œuvre pour les prendre en compte.

### A.1 Contenu du premier fichier

1. les clauses pour l'exemple
2. les prédicats `resolution/3`, `rsorite/3`, `sorite/1`.
3. les prédicats `traduit/2`, `singulier/2`, `tousSingulier/2`, `supprimeNS/2`, `unSeulAtome/2`, `tresProches/2`.
4. les prédicats `ajouteConstante/3`, `proprieteAssociee/3`.
5. la modélisation associée à chacun des *patterns* fournis.
6. les prédicats `filtrage/2`, `pattern/1`.
7. les prédicats `premisses/1`, `solution/1`.

---

5. **Remarque :** Cette amélioration peut être liée à la suivante.

## A.2 Contenu du deuxième fichier

1. modifications de la reconnaissance de pattern pour gérer l'univers de calcul exprimé explicitement.
2. modifications de la résolution pour gérer les équivalences et les termes antonymes.

## B Exemples

1. Les bébés **bebes**(X)
  - Les bébés sont illogiques ;
  - Nul n'est méprisé quand il peut venir a bout d'un crocodile ;
  - Les gens illogiques sont méprisés ;
  - Donc ...
2. Les cadeaux **cadeaux**(X)
  - Seules les casseroles m'appartenant sont en fer blanc ;
  - Tous les cadeaux venant de vous sont fort utiles ;
  - Aucune des casseroles m'appartenant n'est utile ;
  - Donc ...
3. Les logiciens **logiciens**(X)
  - Tous les individus sains d'esprit sont de possibles logiciens ;
  - Aucun malade mental n'est un juré possible ;
  - Aucun de vos enfants n'est un logicien possible ;
  - Les non malade mental sont sains d'esprit
  - Donc ...
4. L'école **ecole**(X)
  - Aucun enfant de moins de douze ans dans cette école n'est interne ;
  - Tous les enfants studieux ont les cheveux roux ;
  - Aucun des externes n'est helléniste ;
  - Seuls les élèves de moins de douze ans sont paresseux ;
  - Les non externes sont internes ;
  - Les non paresseux sont studieux
  - Donc ...
5. Les canards **canards**(X)
  - Les canards de ce village ne sont pas à col en dentelle à moins qu'ils ne soient à Mme Martin,
  - Aucun canard de ce village qui est a Mme Martin n'est gris,
  - Donc ...
6. Les fox-terriers **fox**(X)
  - Aucun fox-terrier n'est parmi les signes du zodiaque ;

- Aucun objet qui n'est pas parmi les signes du zodiaque n'est une comète ;
- Seuls les fox-terriers ont la queue bouclée ;
- Donc ...

7. Les chatons **chatons(X)**

- Aucun chaton qui est mangeur de poisson n'est réfractaire à l'étude ;
- Aucun chaton qui est sans queue n'est prêt à jouer avec un gorille ;
- Les chatons qui sont moustachus sont mangeurs de poisson ;
- Aucun chaton qui est amoureux de l'étude n'est aux yeux verts ;
- Aucun chaton n'est avec queue s'il n'est pas moustachu ;
- Les non avec queue sont sans queue ;
- Les non amoureux de l'étude sont réfractaires à l'étude ;
- Donc ...

8. Les exercices de logique **exercices(X)**

- Les exercices qui sont résolus sans ronchonner sont ceux que je comprends ;
- Cet exercice n'est pas disposé régulièrement ;
- Aucun exercice qui est facile n'est migraineux ;
- Les exercices qui ne sont pas disposés régulièrement ne sont pas parmi ceux que je comprends ;
- Les exercices sont résolus sans ronchonner à moins qu'ils ne soient migraineux ;
- Donc ...

9. Les animaux **animaux(X)**

- Les bestiaux qui ne sont pas rieurs ne sont pas excitables ;
- Les anes n'ont pas de cornes ;
- Les buffles sont tres forts ;
- Aucun bestiaux qui est rieur n'est facile à avaler ;
- Aucun bestiaux qui n'est pas a corne n'est très fort ;
- Les bestiaux sont excitables sauf s'ils sont des buffles ;
- Donc ...

## Références

[Carroll, 1966] Lewis Carroll. *Logique sans peine*. Hermann, 1966.

[L'Hospitalier, 1998] Yvon L'Hospitalier. *Énigmes et Jeux logiques*. Eyrolles, 1998.