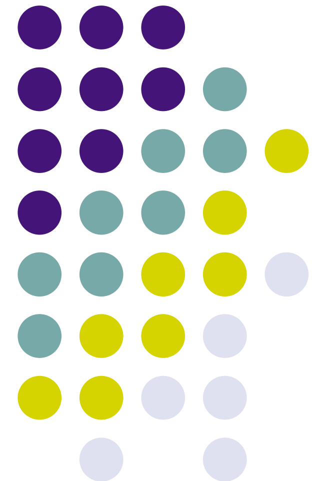
Introduction to PROLOG for NLP applications

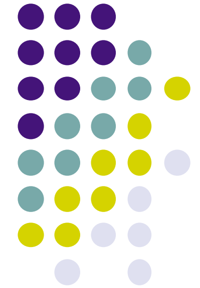
J. Savoy
Université de Neuchâtel



A. Gal, G. Lapalme, P. Saint-Dizier, H. Somers : *Prolog for natural language processing*. John Wiley & Sons, Chichester (UK).

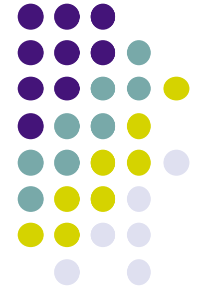
W. F. Clocksin, C. S. Mellish: Programming in Prolog. Springer Verlag, Berlin.

Prolog



- Acronym for
  Programmation Logique (Logic Programming)
  Alain Colmerauer, 1970
- Very different than other programming languages.
  A new paradigm (e.g., imperative, functional, object-oriented, parallel, etc.)
- Do not specify "how" to resolve a problem (algorithm).
  Specify what is true (facts) or
  how we can prove it is true (rules)
- Freely available SWI-Prolog (Amsterdam)

Syntax of Prolog



- Syntax

  `p :- q1, q2, q3.`

- Semantics

  `p` is true if `q1` is true, and `q2` is true, and `q3` is true.

- Head of the rule: `p`

  Body: the propositions `q1`, `q2`, `q3`

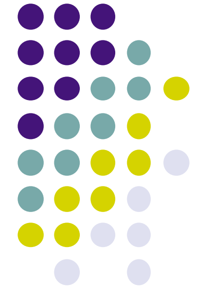
  (in this case, they are literals)

- The symbol `:-` means "if" (the head is true if …)

  the comma `,` separating the conditions (this is a "and")

  the final stop (`.`)

Predicate



- Predicate

  Function (relation) that is true/false

  Composed of a name (with / without argument(s))

  Example

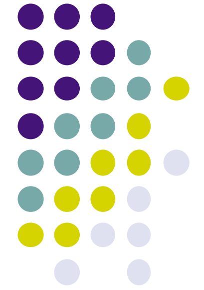
  ```
  p
  q2
  ```

  and argument(s)

  Add parenthesis and if needed, the comma

  ```
  blue(sky)
  love(mary, john)
  give(paul, mary, book)
  ```

Predicate



- Facts

  A rule that is always true (or a rule without any condition)

```
happy(john).
human(paul).
```

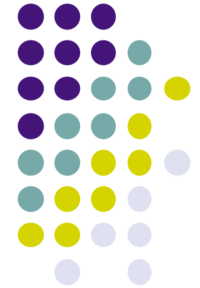
- Rules

  It is true if a given set of conditions are true

```
happy(P) :- healthy(P), wise(P), rich(P).
happy(P) :- student(P).
```

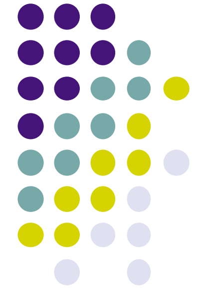
- The relation `happy` is true in two distinct cases.

Terms



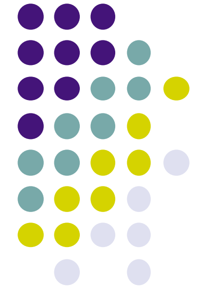
- Terms
  The name of predicate arguments are terms.
  - Atomic symbols (numbers or begin with a lowercase)
    Use to define specific object (constants)
    `mary, book, 3.15, -8, sing, …`
  - Variables (begin with an uppercase or an underscore)
    Define objects waiting to be bounded
    `X, Someone, Mary89, _C23, _9, …`
  - Compound terms (structure)
    A functor (predicate name) and its arguments
    `aime(jean, marie), happy(john), np(adj(white), nn(house)), …`

Terms



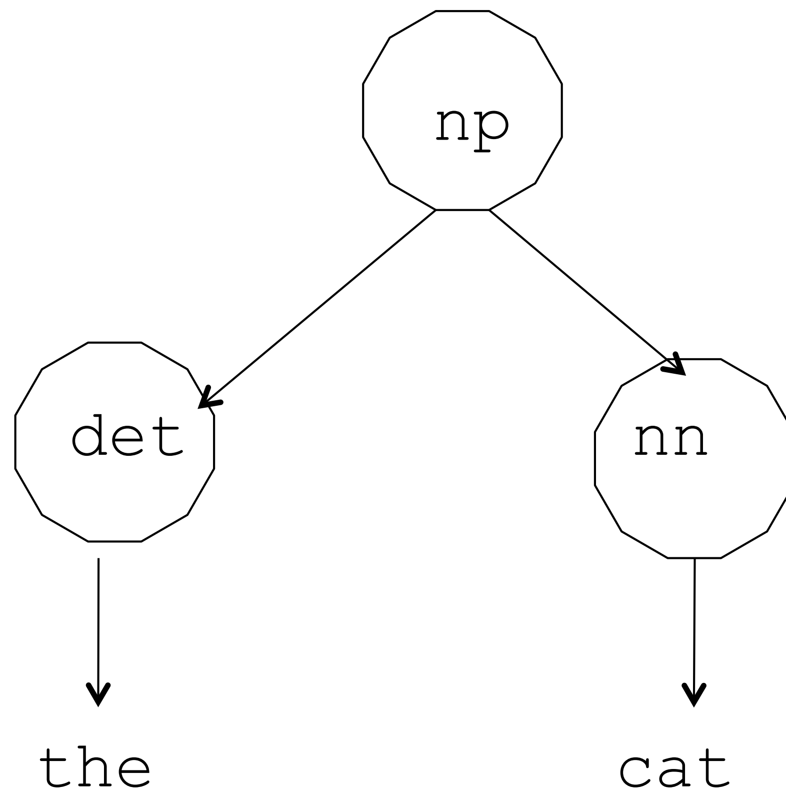
- Terms
  The name of the arguments are terms.
  - Atomic symbols (numbers or begin with a lowercase) define a given specific object (always the same)
    `mary, book, 3.15, -8, sing,` …
  - Variables (begin with an uppercase or underscore (objects waiting to be bounded)
    `X, Someone, Mary89, _C23, _9,` …
  - Compound terms
    (a functor (predicate name) (atomic) and its arguments)
    `aime(jean, marie), happy(john),`
    `np(adj(white), nn(house)),` …

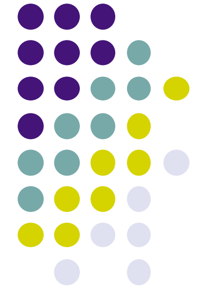
Terms



- The order is important (but free)
- Graphical representation of compound terms

  `np(det(the), nn(cat))`

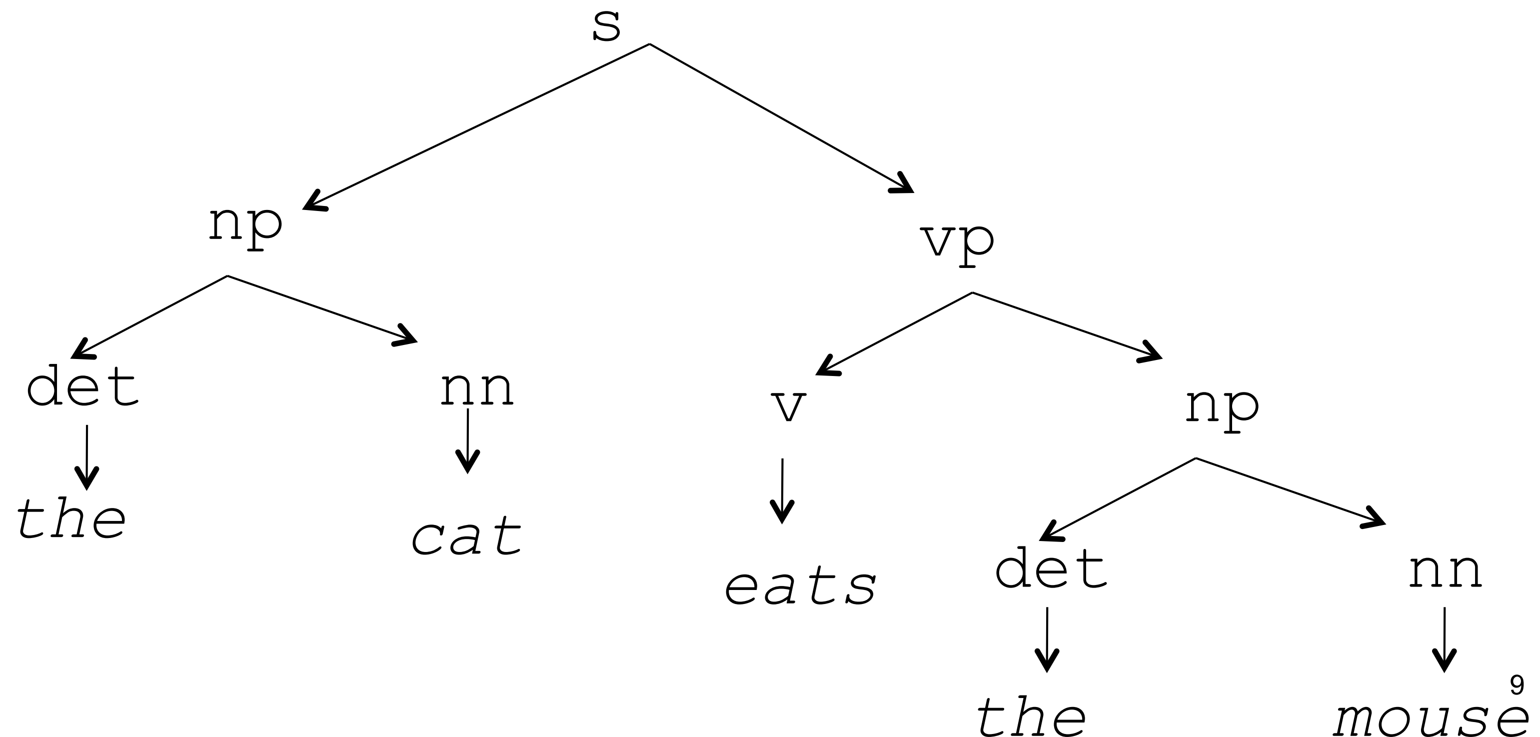




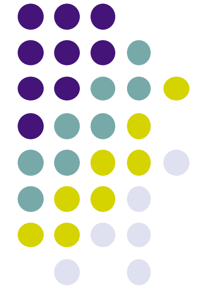
Terms

- Graphical representation of a compound term

```
s(np(det(the), nn(cat)),
     vp(v(eats), np(det(the), nn(mouse)))).
```



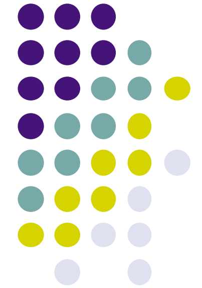
Prolog



- A simple program: enumerate facts and rules.

```
determiner(the).
determiner(this).
determiner(these).
noun(cat).
noun(mice).
noun(sheep).
adjective(lazy).
number(this,singular).
number(these,plural).
number(cat,singular).
number(mice,plural).
number(sheep,_).
number(the,_).
```

Prolog

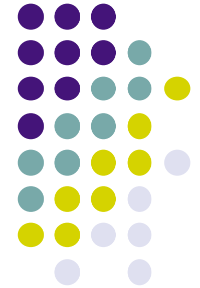


- And the rules (with comments!)

```
% Agreement between two words
agree(Word1,Word2) :- number(Word1,N),
                      number(Word2, N).

% How to form noun phrase
np(Mod,Gov) :- determiner(Mod),
               noun(Gov), agree(Mod, Gov).
np(Mod,Gov) :- adjective(Mod), noun(Gov).
```

- The order of the rule is important
- We must give all the possible definitions of a rule together

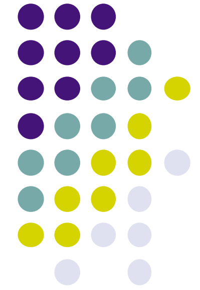
Prolog



- Usually it is a good idea to regroup all facts and rules of a given project into the same file (with the extension .pl)
- You can edit this file with your own editor (text only)
- You can launch the Prolog interpreter, by

```
machine% swipl
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Ver 5.6.64)
Copyright (c) 1990-2008 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free
software,and you are welcome to redistribute it under certain
conditions.
Please visit http://www.swi-prolog.org for details.
For help, use ?- help(Topic). or ?- apropos(Word).
```

Prolog

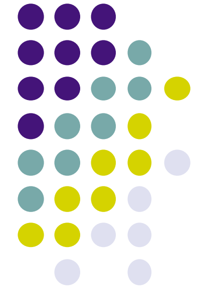


- With your PC, you have a directory "Program Files" where is located the directory "`pl`" (containing the SWI Prolog). Inside the "pl" folder, you can find the "`bin`" folder containing the executable program `plwin.exe`. You can create a shortcut to this program.

>double clik on "plwin.exes"

```
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Ver 5.6.64)
Copyright (c) 1990-2008 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free
software,and you are welcome to redistribute it under certain
conditions.
Please visit http://www.swi-prolog.org for details.
For help, use ?- help(Topic). or ?- apropos(Word).
```

Prolog



- After launching the Prolog interpreter, the system will prompt as

```
|?-
```

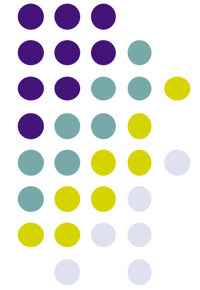
- You can load the corresponding file (or program) with the predicate

```
|?- consult('afilename.pl').
|?- ['afilename.pl'].
|?- reconsult('afilename.pl').
```

- and enter quit to end the session.

```
|?- halt.
```

Prolog

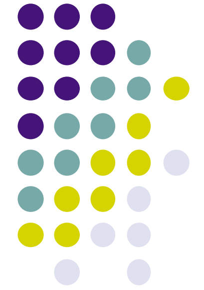


- With our first program and closed questions (goal)

```
|?- noun(cat).
true.
|?- noun(bird).
false.
|?- agree(this,cat).
true.
|?- agree(cat,mice).
false.
```

- Closed world assumption: if we cannot prove something, it is false.
- Prolog may return all possible answers (ways) to prove the goal.

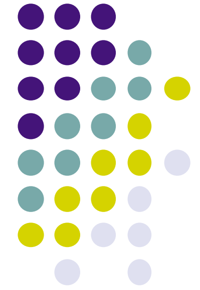
Prolog



- With our first program and open questions

```
|?- number(cat, N).
N = singular.

|?- number(mice, M).
M = plural.

|?- number(Mot,singular).
Mot = this ;
Mot = cat ;
Mot = sheep ;
Mot = the.
```

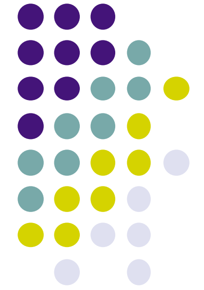
Unification in Prolog



- More than a pattern matching
  The interpreter tries to render both parts equals

1. An un instantiated variable will unify with any object. As a result, that object will be what the variable stands for.
2. Otherwise, an integer or atom will unify with only itself.
3. Otherwise, a compound term will unify wit another compound with
   1. the same functor (name),
   2. the same number of arguments,
   3. and all corresponding arguments must unify.

Prolog

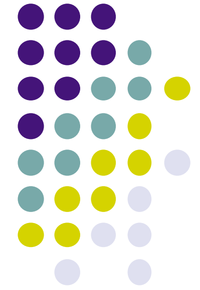


- More complex questions

```
|?- number(Word, singular), noun(Word).
Word = cat ;
Word = sheep ;
false.
|?- number(mice, G).
G = plural.
```

- To answer a given goal, Prolog unifies part of the question with its database. The returned answer is the substitution (or condition) that renders valid (proven) the question.

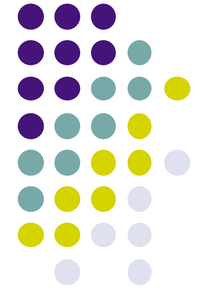
List in Prolog



- Special (predefined) predicate useful to manipulate an arbitrary number of objects (elements) such as the words in a sentence.
- Use the `[]` operator
  and separate the elements by a comma
- Examples of enumerated lists

```
[john]    % list with one element
[john,mary]  % list with two elements
[]        % Empty list (no element)
[john, 23, C]      % list with three elements
[s(np,vp), john]  % list with two elements
```

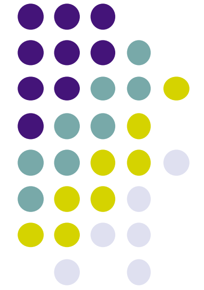
List in Prolog



- As the number of element is not known, we need to manipulate a list with a second constructor
- Use the head-tail `[…|…]` notation
  usually with variables as in `[H|T]`
  where `H` design the first element
  and `T` the tail (a *list* without the first element)
- Examples

```
[a,b,c] = [H|T]
% H = a,    T = [b,c].
[a,b] = [H|T]
% H = a,    T = [b].
[a] = [H|T]
% H = a,    T = [].
```

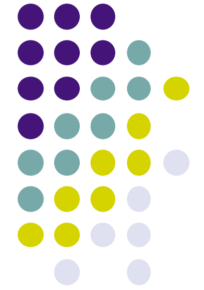
Predicate with Lists



- How can we count the number of element in a list.
- Example

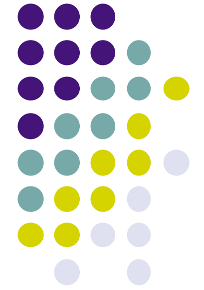
```
length([a,b,c],N)
N = 3.
length([],N)
N = 0.
length(n,N)
ERROR: length/2: Type error: `list'
expected, found `n'.

% The definition (already given in SWI)
length([],0).
length([H|T], N) :- length(T,N1), N is N1+1.
```

Predicate with Lists



- To concatenate two lists
- Example

```
append([a,b,c],[d,e],L)
L = [a,b,c,d,e].
append([a,b,c],L,[a,b,c,d,e])
L = [d,e].

% The definition
append([],L,L).
append([H|T],L,[H|Ts]) :- append(T,L,Ts).
```



Grammar, version 0

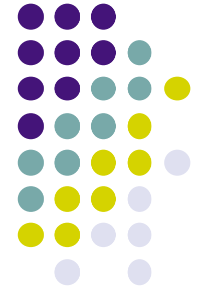
- A first example in French.
  The vocabulary used and its corresponding POS

```
det([le]).
det([la]).

n([souris]).
n([chat]).

v([mange]).
v([trottine]).
```

We will limit ourselves to this rather small vocabulary.

We have use list to represent each word.

Grammar, version 0

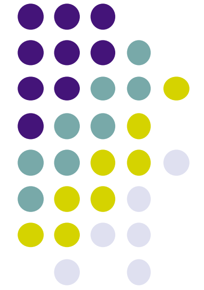


- A minimal French syntax

```
p(L) :- sn(L1), sv(L2), append(L1,L2,L).
sn(L) :- det(L1), n(L2), append(L1,L2,L).
sv(L) :- v(L).
sv(L) :- v(L1), sn(L2), append(L1,L2,L).
```

- A sentence (predicate `p`) is composed first by a noun phrase (predicate `sn`) followed by a verb phrase (predicate `sv`).
- A noun phrase owns a single form (a single rule), a determinant followed by a noun.
- A verb phrase could be a single verb (predicate `v`) or a verb followed by a noun phrase.
- In all cases, if we found at the beginning of the list (of words) what we need, we remove it (see the vocabulary).

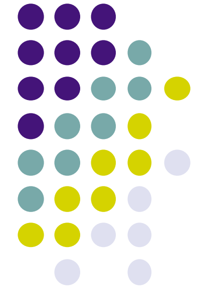
Grammar, version 0



- Example

```
p([le, chat, mange]).
true ;
false.
p([la, souris, trottine]).
true ;
false.
p([la, souris, Action]).
Action = mange ;
Action = trottine ;
false.
```

Grammar, version 0

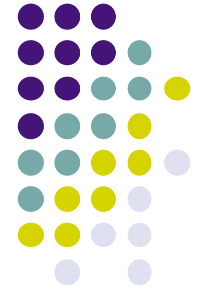


- Generate all possible sentences

```
|?- p(S).
S = [le, souris, mange] ;
S = [le, souris, trottine] ;
S = [le, souris, mange, le, souris] ;
S = [le, souris, mange, le, chat] ;
S = [le, souris, mange, la, souris] ;
S = [le, souris, mange, la, chat] ;
S = [le, souris, trottine, le, souris] ;
…
```

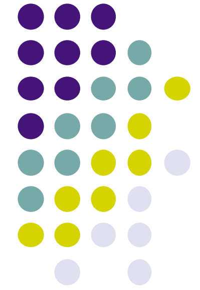
- We can thus use our program to parse a sentence (according to our minimal syntax of French) or to generate sentences (according to this grammar).

Grammar, version 0



- Problems
- The first solution is *ad hoc*.
  Usually we prefer the following computational model:
  Input: using one parameter (term/ list)
  Output: return information in another parameter (list).
- Difficult to extent this program to include other features
  - agreement between the determinant and the noun
  - agreement between the subject and the verb
  - agreement between the transitive verb and the complement
  - output the POS and/or the parsed tree

Grammar, version 1

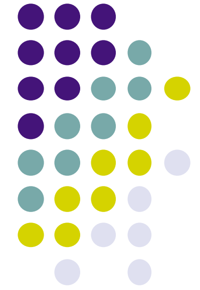


- We will use the list difference approach. The first list indicates the input elements (of words) and the second the output list.

```
p(L0,L) :- sn(L0,L1), sv(L1,L).
sn(L0,L) :- det(L0,L1), n(L1,L).
sv(L0,L) :- v(L0,L).
sv(L0,L) :- v(L0,L1), sn(L1,L).
```

- If the parsing is possible, `L = []`.
- In the syntactic elements, we transform the input list (by removing elements corresponding to the analyzed syntactic variable). More than one solution may exist (and the Prolog interpreter will explore them).

Grammar, version 1

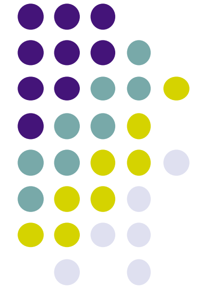


- We store the vocabulary using the difference list.

```
det(L0,L) :- terminal(le,L0,L).
det(L0,L) :- terminal(la,L0,L).
n(L0,L) :- terminal(souris,L0,L).
n(L0,L) :- terminal(chat,L0,L).
v(L0,L) :- terminal(mange,L0,L).
v(L0,L) :- terminal(trottine,L0,L).
terminal(Word,[Word|L],L).
```

- We remove the corresponding word from the head of the list and return the input list minus this word.

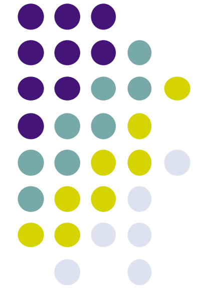
Grammar, version 1



- Examples

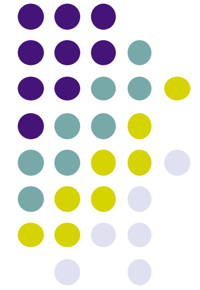
```
?- p([le, chat, mange],L).
L = [] ;
false.

?- p([le, chat | R],[]).
R = [mange] ;
R = [trottine] ;
R = [mange, le, souris] ;
R = [mange, le, chat] ;
R = [mange, la, souris] ;
R = [mange, la, chat] ;
R = [trottine, le, souris] ;
R = [trottine, le, chat] ;
R = [trottine, la, souris] ;
R = [trottine, la, chat] ;
false.
```

Grammar, version 1



- Problems?
- No agreement between the determinant and the noun (or between the subject and the verb, not needed however in our very simple vocabulary)
  Examples
  "la chat trottine" (*)
  "le chat mange le souris" (*)
- A verb could be intransitive (without direct object). We do not consider this constraint in our solution.
  "le chat trottine la souris" (*)

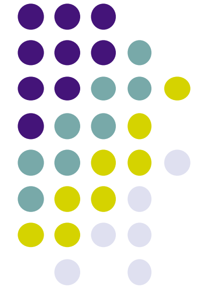
Grammar, version 2



- We still use the list difference approach. We add the constraint about the agreement (gender) between the determinant and the noun. For the verb, we take account that intransitive verb cannot have a complement.

```
p(L0,L) :- sn(L0,L1), sv(L1,L).
sn(L0,L) :- det(Genre,L0,L1),
            n(Genre,L1,L).
sv(L0,L) :- v(_,L0,L).
sv(L0,L) :- v(transitif,L0,L1), sn(L1,L).
```

Grammar, version 2

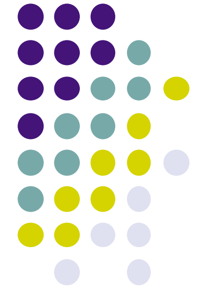


- And the new vocabulary with its attributes (gender, transitivity).

```
det(masculin,L0,L) :- terminal(le,L0,L).
det(feminin,L0,L) :- terminal(la,L0,L).
n(feminin,L0,L) :- terminal(souris,L0,L).
n(masculin,L0,L) :- terminal(chat,L0,L).
v(transitif,L0,L) :- terminal(mange,L0,L).
v(intransitif,L0,L):-terminal(trottine,L0,L).
terminal(Mot,[Mot|L],L).
```

- We discriminate between feminine and masculine nouns and determinant.
- A verb can be transitive or intransitive

Grammar, version 2

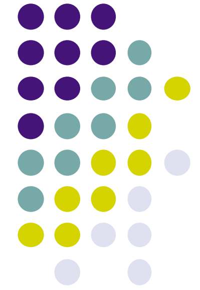


- Example

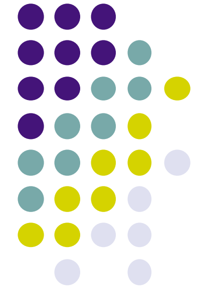
```
?- p([le,chat,mange],[]).
true ;
false.
?- p([la,chat,mange],[]).
false.
?- p([le,chat | R],[]).
R = [mange] ;
R = [trottine] ;
R = [mange, le, chat] ;
R = [mange, la, souris] ;
false.
```

- Every thing is OK?

Grammar, version 2



- Problems
- We want more than just a binary answer (correct, failure) but the parsing tree
- It makes no sense to accept stupid sentences such as "le chat mange le chat" (*)
  The subject and the complement are the same!
- We need to introduce semantic constraints
  "la souris mange le chat" (*)

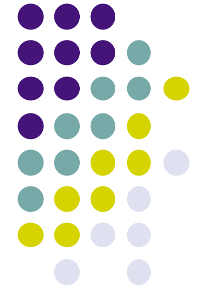


Grammar, version 3

- The final example is more complex. We return the parsing tree.

```
p(ph(SN_Struct,SV_Struct),L0,L) :-
      sn(SN_Struct,L0,L1),
      sv(SV_Struct,L1,L),
      % be sure that the 2 SN are differents
     not(SV_Struct = svb(_,SN_Struct)).
sn(snm(Det_Struct,N_Struct),L0,L) :-
      det(Det_Struct,Genre,L0,L1), n(N_Struct,Genre,L1,L).
sv(svb(vb(Mot)),L0,L) :- v(vb(Mot,_),_,L0,L).
sv(svb(vb(Mot),SN_Struct),L0,L) :-
      v(vb(Mot,Comp),transitif,L0,L1), sn(SN_Struct,L1,L),
      % check that the complement is valid
      SN_Struct = snm(_,nm(Nom)),   % extract the noun
      T =.. [Comp,Nom],             % built the predicate
      call(T).                      % call it
```

Grammar, version 3



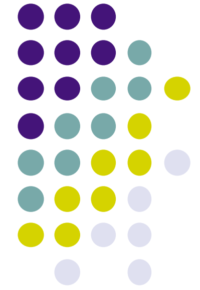
- The vocabulary

```
det(dt(le),masculin,L0,L) :- terminal(le,L0,L).
det(dt(la),feminin,L0,L)  :- terminal(la,L0,L).
n(nm(souris),feminin,L0,L) :-
    terminal(souris,L0,L).
n(nm(chat),masculin,L0,L)  :- terminal(chat,L0,L).
v(vb(mange,prey),transitif,L0,L) :-
    terminal(mange,L0,L).
v(vb(trottine,_),intransitif,L0,L) :-
     terminal(trottine,L0,L).
terminal(Mot,[Mot|L],L).
```

- And the semantic predicates

```
prey(souris).
```

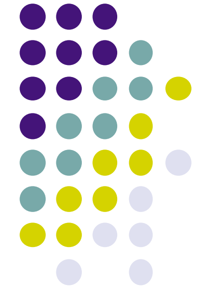
Grammar, version 3



- Example

```
?- p(S,[le,chat,trottine],[]).
S = ph(snm(dt(le), nm(chat)), svb(vb(trottine))) ;
false.
?- p(Struct,[le,chat,mange,la,souris],[]).
Struct = ph(snm(dt(le), nm(chat)), svb(vb(mange),
snm(dt(la), nm(souris)))) ;
false.
?- p(S,[le,chat | R],[]).
S = ph(snm(dt(le), nm(chat)), svb(vb(mange)))
R = [mange] ;
S = ph(snm(dt(le), nm(chat)), svb(vb(trottine)))
R = [trottine] ;
S = ph(snm(dt(le), nm(chat)), svb(vb(mange),
snm(dt(la), nm(souris))))
R = [mange, la, souris] ;
false.
```

Conclusion



- PROLOG is very useful to analyze (parse) or to generate sentences according to a syntax.
- We may work on the vocabulary (and the corresponding POS and grammatical categories) on the one hand, and on the other on the syntax.
- Difference list is a powerful strategy in parsing.
- PROLOG is however not very useful for input/output operations.