INFf3
Programmation logique

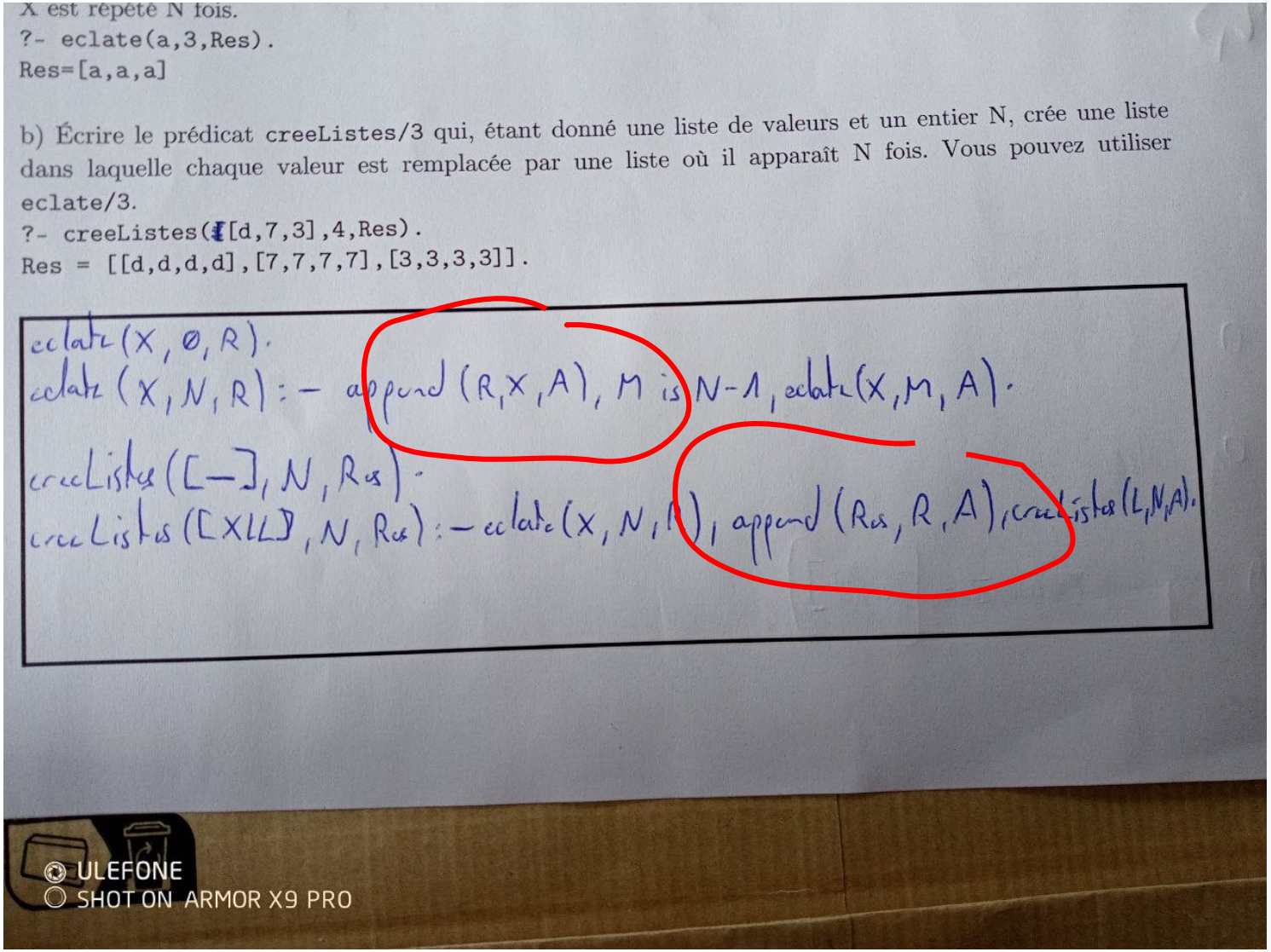
Cours 5 : Arbres et grammaires

Benoît Lemaire

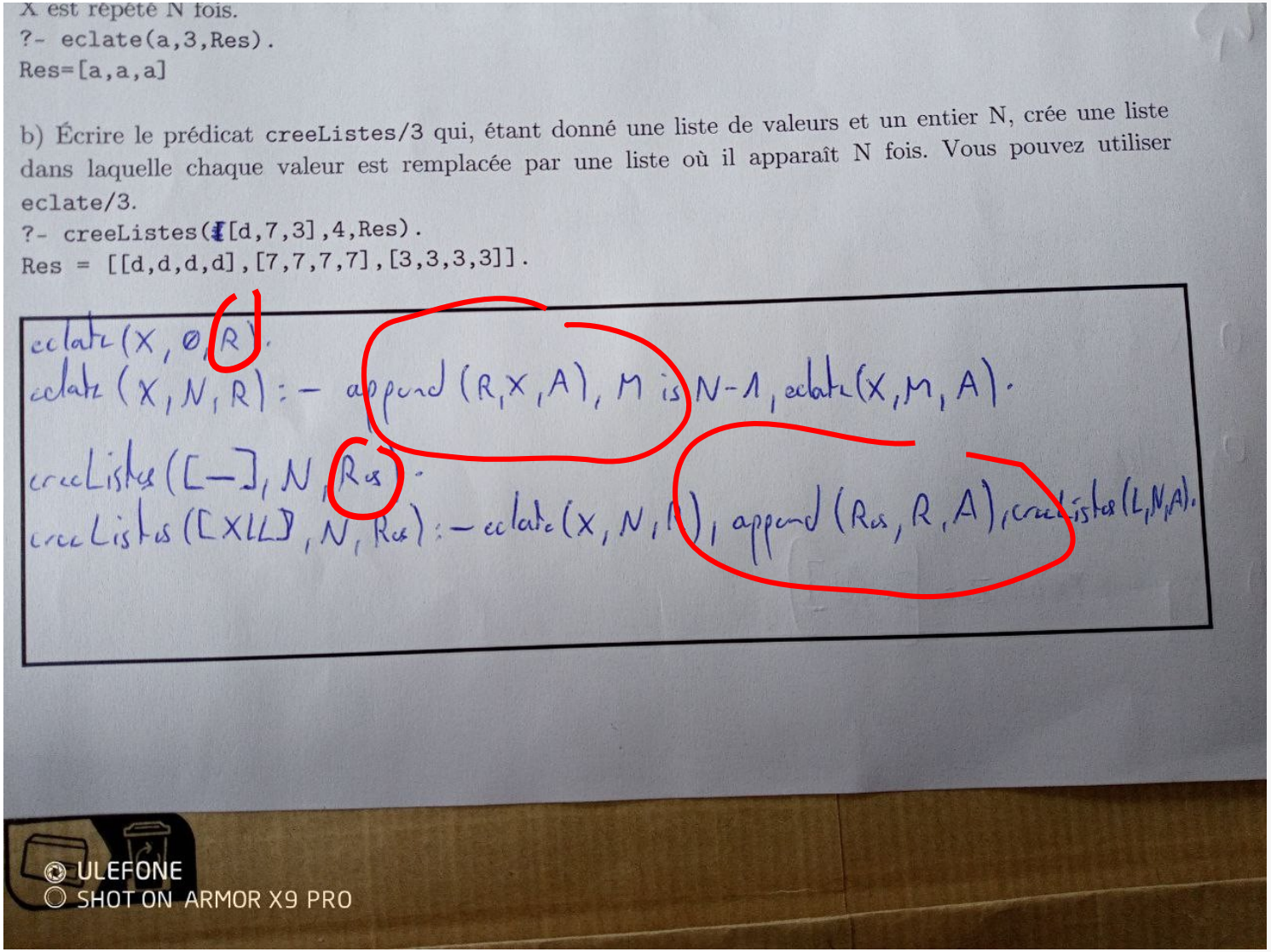
Université Grenoble Alpes
L2 - MIASHS
Grenoble – France

Ces diapos se sont inspirées de celles créées par Jean-Michel Adam

Erreurs classiques



Erreurs classiques



Erreurs classiques

Exercice 5 (4 points = 2+2, environ 12 minutes)

a) Écrire le prédicat `eclate/3` qui étant donné une valeur X et un nombre N crée une liste dans laquelle X est répété N fois.

```
?- eclate(a,3,Res).
Res=[a,a,a]
```

b) Écrire le prédicat `creeListes/3` qui, étant donné une liste de valeurs et un entier N, crée une liste dans laquelle chaque valeur est remplacée par une liste où il apparaît N fois. Vous pouvez utiliser `eclate/3`.

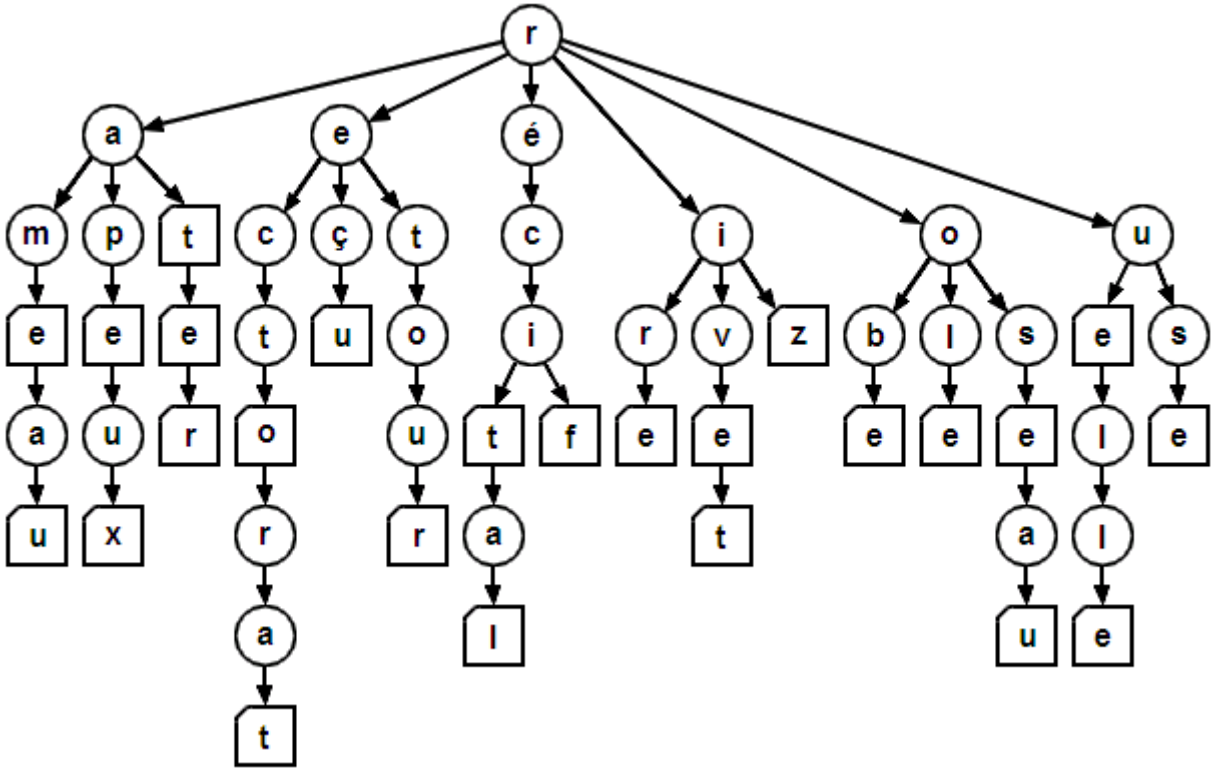
```
?- creeListes([[d,7,3],4,Res).
Res = [[d,d,d,d],[7,7,7,7],[3,3,3,3]].
```

a) eclate(_,0,R).
eclate(X,N,R):- N1 is N-1, eclate(X,N1,[X|R]).

b) creeListes([_|L],0,R).
creeListes([X|L],N,R):- N1 is N-1, eclate(X,N,R), creeListes(L,N1,R).
creeListes(L,N1,[X|R]).

Arbres

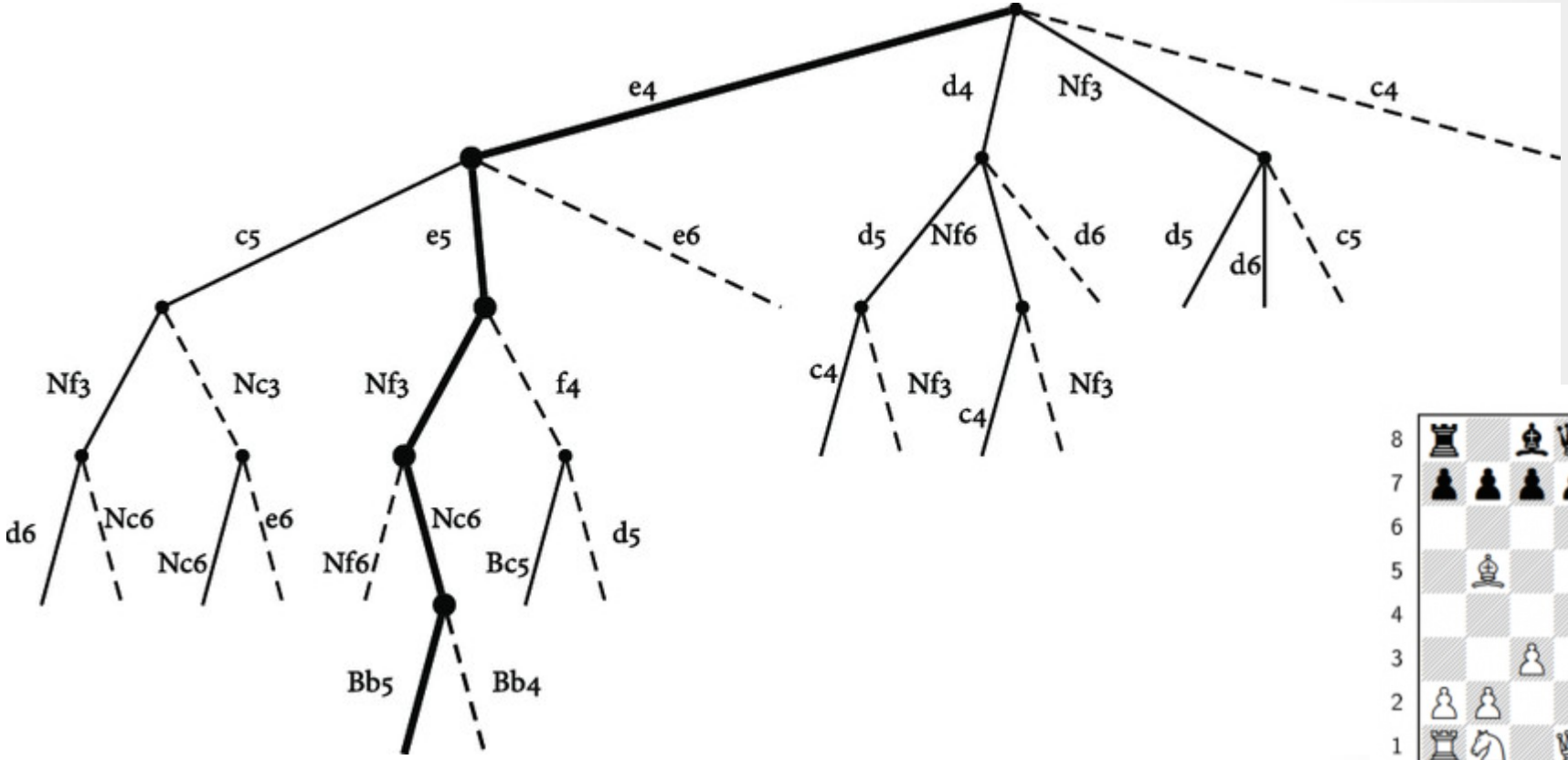
- Représentation très utilisée en informatique
- Exemple : arbre lexicographique



https://algo.developpez.com/faq/?page=Structures-arborescentes

Arbres

Exemple : arbre des coups possibles aux échecs

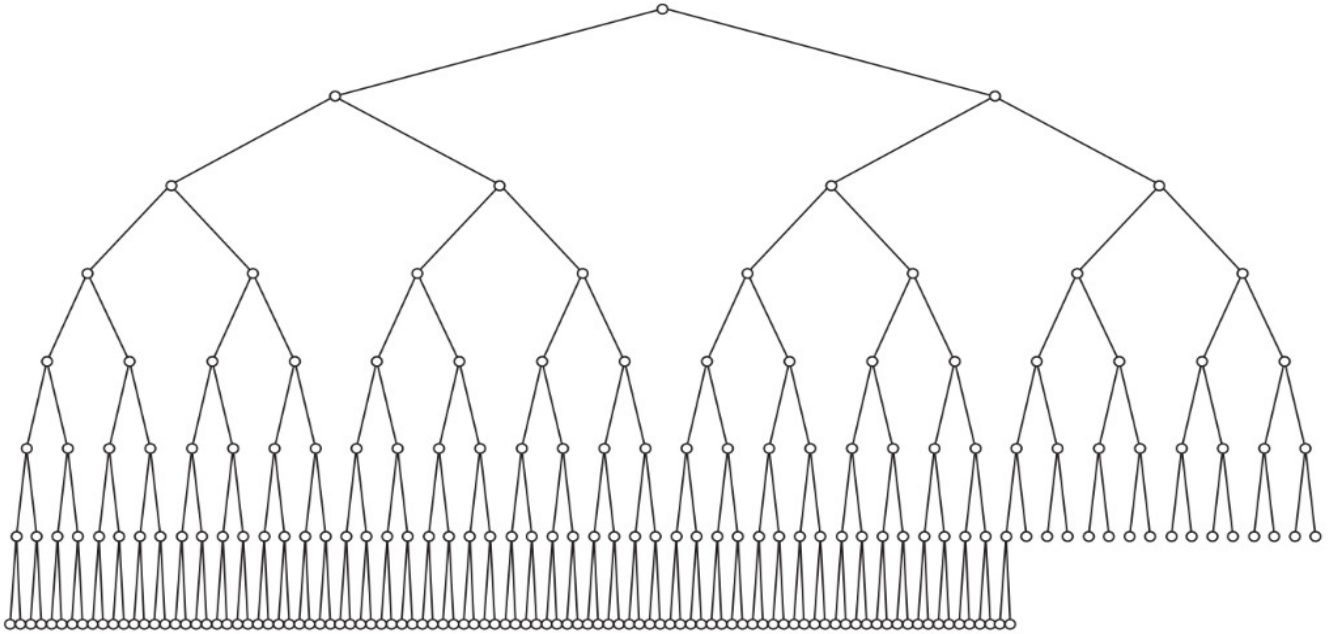


(Franchi, 2011)

https://fr.wikipedia.org/wiki/Arbre_binaire_de_reche

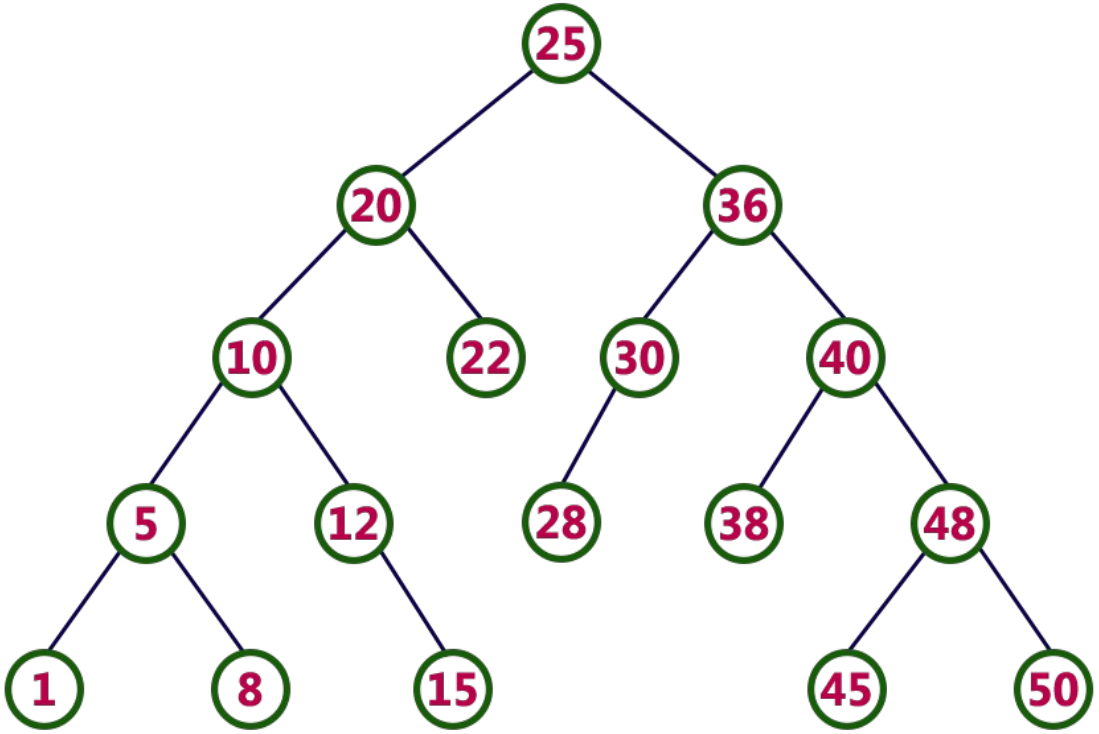
Arbres

- Accès très rapide aux feuilles
- Un arbre binaire équilibré contenant N nœuds a une hauteur (distance racine-feuilles) d'environ $\log_2(N)$.

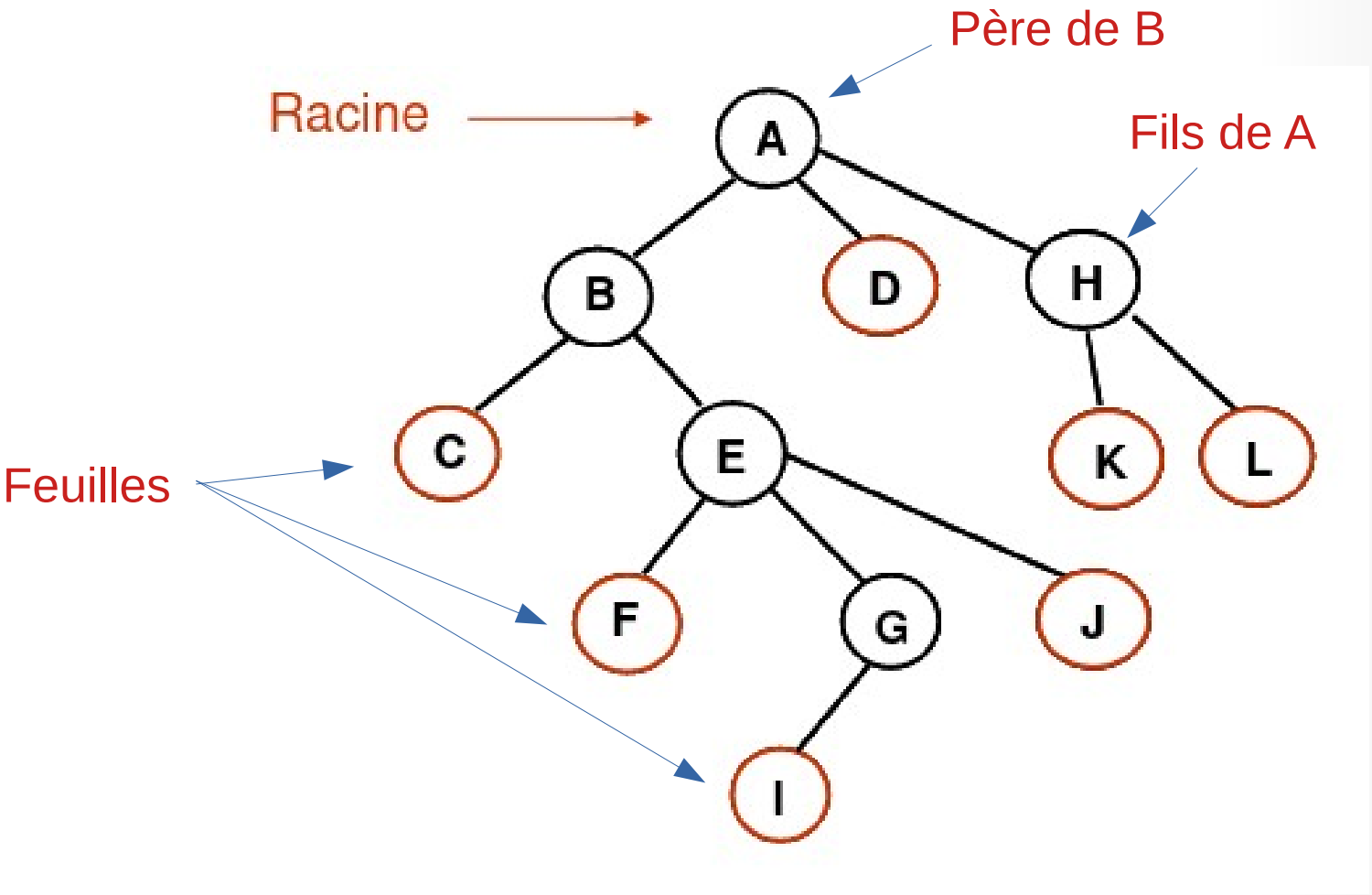


Applications

- Arbre binaire de recherche : tous les nœuds inférieurs à la racine sont dans le sous-arbre gauche ; tous les nœuds supérieurs sont dans le sous-arbre droit

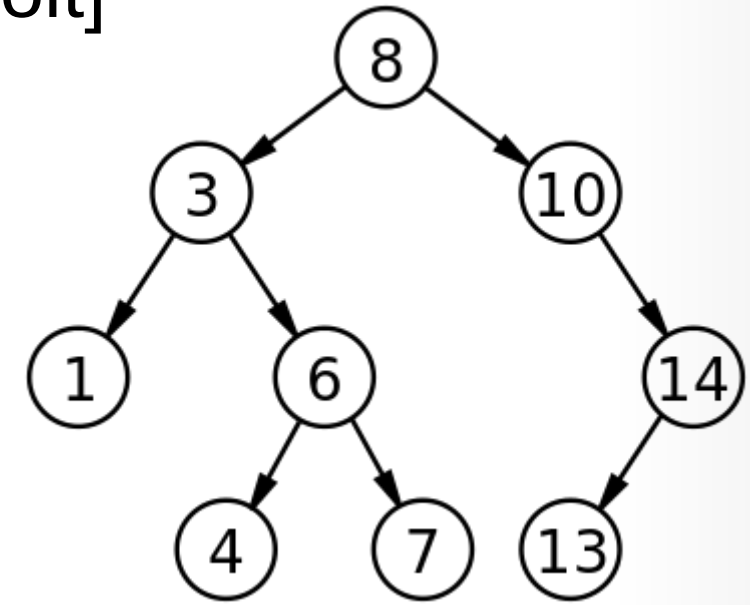


Arbres - terminologie



En Prolog

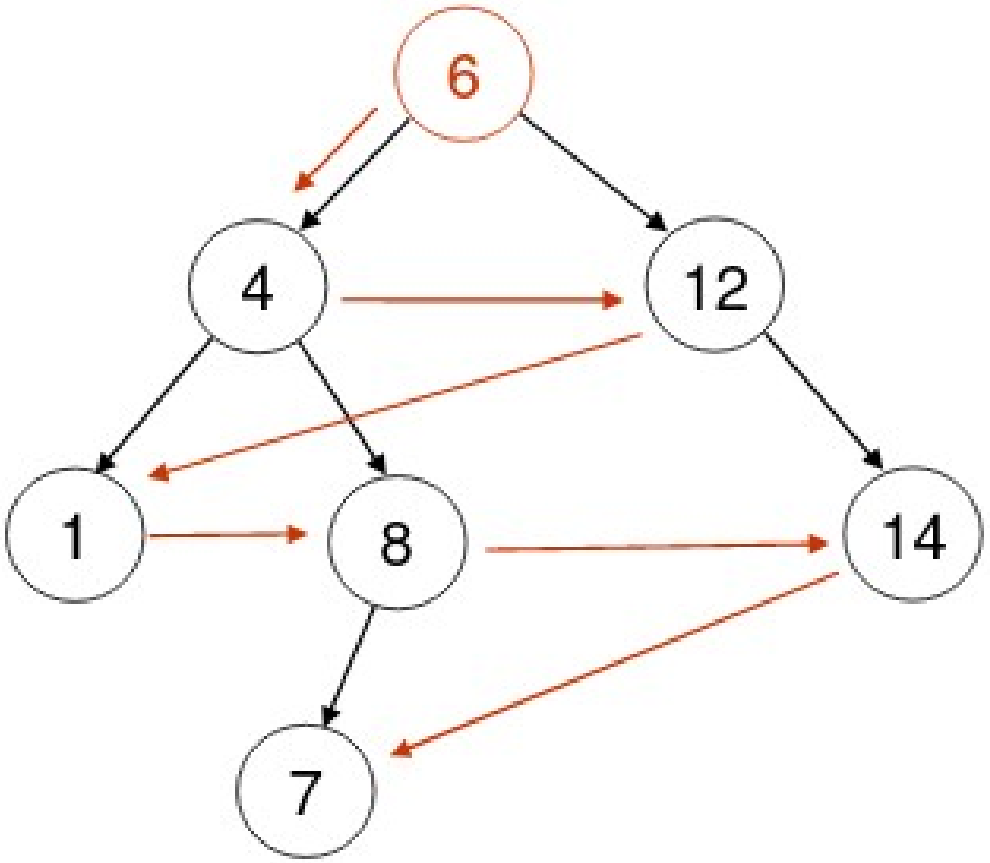
- Un arbre binaire est une liste [racine, sous-arbre gauche, sous-arbre droit]




```
= [8,[3,[1,[],[]],[6,[4,[],[]],[7,[],[]]]],
     [10,[],[14,[13,[],[]],[]]]]
```

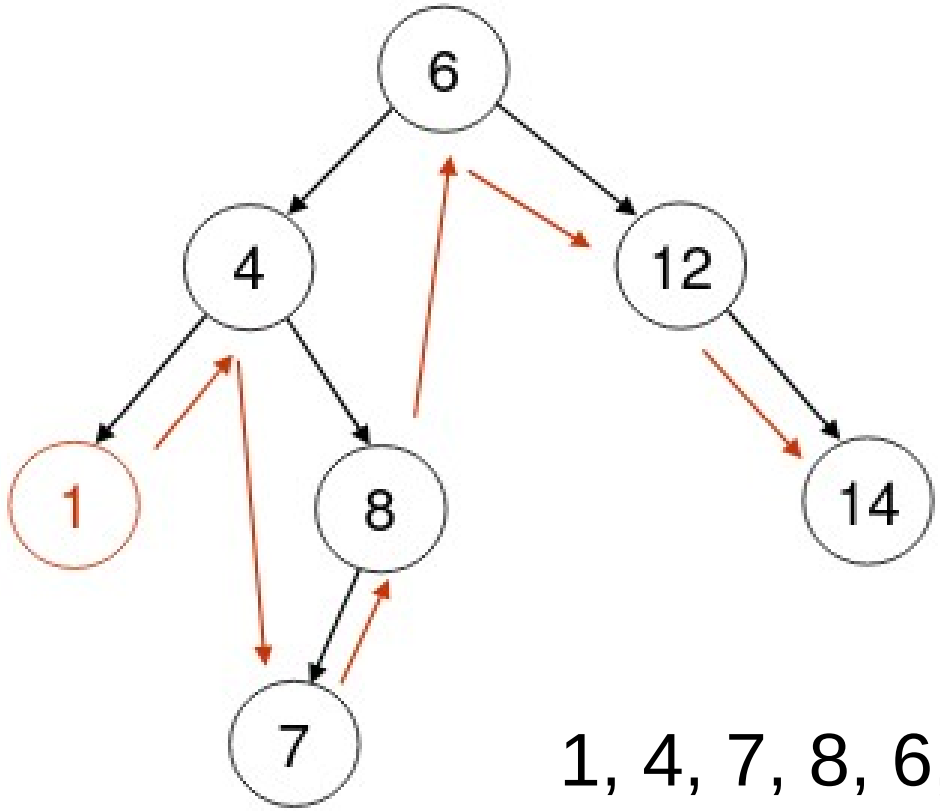
Parcours en largeur

- Noeuds de niveau 0 (racine), puis nœuds de niveau 1, puis nœuds de niveau 2, ...



Parcours en profondeur

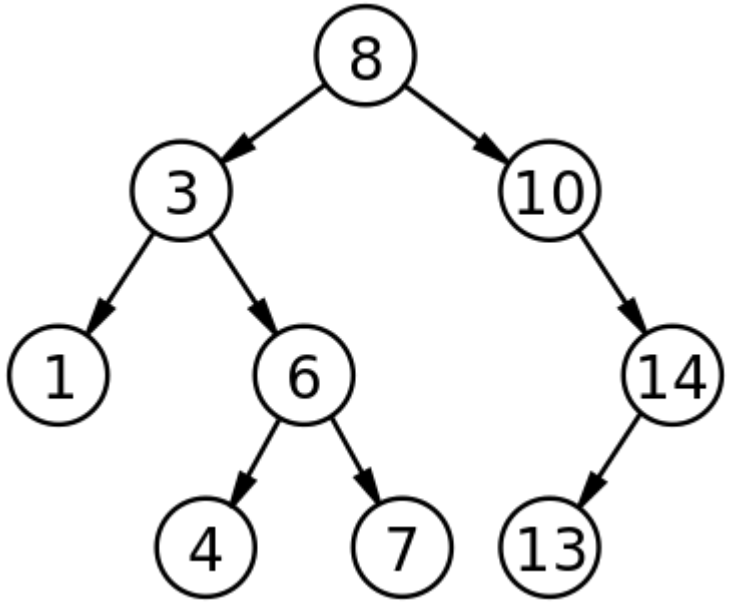
- Version infixée : Sous-arbre gauche, puis racine, puis sous-arbre droit.



1, 4, 7, 8, 6, 12, 14

Parcours en profondeur

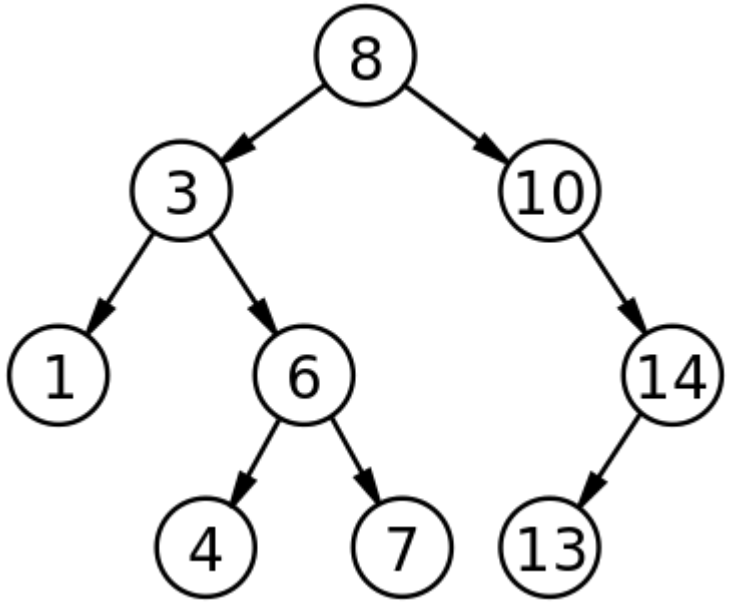
- Version préfixée: racine, puis sous-arbre gauche, puis sous-arbre droit.



8, 3, 1, 6, 4, 7, 10, 14, 13

Parcours en profondeur

- Version postfixée: sous-arbre gauche, puis sous-arbre droit puis racine.

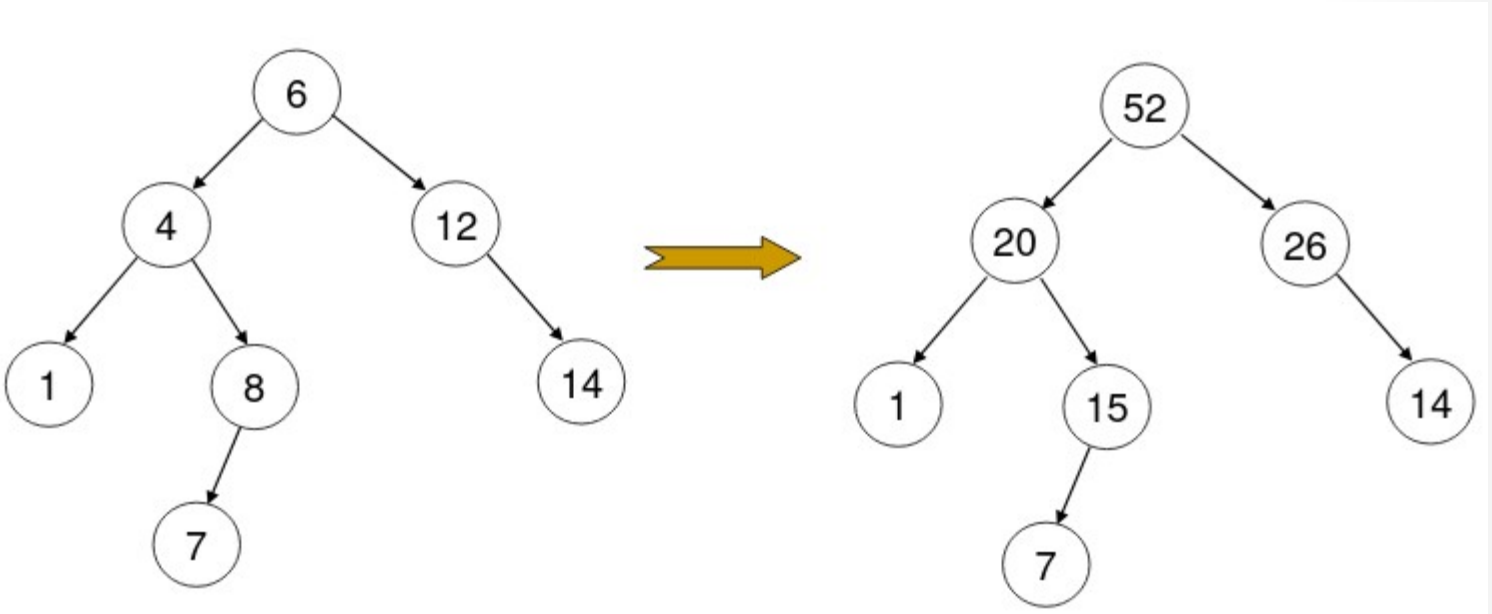


1, 4, 7, 6, 3, 13, 14, 10, 8

Parcours en profondeur en Prolog

```
parcoursPrefixe([]).

parcoursPrefixe([Racine,G,D]):- writef("%t ",[Racine]),
parcoursPrefixe(G),parcoursPrefixe(D).


parcoursInfixe([]).

parcoursInfixe([Racine,G,D]):- parcoursInfixe(G), writef("%t ",
[Racine]), parcoursInfixe(D).



parcoursPostfixe([]).

parcoursPostfixe([Racine,G,D]):- parcoursPostfixe(G),
parcoursPostfixe(D),writef("%t ",[Racine]).
```

Exemple : Étant donné un arbre, créer un autre arbre dont les nœuds valent la somme des valeurs des sous-arbres



Exemple : Étant donné un arbre, créer un autre arbre dont les nœuds contiennent la somme des valeurs des sous-arbres

```
creer_som([],[]).

creer_som([N,[],[]], [N,[],[]]).

creer_som([N,G,[]], [NR, [NG,FG,FD], []]):- creer_som(G,[NG,FG,FD]),
NR is N+NG.

creer_som([N,[],D], [NR, [], [ND,FG,FD]]):- creer_som(D,[ND,FG,FD]),
NR is N+ND

creer_som([N,G,D], [NR, [NG,FG1,FD1], [ND,FG2,FD2]]):-creer_som(G,
[NG,FG1,FD1]), creer_som(D,[ND,FG2,FD2]), NR is N+ND+NG.
```

Grammaires

- Définit la syntaxe d’un langage.
- G = < VT, VN, S, R>
- VT : vocabulaire terminal, les symboles du langage
- VN : vocabulaire non terminal, les noms utilisés pour décrire les règles de construction du langage
- R : règles de la grammaire
- S : symbole de départ

Exemple

S	→	SN SV
SN	→	NP
SV	→	VT SN
NP	→	Marie | Pierre
VT	→	regarde

```
s(L)   :- sn(L1), sv(L2), append(L1, L2, L).
sn(L) :- np (L).
sv(L) :- vt (L1), sn(L2), append(L1, L2, L).
np([pierre]).
np([marie]).
vt([regarde]).
```

Exemple

```
s(L)   :- sn(L1), sv(L2), append(L1, L2, L).
sn(L) :- np (L).
sv(L) :- vt (L1), sn(L2), append(L1, L2, L).
np([pierre]).
np([marie]).
vt([regarde]).

?- s([pierre, regarde, marie]).
true

?- s(X).
X = [pierre, regarde, pierre] ;
X = [pierre, regarde, marie] ;
X = [marie, regarde, pierre] ;
X = [marie, regarde, marie].
```

Autre écriture

peut-être écrit !

```
s --> sn, sv.
sn --> np.
sv --> v, sn.
np --> [pierre].
np --> [marie].
v --> [regarde]
```

transformé par Prolog en :

```
s(L1, L2) :- sn(L1, L3), sv(L3, L2).
sn(L1, L2) :- np(L1, L2).
sv(L1, L2) :- v(L1, L3), sn(L3, L2).
np([pierre|L], L).
np([marie|L], L).
v([regarde|L], L).
```

Autre écriture

peut-être écrit :

```
s --> sn, sv.
sn --> np.
sv --> v, sn.
np --> [pierre].
np --> [marie].
v --> [regarde]
```

transformé par Prolog en :

```
s(L1, L2) :- sn(L1, L3), sv(L3, L2).
sn(L1, L2) :- np(L1, L2).
sv(L1, L2) :- v(L1, L3), sn(L3, L2).
np([pierre|L], L).
np([marie|L], L).
v([regarde|L], L).
```

Notation DCG en Prolog

- Les symboles de prédicats et de fonctions, les variables et les constantes obéissent à la syntaxe habituelle de Prolog.
- Les symboles adjacents dans une partie droite de règle sont séparés par une virgule
- La flèche est le symbole "→"
- Les terminaux sont écrits entre "[" et"]"
- La chaîne vide ε est représentée par "[]".
- On peut insérer en partie droite des règles, des buts autres que les symboles de la grammaire ; dans ce cas, ils figurent entre "{" et "}".

Exemple avec accord en genre et en nombre

```
s --> sn(_), sv.
sn(G) --> det(G), n(G), optrel.
sn(G) --> np(G).
sv --> vt, sn(_).
sv --> vi.
optrel --> [].
optrel --> [qui], sv.
det(G) --> [Mot], {lexique(Mot,det,G)}.
n(G) --> [Mot], {lexique(Mot,n,G)}.
np(G) --> [Mot], {lexique(Mot,np,G)}.
vt --> [Mot], {lexique(Mot,vt,_)}.
vi --> [Mot], {lexique(Mot,vi,_)}.
```

```
lexique(qui,conj,_).
lexique(tout,det,m).
lexique(un,det,m).
lexique(une,det,f).
lexique(etudiant,n,m).
lexique(programme,n,m).
lexique(fille,n,f).
lexique(marie,np,f).
lexique(ecrit,vt,_).
lexique(boucle,vi,_).
lexique(dort,vi,_).
```