PROLOG: DCG

Definite Clause Grammar

Michel RUEHER

DCG - Concepts

- **Prolog** a été conçu pour ***l'analyse du langage naturel***

- Formalisme des clauses de Horn est très proche de celui des ***grammaires «context free»***

Rappel :

Règle *«context free»* : $X \rightarrow \alpha$ où

X : symbole non terminal

α : est une chaîne de terminaux ou de non-terminaux

Si X peut être remplacé par α sans tenir compte du contexte où il apparaît

DCG : sucre ***syntaxique***

- **DCG** : sucre ***syntaxique***
  → Traduction *automatique* en Prolog
- **Exemple 1:**

DCG : groupe_nominal --> det.
det --> [le].
det --> [un].

Prolog généré:

```
groupe_nominal(A, B) :- det(A, B).
det([le|A], A).
det([un|A], A).
```

Les arguments des prédicats comme **([le|A],A)** sont des ***«listes de différence »***

Listes de différence

- Une liste **L** est représentée par la paire: **($[E_1,E_2,\ldots,E_n|X]$,X)**
- Avantage des listes de différence: « pointeur » sur la fin de la liste :
  → la concaténation de **(S1,S2)** et **(S2,S3)** est **(S1,S3)**

Exemple:

Programme : p([X1,X2],[X2,X3],[X1,X3]).

```
|?- p([[a,b|X],X],[[c|X3],X3],[Z,X3]).
X = [c|X3]]
Z = [a,b,c|X3]
```

```
?- p([[a,b|X],X],[[c|Y],Y],[Z,[]]).
Z = [a,b,c]
X = [c]
```

Autre notation : **p([X-X1],[X1-X2],[X-X2]).**

Et ?- p([[a,b|X]-X],[[c|Y]-Y],[Z-[]]).

Complexité: l'unification n'est plus dépendante de la longueur de liste

Exemple 1 (suite)

DCG:

```
phrases --> groupe_nominal,  groupe_verbal.

groupe_nominal --> det, nom.

groupe_verbal  --> verbe, adjectif.

det --> [le].

det --> [un].

nom--> [plat].

adjectif--> [rouge].

verbe --> [est].
```

Exemple 1 (suite)

DCG:

```
phrases --> groupe_nominal,  groupe_verbal.
groupe_verbal  --> verbe, adjectif.
det --> [le].
nom--> [plat].
adjectif--> [rouge].
groupe_nominal --> det, nom.
det --> [un].
verbe --> [est].
```

- **Analyser la phrase :**

  ?- phrases([le,plat,est,rouge],[]).

  true ?

- **Générer la phrase** :

  ?- phrases(X,[]).

  X = [le,plat,est,rouge]?

  yes

Exemple 1 (suite)

Programme Prolog généré:

```
phrases(A, B) :-
        groupe_nominal(A, C),
        groupe_verbal(C, B).

groupe_nominal(A, B) :-
        det(A, C),
        nom(C, B).

groupe_verbal(A, B) :-
        verbe(A, C),
        adjectif(C, B).

det([le|A], A).
det([un|A], A).

nom([plat|A], A).

adjectif([rouge|A], A).
verbe([est|A], A).
```

Questions:

```
| ?- phrases(X,[]).
X = [le,plat,est,rouge] ? ;
| ?- phrases([le,plat,est,rouge,et],H).
H = [et] ? ;
```

Traduction des règles de grammaire (1)

Règle de grammaire : head → body

Prend en entrée une chaîne, analyse une partie initiale, et produit la partie restante en sortie pour une analyse ultérieure.

- Arguments pour les entrées et sorties: non écrits explicitement
- ***Body*** **:** terminaux et non-terminaux (séparés par des virgules)
  Terminal : liste Prolog qui représente les éléments qu'il contient

Traduction:

```
p --> q.          se traduit en:      p(S0, S) :- q(S0, S).
                % S0 : input string  et  S : output string

p(X, Y) -->              se traduit en        p(X, Y, S0, S) :-
    q(X),                                         q(X, S0, S1),
    r(X, Y),                                      r(X, Y, S1, S2),
    s(Y).                                         s(Y, S2, S).
```

Traduction des règles de grammaire (2)

- **Symbole non terminal**: tout terme Prolog (autre qu'une variable ou un nombre)
- **Un symbole terminal**: tout terme Prolog
- Séquence de symboles terminaux : liste Prolog (séquence vide: liste vide [] ou "")
- **Conditions supplémentaires**: appels Prolog inclus dans le côté droit, entre accolades {}
- Côté gauche d'une règle : un non-terminal, éventuellement suivi d'une séquence de terminaux
- Le symbole **!** peut être inclus dans le côté droit comme dans une clause Prolog (pas besoin d'être mis entre accolades {})

Traduction des règles de grammaire (3)

phrases --> groupe_nominal, groupe_verbal.

se traduit en

```
phrases(A, B) :-
    groupe_nominal(A, C),
    groupe_verbal(C, B).
```

p --> [go,to], q, [stop].

se traduit en

```
p([go, to|A], B) :-
        q(A, C),
        C = [stop|B].
```

Grammaires non context-free

- Les DCG sans arguments supplémentaires : ***grammaires context-free*** (un seul argument à gauche)
- Ajout d'arguments supplémentaires: ***grammaires context-sensitives***

Exemple:

`s --> a(N),b(N).`	`s(A, B) :- a(C, A, D), b(C, D, B).`
`a(0) --> [].`	`a(0, A, A).`
`a(M) --> [a], a(N), {M is N + 1}.`	`a(M, [a|B], C) :- a(D, B, E), M is D + 1, E=C.`
`b(0) --> [].`	`b(0, A, A).`
`b(M) --> [b], b(N), {M is N + 1}.`	`b(M, [b|B], C) :- b(D, B, E), M is D + 1, E=C.`

```
| ?- s([a,a,b,b],X).

X = [a,a,b,b] ? ;

X = [] ? no
```

Ajout d'un motif – calcul de l'arbre d'analyse

```
phrases(ph(N,V))                --> groupe_nominal(N) , groupe_verbal(V).
groupe_nominal(gn(PA,N))        --> prep_article(PA), groupe_nominal2(N).
groupe_nominal(gn(N))           --> groupe_nominal2(N).
groupe_nominal2(gn2(A,N))       --> adjectif(A), groupe_nominal2(N).
groupe_nominal2(gn2(N))         --> nom(N).
groupe_verbal(gv(V))            --> verbe(V).
groupe_verbal(gv(V,N))          --> verbe(V), groupe_nominal(N).
prep_article(pa(le))            --> [le].
prep_article(pa(un))            --> [un].
nom(nom(os))                    --> [os].
nom(nom(plat))                  --> [plat].
verbe(verbe(contient))          --> [contient].
adjectif(adjectif(beau))        --> [beau].
```

Ajout d'un motif – calcul de l'arbre d'analyse

```
phrases(ph(A, B), C, D) :-
        groupe_nominal(A, C, E),
        groupe_verbal(B, E, D).

groupe_nominal(gn(A, B), C, D) :-
        prep_article(A, C, E),
        groupe_nominal2(B, E, D).
...

prep_article(pa(le), [le|A], A).
prep_article(pa(un), [un|A], A).
nom(nom(os), [os|A], A).
nom(nom(plat), [plat|A], A).
...
```

| ?- phrases(A,[le,beau,plat,contient,un,os],[]).

A=ph(gn(pa(le),gn2(adjectif(beau),gn2(nom(plat)))),gv(verbe(contient),gn(pa(un),gn2(nom(os))))) ?

Ajout d'un motif – Accord du sujet et de l'objet

```
phrases(Nb)       --> groupe_nominal(Nb) , groupe_verbal(Nb).
groupe_nominal(Nb)  -->  prep_article(Nb), groupe_nominal2(Nb).
groupe_nominal(Nb)        -->  groupe_nominal2(Nb).
...
prep_article(sing)      -->  [le].
prep_article(plur)      -->  [des].
prep_article(sing)      -->  [un].
nom(_)              -->  [os].
nom(sing)                -->  [plat].
nom(plur)                -->  [plats].
verbe(sing)         -->  [contient].
verbe(plur)   -->        [contiennnent].
adjectif(sing)      -->  [beau].
adjectif(plur)      -->  [beaux].

| ?-   phrases(A,[le,beau,plat,contient,un,os],[]).

A = sing ?
```

DCG – Exemples : analyse d'expressions arithmétiques

```
expr --> term, "+", expr.
expr --> term, "-", expr.
expr --> term.

term --> number, "*", term.
term --> number, "/", term.
term --> number.

number --> "+", number.
number --> "-", number.
number --> [C], {"0"=<C, C=<"9"}. % C is the character code of some digit.
```

?- expr("-2+3*5+1", []. **->** true

?- expr("-2+3*5+1", T).

T = [] ? ;

T = [43,49] ? ; name(L,[43,49]). -> L = '+1'

T = [42,53,43,49] ? ; name(L,[42,53,43,49]). -> L = '*5+1'

DCG – Exemples : analyse et évaluation d'expressions arithmétiques (1)

```
expr(Z) --> term(X), "+", expr(Y), {Z is X + Y}.
expr(Z) --> term(X), "-", expr(Y), {Z is X - Y}.
expr(X) --> term(X).

term(Z) --> number(X), "*", term(Y), {Z is X * Y}.
term(Z) --> number(X), "/", term(Y), {Z is X / Y}.
term(Z) --> number(Z).

number(C) --> "+", number(C).
number(C) --> "-", number(X), {C is -X}.
number(X) --> [C], {"0"=<C, C=<"9", X is C - "0"}.
                        % C is the character code of some digit.
```

?- expr(Z, "-2+3*5+1", []).

***Z*=14.**

DCG –Exemples : Arbres

Règle de grammaire :

Tree ::= node(Tree,Tree) | leaf(Data)

DCG :

```
tree(leaf(Data)) --> [Data].
tree(node(L,R)) --> tree(L), tree(R).
```

Prolog Program:

```
tree(leaf(A), [A|B], B).
tree(node(A, B), C, D) :-    tree(A, C, E), tree(B, E, D).
```

Question:

?- T= node(node(leaf(a),leaf(b)),leaf(c)), tree(T,A,B).

```
A = [a,b,c|B]
```

Remarque: tree/3 génère une liste(‘’reverse’’ du parsing)

DCG –Exemples : listes

Règle de grammaire : List ::= cons(E,List) | []

DCG :

```
literal([])--> [].

literal([X|Xs])--> [X], literal(Xs).
```

Prolog Program:

```
literal([], A, A).

literal([A|B], [A|C], D) :- literal(B, C, D).
```

Questions:

?- literal(A, [a,b,c], D).

A = [], D = [a,b,c] ? ;
A = [a], D = [b,c] ? ;
...
A = [a,b,c], no

?- literal([a,b], C, [c,d,e]).

C = [a,b,c,d,e] ? ;