

## **T.P. Traitement Automatique des Langues (Retoré)** **Master Info 1ère année — Université Bordeaux 1**

Il s'agit d'illustrer le cours par une grammaire de clauses définies à écrire en Sicstus Prolog. En fait Prolog permet d'écrire la DCG comme un ensemble de règles de production, mais pour comprendre le fonctionnement des grammaires en Prolog, on écrira les clauses correspondantes; du reste cette syntaxe Prolog pour les grammaires n'est qu'un confort purement syntaxique: si on liste le programme entré, ce sont les clauses et non règles de la DCG qui apparaissent. Le manuel de Sicstus Prolog est disponible sur internet à l'adresse: <http://www.sics.se/isl/sicstus/>

On souhaite modéliser par des clauses écrites en Prolog la grammaire de phrases simples avec des relatives qui sont aussi bien introduite par le pronom « qui » (dont l'antécédent est le sujet de la relative) que par le pronom « que » (dont l'antécédent est l'objet de la relative). Pierre regarde Marie qui dort. Marie regarde Pierre qui mange une pomme. Marie qui regarde Pierre tombe. Pierre dort. Les pommes tombent. Outre les pronoms relatifs mentionnés, les mots considérés sont les articles définis et indéfinis, deux noms propres « Pierre » et « Marie », les noms communs « pomme », « femme », les verbes transitifs « regarder », et « manger », les verbes intransitifs « dormir » et « tomber ».

**1** Donner une grammaire algébrique (ou non contextuelle) engendrant ces phrases et le moins possible de phrases incorrectes, sans se soucier des phénomènes d'accord ni de vraisemblance.

**2** Donner un programme Prolog (un ensemble de clauses) modélisant cette grammaire de sorte que pour une phrase modélisée par des clauses sans corps  $entre(mot_{i+1}, place_i, place_{i+1})$ . on ait  $S(0, n)$  si et seulement si cette phrase appartient au langage engendré par la grammaire.

**3** On ajoute la clause  $entre(Mot, [Mot|Reste], Reste)$  et on supprime les clauses  $entre(mot_{i+1}, place_i, place_{i+1})$ . servant à entrer la phrase. Montrer que la requête  $s([mot_1, \dots, mot_n], [])$  répond « yes » ou « no » suivant que la phrase est ou non engendrée par la grammaire.

**4** La phrase: « Marie dorment. » est elle produite par votre grammaire? La phrase: « Le pomme mange Marie. » est-elle reconnue? Modifier la grammaire Prolog pour éviter ce genre d'inconvénients en utilisant des traits (appelés aussi attributs), c'est-à-dire des variables supplémentaires. Il faut donc veiller à ce qu'un

nom ait un article dont le nombre et le genre soient les mêmes. De même il faut que le verbe s'accorde avec son sujet, y compris dans les relatives; mais il faut aussi faire en sorte qu'un verbe dont le sujet (ou l'objet) doit être animé ou au contraire inanimé ait bien un sujet (ou un objet) animé ou non, y compris dans les relatives. Pour simplifier, vu le vocabulaire choisi, on se contera d'accorder le verbe en nombre (tout les noms sont des troisièmes personnes), et de vérifier un trait humain ou non humain. On notera que ces deux types de problèmes sont de nature différentes: l'accord est un phénomène morphologique (ou morpho-syntaxique) et la nature du sujet ou de l'objet d'un verbe est un phénomène sémantique; ils sont néanmoins traités de manière similaire dans les grammaires d'unification dont les DCG.

5 Plutôt qu'un simple « yes » ou « no » on souhaite obtenir l'arbre d'analyse de la phrase de sorte que  $S(T, [mot_1, \dots, mot_n], [])$  donne pour  $T$  l'analyse syntaxique de la phrase, puis « yes ».

```
|?- s(T, [une, femme, que, pierre, regarde, tombe], []).
T=
[phrase,
 [syntagme_nominal, une, femme,
 [relative, que,
 [syntagme_nominal, pierre, [t]], regarde]],
 [syntagme_verbal, tombe]]
```

**1** Il faut distinguer les relatives introduites par « qui » et « que ». Bien sûr la présence d'une relative n'est pas nécessaire, d'où les productions vides (et si on les omet, le programme boucle).

<i>s</i>	→	<i>sn sv</i>
<i>sn</i>	→	<i>det n reln</i>
<i>sn</i>	→	<i>det n rela</i>
<i>sn</i>	→	<i>np reln</i>
<i>sn</i>	→	<i>np rela</i>
<i>reln</i>	→	<i>pron sv</i>
<i>rela</i>	→	<i>proa sn vt</i>
<i>reln</i>	→	$\epsilon$
<i>rela</i>	→	$\epsilon$
<i>sv</i>	→	<i>vi</i>
<i>sv</i>	→	<i>vt sn</i>
<i>proa</i>	→	<i>que</i>
<i>pron</i>	→	<i>qui</i>
<i>vi</i>	→	<i>tombe</i>
<i>vi</i>	→	<i>tombent</i>
<i>vt</i>	→	<i>regarde</i>
<i>vt</i>	→	<i>regardent</i>
<i>vt</i>	→	<i>mange</i>
<i>vt</i>	→	<i>mangent</i>
<i>vi</i>	→	<i>dort</i>
<i>vi</i>	→	<i>dorment</i>
<i>det</i>	→	<i>une</i>
<i>det</i>	→	<i>un</i>
<i>det</i>	→	<i>la</i>
<i>det</i>	→	<i>le</i>
<i>det</i>	→	<i>des</i>
<i>det</i>	→	<i>les</i>
<i>n</i>	→	<i>pommes</i>
<i>n</i>	→	<i>pomme</i>
<i>n</i>	→	<i>femme</i>
<i>n</i>	→	<i>femmes</i>
<i>np</i>	→	<i>pierre</i>
<i>np</i>	→	<i>marie</i>

2 Il suffit d'introduire pour chaque non terminal deux places, et un prédicat  $entre(m,X,Y)$  signifiant qu'entre les places  $X$  et  $Y$  on a le mot  $m$ . Une clause:

$$sn \longrightarrow det \ n \ reln$$

est transformée en

$$sn(P,Q) \longrightarrow det(P,X) \ n(X,Y) \ reln(Y,Q)$$

Ce qui se comprend ainsi: s'il existe des places  $X,Y$  pour lesquelles

- entre  $P$  et  $X$  on a un déterminant,
- entre  $X$  et  $Y$  on a un nom
- entre  $Y$  et  $Q$  on a une relative,

alors entre  $P$  et  $Q$  on a un syntagme nominal. Il reste ensuite à entrer la phrase ou le constituant, par exemple « une pomme qui tombe » sous la forme  $entre(une,0,1)$   $entre(pomme,1,2)$ ,  $entre(qui,2,3)$  et

$entre(tombe,3,4)$ . Alors on a bien « yes » pour la requête  $sn(0,4)$  (ou  $sn(0,2)$  compte tenu de  $reln(Z,Z)$ ) et « no » pour  $s(0,3)$ . Voici les clauses Prolog correspondantes:

```
s(P,Q):-sn(P,X), sv(X,Q).
sn(P,Q):- det(P,X),n(X,Y), reln(Y,Q).
sn(P,Q):- det(P,X),n(X,Y), rela(Y,Q).
sn(P,Q):- np(P,X), reln(X,Q).
sn(P,Q):- np(P,X), rela(X,Q).
reln(P,Q):- pron(P,X), sv(X,Q).
rela(P,Q):- proa(P,X), sn(X,Y), vt(Y,Q).
reln(P,P).
rela(P,P). sv(P,Q):-vi(P,Q).
sv(P,Q):-vt(P,X),sn(X,Q).
proa(P,Q):- entre(que,P,Q).
pron(P,Q):- entre(qui,P,Q).
vi(Q):-entre(tombe,P,Q).
vi(P,Q):-entre(tombent,P,Q).
vt(P,Q):-entre(regarde,P,Q).
vt(P,Q):-entre(regardent,P,Q).
vt(P,Q):-entre(mange,P,Q).
```

```

vt(P,Q):-entre(mangent,P,Q).
vi(P,Q):-entre(dort,P,Q).
vi(P,Q):-entre(dorment,P,Q).
det(P,Q):-entre(une,P,Q).
det(P,Q):-entre(un,P,Q).
det(P,Q):-entre(la,P,Q).
det(P,Q):-entre(le,P,Q).
det(P,Q):-entre(des,P,Q).
det(P,Q):-entre(les,P,Q).
n(P,Q):-entre(pommes,P,Q).
n(P,Q):-entre(pomme,P,Q).
n(P,Q):-entre(femme,P,Q).
n(P,Q):-entre(femmes,P,Q).
np(P,Q):-entre(pierre,P,Q).
np(P,Q):-entre(marie,P,Q).

```

% ce moyen d'entrer les phrases est particulièrement pénible,

% la question suivante améliore tout cela

```

entre(pierre,0,1).
entre(qui,1,2).
entre(tombe,2,3).
entre(mange,3,4).
entre(une,4,5).
entre(pomme,5,6).
% donne s(0,6) mais pas s(0,5)
entre(une,0,1).
entre(pomme,1,2).
entre(tombe,2,3).
entre(pierre,3,4).
% donne s(0,3) mais pas s(0,4).
entre(marie,0,1).
entre(dorment,1,2).
% donne s(0,2)

```

**3** On peut vérifier (par récurrence) que lorsqu'on appelle un prédicat  $u(liste_1, liste_2)$  alors  $liste_1$  s'écrit  $[mot_1 | \dots | mot_k | liste_2]$  et  $u(liste_1, liste_2)$  est vrai lorsque les

clauses permettent d'établir que la suite de mots  $mot_1, \dots, mot_k$  est de type  $u$ . Le mieux est de faire tourner un petit exemple à la main, par exemple

$$s([une, femme, regarde, pierre], []).$$

Abrégeons les mots en  $u, f, r, p$ .

On a  $entre(f, [f, r, p], [r, p])$   
 et  $entre(u, [u, f, r, p], [f, r, p])$   
 et donc  $n([f, r, p], [])$  et  $det([u, f, r, p], [f, r, p])$ .

Comme  $rela([r, p], [r, p])$   
 (par la clause  $sn(P, Q) : -det(P, X), n(X, Y), rela(Y, Q).$ )  
 on obtient (avec  $X = [f, r, p]$  et  $Y = [r, p]$ )  
 $sn([u, f, r, p], [r, p])$ . (1)

On a  $entre(p, [p], [])$  et donc  $np([p], [])$ .  
 Avec  $rela([], [])$ , par la clause  $sn(P, Q) : -np(P, X), rela(X, Q).$ ,  
 on a  $entre(p, [p], [])$  et donc avec  $X = []$   
 on obtient  $sn([p], [])$ . (2)

On a  $entre(r, [r, p], [p])$  et donc  $vt([r, p], [p])$ .  
 Donc par la clause  $sv(P, Q) : -vt(P, X), sn(X, Q)$ . avec  $X = [p]$   
 et en utilisant (2) on a  
 $sv([r, p], [])$ . (3)

Finalement, à partir de (1) et (3) par la clause:  $s(P, Q) : -sn(P, X), sv(X, Q)$ .  
 on a  $s([u, f, r, p], [])$  avec  $X = [r, p]$ .

**4** Bien sur les grammaires non contextuelles ne sont pas suffisantes pour rendre compte des phénomènes d'accord. Le principe pour en rendre compte est de distribuer le même trait lorsque la! valeur de ce trait doit être la même dans deux constituants....mais ensuite les constituants qui sont des prédicats doivent bien sûr toujours inclure ces traits, même à un endroit où ils ne servent plus à rien. Cela provoque d'ailleurs des « warnings » de Prolog, qui s'inquiète de ces variables inutilisées ensuite. Chaque verbe requiert le caractère humain ou non humain de son sujet et de son objet, et il faut donc transmettre à la relative (qui contient un verbe) les propriétés de son antécédent. Dans le cas d'une relative introduite par « que » l'antécédent devient l'objet de la phrase subordonnée, et dans une relative introduite par « qui » le sujet de la phrase subordonnée.

$entre(\text{Word}, [\text{Word}|\text{Rest}], \text{Rest})$ .

s(P,Q):-sn(P,X,N,G,H), sv(X,Q,N,H,Ho).  
sn(P,Q,N,G,H):- det(P,X,N,G), n(X,Y,N,G,H), reln(Y,Q,N,H,Ho).  
sn(P,Q,N,G,H):- det(P,X,N,G),n(X,Y,N,G,H), rela(Y,Q,N,Hs,H).  
sn(P,Q,N,G,H):- np(P,X,N,G,H), reln(X,Q,N,H,Ho).  
sn(P,Q,N,G,H):- np(P,X,N,G,H), rela(X,Q,N,Hs,H).  
reln(P,Q,N,H,Ho):- pron(P,X), sv(X,Q,N,H,Ho).  
rela(P,Q,N,H,Ho):- proa(P,X), sn(X,Y,No,G,H), vt(Y,Q,No,H,Ho).  
reln(P,P,N,H,Ho).  
rela(P,P,N,H,Ho).  
sv(P,Q,N,H,Ho):-vi(P,Q,N,H).  
sv(P,Q,N,H,Ho):-vt(P,X,N,H,Ho),sn(X,Q,No,G,Ho).  
proa(P,Q):- entre(P,Q).  
pron(P,Q) :- entre(P,Q).vi(P,Q,sing,X):-entre(tombe,P,Q).  
vi(P,Q,plur,X):-entre(tombent,P,Q).  
vt(P,Q,sing,hum,X):-entre(regarde,P,Q).  
vt(P,Q,plur,hum,X):-entre(regardent,P,Q).  
vt(P,Q,sing,hum,nhum):-entre(mange,P,Q).  
vt(P,Q,plur,hum,nhum):-entre(mangent,P,Q).  
vi(P,Q,sing,hum):-entre(dort,P,Q).  
vi(P,Q,plur,hum):-entre(dorment,P,Q).  
det(P,Q,sing,fem):- entre(une,P,Q).  
det(P,Q,sing,masc):- entre(un,P,Q).  
det(P,Q,sing,fem):- entre(la,P,Q).  
det(P,Q,sing,masc):- entre(le,P,Q).  
det(P,Q,plur,G):- entre(des,P,Q).  
det(P,Q,plur,G):- entre(les,P,Q).  
n(P,Q,plur,fem,nhum):- entre(pommes,P,Q).  
n(P,Q,sing,fem,nhum):- entre(pomme,P,Q).  
n(P,Q,sing,fem,hum):-entre(femme,P,Q).  
n(P,Q,plur,fem,hum):-entre(femmes,P,Q).  
np(P,Q,sing,masc,hum):-entre(pierre,P,Q).  
np(P,Q,sing,fem,hum):-entre(marie,P,Q).  
l?-s([pierre,que,marie,que,pierre,regarde,regarde,regarde,une,pomme,qui,tombe],[]).  
yes  
l?-s([pierre,que,marie,que,une,pomme,regarde,regarde,regarde,une,pomme,qui,tombe],[]).

no  
 l?- s([une,pomme,qui,regarde,pierre,tombe],[]).  
 no  
 l?- s([une,pomme,que,pierre,regarde,tombe],[]).  
 yes

**5** Il suffit d'adjoindre à tout les prédicats décrivant les constituants une variable qui va récursivement prendre la valeur de la liste construite; on peut être plus ou moins précis. Par exemple j'ai choisi de ne pas mentionner le type d'une expression constituée d'un seul mot, mais de noter par [t] l'absence de relative après un nom ou un nom propre.

entre(Word,[Word|Rest],Rest).  
 s([phrase,SN,SV],P,Q):-sn(SN,P,X,N,G,H), sv(SV,X,Q,N,H,Ho).  
 sn([syntagme\_nominal,DET,NOM,RELN],P,Q,N,G,H):-  
 det(DET,P,X,N,G), n(NOM,X,Y,N,G,H), reln(RELN,Y,Q,N,H,Ho).  
 sn([syntagme\_nominal,DET,NOM,RELA],P,Q,N,G,H):-  
 det(DET,P,X,N,G), n(NOM,X,Y,N,G,H), rela(RELA,Y,Q,N,Hs,H).  
 sn([syntagme\_nominal,NP,RELN],P,Q,N,G,H):-  
 np(NP,P,X,N,G,H), reln(RELN,X,Q,N,H,Ho).  
 sn([syntagme\_nominal,NP,RELA],P,Q,N,G,H):-  
 np(NP,P,X,N,G,H), rela(RELA,X,Q,N,Hs,H).  
 reln([relative,PRON,SV],P,Q,N,H,Ho):-  
 pron(PRON,P,X), sv(SV,X,Q,N,H,Ho).  
 rela([relative,PROA,SN,VT],P,Q,N,H,Ho):-  
 proa(PROA,P,X), sn(SN,X,Y,No,G,H), vt(VT,Y,Q,No,H,Ho).  
 reln([t],P,P,N,H,Ho).  
 rela([t],P,P,N,H,Ho).  
 sv([syntagme\_verbal,VI],P,Q,N,H,Ho):-vi(VI,P,Q,N,H).  
 sv([syntagme\_verbal,VT,SN],P,Q,N,H,Ho):-  
 vt(VT,P,X,N,H,Ho), sn(SN,X,Q,No,G,Ho).  
 proa(que,P,Q):-entre(que,P,Q).  
 pron(qui,P,Q):-entre(qui,P,Q).  
 vi(tombe,P,Q,sing,X):-entre(tombe,P,Q).  
 vi(tombent,P,Q,plur,X):-entre(tombent,P,Q).  
 vt(regarde,P,Q,sing,hum,X):-entre(regarde,P,Q).

vt(regardent,P,Q,plur,hum,X):-entre(regardent,P,Q).  
 vt(mange,P,Q,sing,hum,nhum):-entre(mange,P,Q).  
 vt(mangent,P,Q,plur,hum,nhum):-entre(mangent,P,Q).  
 vi(dort,P,Q,sing,hum):-entre(dort,P,Q).  
 vi(dorment,P,Q,plur,hum):-entre(dorment,P,Q).  
 det(une,P,Q,sing,fem):-entre(une,P,Q).  
 det(un,P,Q,sing,masc):-entre(un,P,Q).  
 det(la,P,Q,sing,fem):-entre(la,P,Q).  
 det(le,P,Q,sing,masc):-entre(le,P,Q).  
 det(des,P,Q,plur,G):-entre(des,P,Q).  
 det(les,P,Q,plur,G):-entre(les,P,Q).  
 n(pommes,P,Q,plur,fem,nhum):-entre(pommes,P,Q).  
 n(pomme,P,Q,sing,fem,nhum):-entre(pomme,P,Q).  
 n(femme,P,Q,sing,fem,hum):-entre(femme,P,Q).  
 n(femmes,P,Q,plur,fem,hum):-entre(femmes,P,Q).  
 np(pierre,P,Q,sing,masc,hum):-entre(pierre,P,Q).  
 np(marie,P,Q,sing,fem,hum):-entre(marie,P,Q).  
  
 l?-s(T,[une,femme,que,pierre,regarde,tombe],[]).  
 T=[phrase,[syntagme\_nominal,une,femme,  
   [relative,que,[syntagme\_nominal,pierre,[t]],regarde]],  
   [syntagme\_verbal,tombe]]?  
 l?-s(T,[pierre,que,marie,que,pierre,  
   regarde,regarde,regarde,une,pomme,qui,tombe],[]).  
 T=[phrase,[syntagme\_nominal,pierre,  
   [relative,que,[syntagme\_nominal,marie,  
   [relative,que,[...!...!...]],regarde]],  
   [syntagme\_verbal,regarde,  
   [syntagme\_nominal,une,pomme,[relative,qui,[...!...]]]]]?  
 % un peu long, l'arbre ne s'affiche pas totalement....

**Pour en savoir plus, je recommande vivement le livre suivant, simple et clair, dont le seul défaut est de n'être pas traduit:**

**Fernando Pereira & Stuart Shieber. Prolog and Natural Language Analysis. CSLI Lectures Notes (number 10), 1987. Distributed by Cambridge University Press.**

**Pour ceux que cela intéresse, un peu de sémantique:**

*Si on en a le temps, on pourra modifier la grammaire de sorte qu'elle construise un lambda terme simplement typé qui représente la sémantique de la phrase. Il ne s'agit pas d'évaluer le terme mais simplement de le construire formellement: Prolog n'est pas pratique pour définir la  $\beta$ -réduction. Les types de base sont  $e$  (entités) et  $t$  (valeurs de vérité), et voici la sémantique des mots:*

*Nom propre:  $\lambda PP(\text{pierre}) : (e \rightarrow t) \rightarrow t$  Nom commun:  $\lambda xpomme(x) : e \rightarrow t$*

*Verbe intransitif  $\lambda xdormir(x) : e \rightarrow t$  Verbe transitif:  $\lambda xy.manger(x,y) : e \rightarrow (e \rightarrow t)$*

*Article défini:  $\lambda x\tau(x) : (e \rightarrow t) \rightarrow e$  où  $\tau$  est une constante de type  $(e \rightarrow t) \rightarrow t$ .*

*Article indéfini:  $\lambda PQ((\exists x)P(x) \wedge Q(x)) : (e \rightarrow t) \rightarrow (e \rightarrow t) \rightarrow t$*

*Les étapes de la construction sémantique suivent les règles de la grammaire:*

*$S \rightarrow SNSV$  appliquer la sémantique de  $SV$  à celle de  $SN$ .*

*$SV \rightarrow VSN$  composer la sémantique de  $SN$  avec celle de  $SN : \lambda z(sn(v(z)))$ .*

*Pour une relative: faire un « et » entre la sémantique du nom et celle de la relative dont il est le sujet, c'est-à-dire appliquer  $\lambda PQ.\lambda x(P(x) \wedge Q(x))$  à la sémantique du nom et à celle de la relative, toute deux de type  $e \rightarrow t$ .*

*On introduira formellement les constantes nécessaires et on ne se souciera pas trop du nom des variables.*