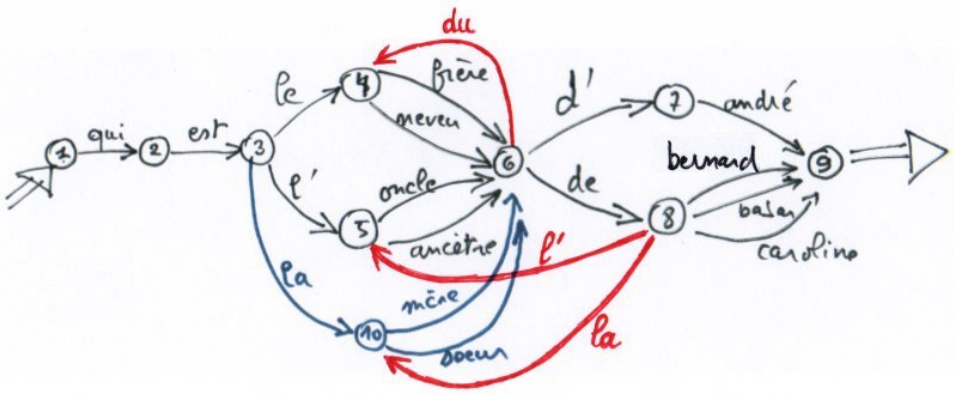
7 Interrogation langage naturelle

L'objectif de ce TD est d'écrire un prédicat qui répond à des questions sur l'arbre généalogique (du TD 1) en formulant ces questions en langue naturelle. Les questions ont une forme *qui est le/la père/mère/frère/soeur... [du/de la père/mère/frère/soeur...[...]] de andré/augustine/bernard...* (les crochets designent ce qui est facultatif, les barres obliques / désignent les choix multiples) sachant que :

- on a le droit d'emboîter autant de relations de parenté que l'on veut ;
- les noms à la fin sont ceux présents dans votre arbre généalogique ;
- une question sera codée sous la forme d'une liste de mots.

L'automate d'états fini qui réalise cette analyse est le suivant.



7.1 Analyse syntaxique

A l'aide de cet automate, écrire le prédicat `analyse(+L)` qui réalise l'analyse syntaxique de la phrase dont les mots sont donnés dans la liste L. Exemple de liste : `[qui, est, le, pere, du, frere, de, andre]`.

7.2 Application à la généalogie

Puisque notre but est de traduire les questions du français au Prolog, après vérification de leur syntaxe avec l'automate, on procède à une *analyse sémantique* (rudimentaire) qui consiste ici à éliminer dans un premier temps les mots *vides* (non pertinents pour la traduction sémantique en Prolog).

1. Définir un prédicat `motVide` qui répond vrai si le mot passé en paramètre fait partie des mots *inutiles* dans le traitement des questions de l'analyse ci-dessus.
2. Définir un prédicat qui *nettoie* une question de ses mots *vides*.

   ```
   ?- nettoie([qui,est,le,pere,de,babar],L).
   L = [pere,babar]
   ```

3. Il reste à transformer la liste nettoyée en un prédicat qui interroge l'arbre généalogique : par exemple, `[pere,babar]` en `pere(X,babar)`. Pour cela on utilise les prédicats `=..` et `call`.
   Écrire le prédicat `reponse1([Predicat,Individu],X)` qui est vrai si X est une réponse à la question initiale qui avait été posée et nettoyée en `[Predicat,Individu]`.

   ```
   ?- reponse1([pere,babar],X).
   X = andre
   ```

4. Même question, mais avec les prédicats avec emboitement (du genre "qui est le cousin du pere de dagobert ? "). Evidemment, il peut y avoir un nombre non borné d'emboitements donc la définition doit être récursive.

```
?- reponse([cousin,pere,dagobert],X).
X = clement
```

5. Écrire un prédicat qui analyse une question complète (avec analyse syntaxique et sémantique) et donne sa réponse.

```
?- question([qui,est,le,cousin,du,pere,de,dagobert],X).
X = clement
```

6. Écrire des prédicats nécessaires pour interroger la base et obtenir la réponse :

```
?- question([qui,est,le,cousin,du,pere,de,dagobert]).
le cousin du pere de dagobert est : clement
```

7. Proposer des prédicats nécessaires pour afficher toutes les réponses :

```
?- question2([qui, est, l, oncle, du, pere, de, dagobert]).
l oncle du pere de dagobert est : babar
l oncle du pere de dagobert est : bernard
```

7.3 Encore mieux

Voici un petit plus. Si on veut partir d'une question sous la forme d'une vraie chaîne de caractères, et non plus d'une liste, voici un ensemble de prédicats qui transforment une chaine de caractères entre guillemets (") où les mots sont séparés par un nombre quelconque de blancs et se terminant par un '?' en une liste de mots. Les nombres qui interviennent dans ces prédicats sont les codes ASCII des caractères spéciaux :

— 32 est le code ASCII de l'espace,
— 63 est le code ASCII du point d'interrogation,
— 44 est le code ASCII de la virgule.

```
/* lecture au clavier d'une question (reconnue au fait qu'elle se termine par un point
d'interrogation) et transcription en une liste d'atomes */
vocab([],[]) :- !.
vocab([32|LC],Lex) :- ! , vocab(LC,Lex).        /* élimination des blancs inutiles */
vocab([44|LC],[","|Lex]) :- ! , vocab(LC,Lex).  /* insertion de la virgule */
vocab([63|_],[]) :- !.                          /* on s'arrête dès que l'on rencontre un
                                                   point d'interrogation */
vocab(Liste,[Mot|Lex]) :- mot(Liste,Mot,Suite), vocab(Suite,Lex).
mot([],[],[]).
mot([32|S],[],S) :- !.      /* détection d'un blanc */
mot([C|LC],[C|M],S) :- mot(LC,M,S).
liste([],[]).
liste([Mot|P],[Atome|L]) :- name(Atome,Mot),liste(P,L).
transforme(Phrase,Liste) :- vocab(Phrase,P),liste(P,Liste).
```

Après cela, on peut poser la requête :

```
?- transforme("qui est  le     cousin du pere  de dagobert ?", L).
L = [qui, est, le, cousin, du, pere, de, dagobert]
```

Donc, en l'intégrant dans l'analyse, on peut poser la question sous la forme d'une chaîne entre guillemets, terminée par un " ?". Écrire des prédicats nécessaires pour interroger la base et obtenir la réponse :

```
?- interro("qui est le cousin du pere de dagobert ?").
le cousin du pere de dagobert est : clement
```