vers les accumulateurs et les structures de différences

On veut modéliser les composants d'un vélo et récupérer une liste de composants (Exemple de Clocksin, Mellish).

```
piece_simple(jante).
piece_simple(rayon).
piece_simple(cadre_arriere).
piece_simple(poignees).
piece_simple(engrenages).
piece_simple(boulot).
piece_simple(ecrou).
piece_simple(fourche).
assemblage(velo, [roue, roue, cadre]).
assemblage(roue, [rayon, jante, moyeu]).
assemblage(cadre, [cadre_arriere, cadre_devant]).
assemblage(cadre_devant, [fourche, poignees]).
assemblage(moyeu, [engrenages, essieu]).
assemblage(essieu, [boulot, ecrou]).
```

Example de Clocksin, Mellish

```
/*  parts_de(+X,-Y) */
parts_de(X,[X]) :- piece_simple(X).
parts_de(X,Pieces) :-
     assemblage(X,LP),
     liste_de_pieces(LP,Pieces).

/* liste_de_pieces(+L,-L1) */
liste_de_pieces([],[]).
liste_de_pieces([X|L],Res) :-
     parts_de(X,L1),
     liste_de_pieces(L,L2),
     append(L1,L2,Res).
```

Accumulateurs

Souvent on veut traverser une structure de donnée de Prolog et calculer un résultat qui dépend ce qu'il y a dans la structure. Un peut garder le résultat intermédiaire dans un accumulateur (cf. récursion terminale en programmation fonctionnelle).

```
/* longueur d'une liste sans accumulateur */
longueur1([],0).
longueur1([_|L],R) :- longueur1(L,R1), R is R1+1.

/* longueur d'une liste avec accumulateur */
longueur2(L,Res) :- longueur2_aux(L,0,Res).

longueur2_aux([],Res,Res).
longueur2_aux([_|L],Val,Res) :-
            Val1 is Val+1,
            longueur2_aux(L,Val1,Res).
```

Accumulateurs

Pour l'exemple du vélo:

```
parts_de_2(X,L) :- parts_acc(X,[],L).
parts_acc(X,A,[X|A]) :- piece_simple(X).
parts_acc(X,A,Pieces) :- assemblage(X,LP),
                         liste_de_pieces_acc(LP,A,Pieces).

liste_de_pieces_acc([], A, A).
liste_de_pieces_acc([P|L], A, Totale) :-
      parts_acc(P,A,L1), liste_de_pieces_acc(L,L1,Totale).
```

Le but `parts_de(velo,L)` renvoie

```
L = [rayon, jante, engrenages, boulot, ecrou, rayon, jante,
engrenages, boulot, ecrou, cadre_arriere, fourche, poignees]
```

mais `parts_de_2(velo,L)` renvoie la liste renversée

```
L = [poignees, fourche, cadre_arriere, ecrou, boulot,
engrenages, jante, rayon, ecrou, boulot, engrenages, jante, rayon]
```

Pourquoi ?

Structure de difference

On utilise `[a,b,c|X]` et `X` pour pouvoir plus tard ajouter quelque chose à la liste commençant avec `a,b,c`. On garde un trou dans la liste. Par exemple,

```
 [eclipse]:Res=[a,b,c|X], p(X,NouveauTrou), NouveauTrou=[z].
```

et `p([d|NouveauTrou],NouveauTrou)`

ou

`p(Trou,Trou).`

```
parts_de_3(X,L) :- parts_acc_3(X,L,Trou), Trou=[].
parts_acc_3(X,[X|Trou],Trou) :-  piece_simple(X).

parts_acc_3(X, P, Trou) :-
    assemblage(X,LP), liste_de_pieces_acc_3(LP, P, Trou).
liste_de_pieces_acc_3([], Trou, Trou).
liste_de_pieces_acc_3([P|L], Total, Trou) :-
    parts_acc_3(P,Total,Trou2),
    liste_de_pieces_acc_3(L,Trou2,Trou).
```

Liste de difference

On peut représenter une liste de difference avec le symbole de fonction (non-interprété) `-`. Par exemple, `[1,2,3|X]-X`.
Avec les listes de différences, plus besoin de parcourir la liste pour concatener !
Avec la définition

```
concat(X-Y,Y-Z,X-Z).
```

on peut faire

```
[eclipse]:  concat([1,2,3|Y]-Y,[4,5,6|Z]-Z,L).
Y = [4, 5, 6|Z]
Z = Z
L = [1, 2, 3, 4, 5, 6|Z] - Z
```

Exemple d'une grammaire hors-contexte (algébrique) en format PROLOG (DCG)

Exemple de Pierre Nugues.

```
phr --> gn, gv.
gn --> art, n.
gn --> art, adj, n.
gv --> v.
gv --> v, gn.
art --> [le].
art --> [la].
n --> [herisson].
n --> [chenille].
adj --> [beau].
adj --> [belle].
v --> [mange].
v --> [court].
```

La traduction naïve en PROLOG

```
phr(L) :- gn(L1), gv(L2), append(L1,L2,L).
gn(L) :- art(L1), n(L2), append(L1,L2,L).
gn(L) :- art(L1), adj(L2), n(L4), append(L1,L2,L3),
         append(L3,L4,L).
gv(L) :- v(L).
gv(L) :- v(L1), gn(L2), append(L1,L2,L).
art([le]).  art([la]).
n([herisson]).
n([chenille]).
adj([beau]).
adj([belle]).
v([mange]).
v([court]).
```

Exemple de buts: `phr([le herrison,court]).` ou `phr(L).`

Une meilleure traduction en PROLOG

```
phr(A, B) :- gn(A, C), gv(C, B).

gn(A, B) :- art(A, C), n(C, B).
gn(A, B) :- art(A, C), adj(C, D), n(D, B).

gv(A, B) :- v(A, B).
gv(A, B) :- v(A, C), gn(C, B).

adj([beau|A], A).
adj([belle|A], A).
v([mange|A], A).
v([court|A], A).
art([le|A], A).
art([la|A], A).
n([herisson|A], A).
n([chenille|A], A).
```

Utilisation

Il y a deux prédicats prédéfinies `phrase/2` et `phrase/3` pour faire l'analyse syntaxique. Exemple:

```
[eclipse]:  phrase(phr,[la,chenille,mange]).
Yes
[eclipse]:  phrase(gv,L).
L = [mange] ;
L = [court] ;
L = [mange, le, herisson] ;
L = [mange, le, chenille] ;
L = [mange, la, herisson] ;  etc.
```

Un peut ajouter un troisième argument pour le reste de la phrase. Exemple:

```
[eclipse]:  phrase(gn,[la, chenille, court],L).
L = [court]
[eclipse]:  phrase(gv,[ mange, la, belle, chenille],L).
L = [la, belle, chenille] ;
L = []
```

Extension

Pour accorder le genre de l'adjectif avec le nom on modifie la grammaire:

```
phr --> gn(G), gv.

gn(G) --> art(G), n(G).
gn(G) --> art(G), adj(G), n(G).

gv --> v.
gv --> v, gn(G).

art(m) --> [le].
art(f) --> [la].
n(m) --> [herisson].
n(f) --> [chenille].
adj(m) --> [beau].
adj(f) --> [belle].
v --> [mange].
v --> [court].
```

La traduction en PROLOG

```
phr(A, B) :- gn(C, A, D), gv(D, B).

gn(A, B, C) :- art(A, B, D), n(A, D, C).
gn(A, B, C) :- art(A, B, D), adj(A, D, E), n(A, E, C).

gv(A, B) :- v(A, B).
gv(A, B) :- v(A, C), gn(D, C, B).

n(m, [herisson|A], A).
n(f, [chenille|A], A).
art(m, [le|A], A).
art(f, [la|A], A).
adj(m, [beau|A], A).
adj(f, [belle|A], A).
v([mange|A], A).
v([court|A], A).
```

Extension

On peut aussi mettre des terminaux directement dans les règles. Par exemple:

```
phr --> gn, gv.

gn --> [le], n.

gv --> v.

v --> [court].

n --> [chien].
```

La traduction en PROLOG

```
phr(A, B) :- gn(A, C), gv(C, B).

gv(A, B) :-  v(A, B).

gn(A, B) :-  'C'(A, le, C), n(C, B).

phr(A, B) :- gn(A, C), gv(C, B).

v([court|A], A).

n([chien|A], A).
```

'C' est un prédicat prédéfinie par
`'C'([X|L],X,L).`

Structure syntaxique

On peut obtenir la structure syntaxique de la phrase reconnu comme suit:

```
phr(phr(GN,GV)) --> gn(GN), gv(GV).

gn(gn(ART,N)) --> art(ART), n(N).

gv(gv(V,GN)) --> v(V), gn(GN).

art(art(le)) --> [le].
art(art(la)) --> [la].

n(n(herisson)) --> [herisson].
n(n(chenille)) --> [chenille].

v(v(mange)) --> [mange].
```

La traduction en PROLOG

```
phr(phr(A, B), C, D) :- gn(A, C, E), gv(B, E, D).

gn(gn(A, B), C, D) :- art(A, C, E), n(B, E, D).

gv(gv(A, B), C, D) :- v(A, C, E), gn(B, E, D).

art(art(le), [le|A], A).
art(art(la), [la|A], A).

n(n(herisson), [herisson|A], A).
n(n(chenille), [chenille|A], A).

v(v(mange), [mange|A], A).
```

Le but `phrase(phr(S),L).` donne toutes les phrases avec leur structure.

Partiel 2016, exercice 1

Pour chacune des requêtes suivantes, donner le résultat renvoyé par l'interpréteur Prolog (sans justifier) :
Dans les questions suivantes, on suppose que le fichier `ex.pl` ci-dessous ait été compilé:

```
/********* ex.pl *********/
a(0,1).
b(X,1):-X=0.
c(X,1):-X=:=0.
d(X,1):-X==0.
/*************************/
```

1. `a(1-1,Y).`
2. `b(0,Y).`
3. `c(1-1,Y).`
4. `d(1-1,Y).`

Partiel 2014, exercice 1

Pour chacune des requêtes suivantes, donner le résultat renvoyé par l'interpréteur Prolog (sans justifier) :
Dans les questions suivantes, on suppose que le fichier `ex.pl` ci-dessous ait été compilé:

```
/* fichier fact.pl */
fact1(0,1).
fact1(N,R) :- N>0, P is N-1, fact1(P,V), R is N*V.
fact2(X,1) :- X=:=0.
fact2(N,R) :- N>0, P is N-1, fact2(P,V), R is N*V.
/* fin de fact.pl */
```

1. `fact1(3-3,R).`
2. `fact2(3-3,R).`
3. `fact1(7-3,R).`
4. `fact2(7-3,R).`

Partiel 2008, Exercice 2

On considère les trois prédicats suivants :

```
if_1(T,P,Q):-T,!,P.
if_1(T,P,Q):-Q.
if_2(T,P,Q):-T,P,!.
if_2(T,P,Q):-Q.
if_3(T,P,Q):-!,T,P.
if_3(T,P,Q):-Q.
```

1. Pour chaque paire `j,k`, ($1 \leq$ `j` $<$ `k` ≤ 3) trouver des arguments `T_jk,P_jk,Q_jk` tels que les buts `if_j(T_jk,P_jk,Q_jk)` et `if_k(T_jk,P_jk,Q_jk)` donnent des résultats différents. Il s'agit d'exhiber trois triplets: `T_12,P_12,Q_12`, puis `T_13,P_13,Q_13` et finalement `T_23,P_23,Q_23`.
2. Dessiner les arbres de dérivation de deux de ces buts, pour une paire `j,k` au choix.
3. Lequel de ces prédicats implémente correctement le `if-then-else`? (motiver brièvement).