XSB Prolog (cont.)

CSE 392, Computers Playing Jeopardy!, Fall 2011

Stony Brook University

http://www.cs.stonybrook.edu/~cse392

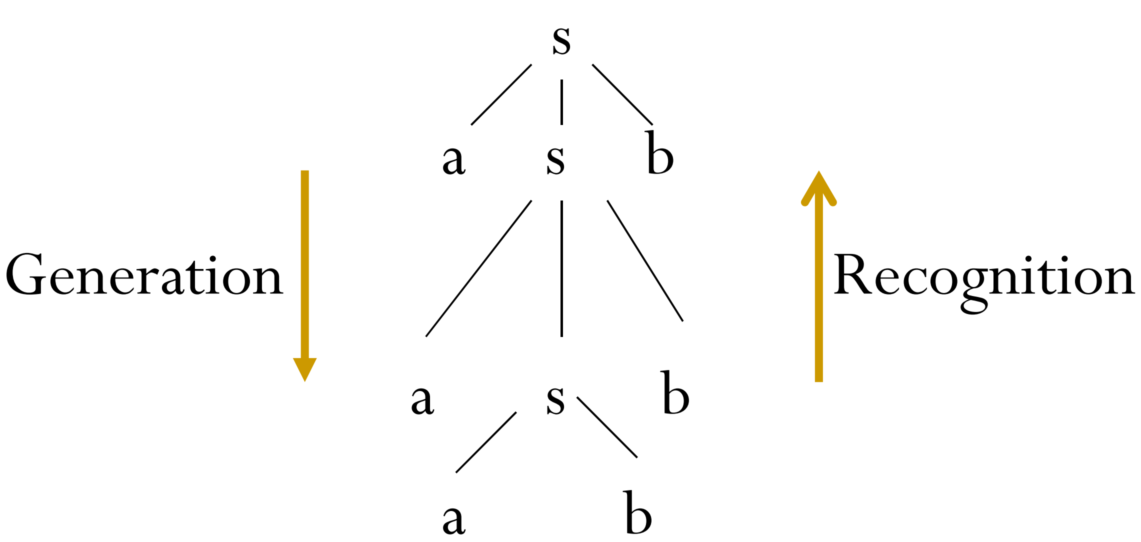
Definite clause grammar (DCG)

- A **DCG** is a way of expressing grammar in a logic programming language such as Prolog
- The definite clauses of a DCG can be considered a set of axioms where the fact that it has a parse tree can be considered theorems that follow from these axioms

A BNF grammar

<s> ::= a b | a <s> b

Grammar generates / recognises



@ Ivan Bratko

Sentence = a a a b b b

COMMAND SEQUENCES FOR A ROBOT

- up
- up up down up down

- CORRESPONDING BNF GRAMMAR

@ Ivan Bratko

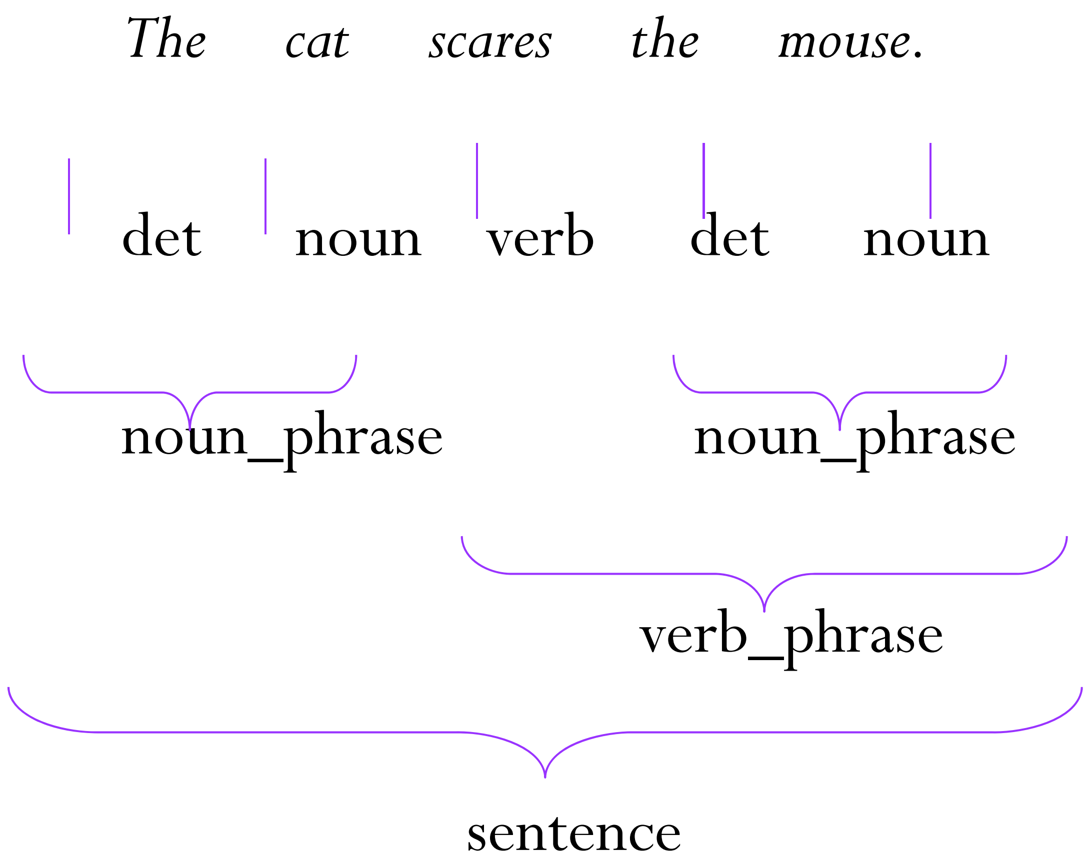
- <move> ::= <step> | <step> <move>
- <step> ::= up | down

CORRESPONDING DCG

```
move --> step.
move --> step, move.
step --> [up].
step --> [down].
?- move( [up,down,up], []).
yes
?- move( [up, X, up], []).
X = up;
X = down;
no
```

@ Ivan Bratko

A SIMPLE NATURAL LANGUAGE DCG



A SIMPLE NATURAL LANGUAGE DCG

```
sentence --> noun_phrase, verb_phrase.
verb_phrase --> verb, noun_phrase.
noun_phrase --> determiner, noun.
determiner --> [ the].
noun --> [ cat].
noun --> [ cats].
noun --> [ mouse].
verb --> [ scares].
verb --> [ scare].
```

THIS GRAMMAR GENERATES

[the, cat, scares, the, mouse]

[the, mouse, scares, the, mouse]

[the, cats, scare, the, mouse]

[the, cats, scares, the, mouse]

CONTEXT DEPENDENT!

NUMBER AGREEMENT CAN BE FORCED BY ARGUMENTS

```
sentence( Number) -->
    noun_phrase( Number), verb_phrase( Number).

verb_phrase( Number) -->
    verb( Number), noun_phrase( Number1).

noun_phrase( Number) -->
    determiner( Number), noun( Number).

noun( singular) --> [ mouse].
noun( plural) --> [ mice].
verb( singular) --> [scares].
verb( plural) --> [scare].
```

MEANING OF MOVES TREE

- meaning(move(Step, Move), Dist) :-
- meaning(Step, D1),
- meaning(Move, D2),
- Dist is D1 + D2.

- meaning(step(up), 1).

- meaning(step(down), -1).

INTERLEAVING SYNTAX AND MEANING

sentence ⟶ parse tree ⟶ meaning

- Avoid parse tree, encode meaning directly in DCG

PROLOG goals in DCG: {Goal}

- move(Dist) --> step(Dist).
- move(Dist) -->
- step(D1), move(D2), {Dist is D1 + D2}.
- step(1) --> [up].
-
- step(-1) --> [down].
-
- ?- move(D, [up, up, down, up], [ ]).
- D = 2

MEANING OF NATURAL LANGUAGE

- Representation of meaning = ?

- Depends on use of meaning,
- e.g. natural language querying

- Logic is a good candidate for representing meaning

SOME MEANINGS IN LOGIC

Sentence	Formalised meaning
``John paints"	*paints(john)*
``John likes Annie"	*likes(john, annie)*

SOME MEANINGS IN LOGIC

- Sentence
- ``A man paints”
- Formalised meaning
- *exists(X, man(X) and paints(X))*
- Note: ``paints” is intransitive verb, ``likes” is trans. verb

A SYNTAX

- sentence ---> noun_phrase, verb_phrase.
- noun_phrase --> proper_noun.
- verb_phrase --> intrans_verb.
- verb_phrase --> trans_verb, noun_phrase.
- intrans_verb --> [paints].
- trans_verb --> [likes].
- proper_noun --> [john].
- …

INCORPORATING MEANING

- % “john” means “john”
- proper_noun(john) --> [john].
- % “paints” means “paints(X)”
- intrans_verb(paints(X)) --> [paints].

Another DCG Example

```
sentence --> noun_phrase, verb_phrase.
noun_phrase --> det, noun.
verb_phrase --> verb, noun_phrase.
det --> [the].
det --> [a].
noun --> [cat].
noun --> [bat].
verb --> [eats].

?- sentence(X,[]).
```

DCG

- Not only <u>context-free grammars</u>
- <u>Context-sensitive grammars</u> can also be expressed with DCGs, by providing extra arguments

```
s --> symbols(Sem,a), symbols(Sem,b), symbols(Sem,c).
symbols(end,_) --> [].
symbols(s(Sem),S) --> [S], symbols(Sem,S).
```

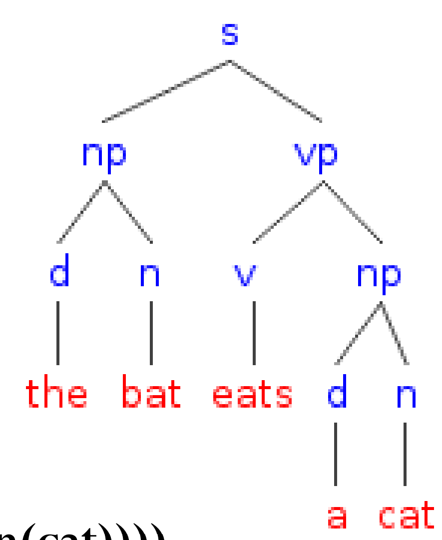
DCG

```
sentence --> pronoun(subject), verb_phrase.
verb_phrase --> verb, pronoun(object).
pronoun(subject) --> [he].
pronoun(subject) --> [she].
pronoun(object) --> [him].
pronoun(object) --> [her].
verb --> [likes].
```

Parsing with DCGs

```
sentence(s(NP,VP)) --> noun_phrase(NP), verb_phrase(VP).
noun_phrase(np(D,N)) --> det(D), noun(N).
verb_phrase(vp(V,NP)) --> verb(V), noun_phrase(NP).
det(d(the)) --> [the].
det(d(a)) --> [a].
noun(n(bat)) --> [bat].
noun(n(cat)) --> [cat].
verb(v(eats)) --> [eats].

?- sentence(Parse_tree, [the,bat,eats,a,cat], []).
Parse_tree = s(np(d(the),n(bat)),vp(v(eats),np(d(a),n(cat))))
```



```
s --> np, vp.
np --> det, n.
vp --> tv, np.
vp --> v.
det --> [the].
det --> [a].
det --> [every].
n --> [man].
n --> [woman].
n --> [park].
tv --> [loves].
tv --> [likes].
v --> [walks].
```

```
| ?- s([a,man,loves,the,woman],[]).
yes
| ?- s([every,woman,walks],[]).
yes
| ?- s([a,woman,likes,the,park],[]).
yes
| ?- s([a,woman,likes,the,prak],[]).
no
```

Another DCG

http://www.csupomona.edu/~jrfisher/www/prolog_tutorial/7_2.html

```
s --> np, vp.               /* sentence */
np --> pn.                  /* noun phrase */
np --> d, n, rel.
vp --> tv, np.              /* verb phrase */
vp --> iv.
rel --> [].                 /* relative clause */
rel --> rpn, vp.
pn --> [PN], {pn(PN)}.      /* proper noun */
pn(mary).
pn(henry).
rpn --> [RPN], {rpn(RPN)}.  /* relative pronoun */
rpn(that).
rpn(which).
rpn(who).
```

```
iv --> [IV], {iv(IV)}.      /* intransitive verb */
iv(runs).
iv(sits).
d --> [DET], {d(DET)}.      /* determiner */
d(a).
d(the).
n --> [N], {n(N)}.          /* noun */
n(book).
n(girl).
n(boy).
tv --> [TV], {tv(TV)}.      /* transitive verb */
tv(gives).
tv(reads).
```

```
?- s([the,boy,who,sits,reads,a,book],[]).         yes
```

- DCG rules can contain arguments (generate parse trees)

```
s(s(NP,VP)) --> np(Num,NP), vp(Num,VP).
np(Num,np(PN)) --> pn(Num,PN).
np(Num,NP) -->
  d(Det),
  n(Num,N),
  rel(Num,Rel),
  {build_np(Det,N,Rel,NP)}. /* embedded Prolog goal */
/* Prolog rules for build_np */
build_np(Det,N,rel(nil),np(Det,N)).
build_np(Det,N,rel(RP,VP),np(Det,N,rel(RP,VP))).
vp(Num,vp(TV,NP)) -->
        tv(Num,TV),
        np(_,NP).
vp(Num,vp(IV)) --> iv(Num,IV).
rel(_Num,rel(nil)) --> [].
rel(Num,rel(RP,VP)) -->
        rpn(RP), vp(Num,VP).
?- s(Parse_form,'The boy who sits reads the book',[]).
Parse_form=s(np(d(the),n(boy),rel(rpn(who),vp(iv(sits)))),vp(tv(reads),np(d(a),n(book))))
```

```
d(d(DET)) --> [DET], {d(DET)}.
d(a).
d(the).

n(sing,n(N)) --> [N], {n(N,_X)}.
n(plu,n(N)) --> [N], {n(_X,N)}.
n(book,books).
n(girl,girls).
n(boy,boys).

tv(sing,tv(TV)) --> [TV], {tv(TV,_X)}.
tv(plu,tv(TV)) --> [TV], {tv(_X,TV)}.
tv(gives,give).
tv(reads,read).
```

Cut (logic programming)

- Cut (! in Prolog) is a goal which always succeeds, but cannot be backtracked past
- **Green cut**

  gamble(X) :- gotmoney(X),!.

  gamble(X) :- gotcredit(X), \+ gotmoney(X).

  - **cut says "stop looking for alternatives"**
  - by explicitly writing \+ gotmoney(X), it guarantees that the second rule will always work even if the first one is removed by accident or changed
- **Red cut**

  gamble(X) :- gotmoney(X),!.

  gamble(X) :- gotcredit(X).

Prolog Wordnet

- Wordnet: http://wordnet.princeton.edu/wordnet/
- **Prolog version of WordNet 3.0**
- **Accessing WordNet from Prolog, Sarah Witzig**

http://www.ai.uga.edu/mc/pronto/Witzig.pdf

Prolog Wordnet Examples

- OpenRuleBench Prolog Wordnet examples:

http://projects.semwebcentral.org/scm/viewvc.php/openrulebench/wordnet/?root=rulebench

```
sameSynsets(Word1,Word2):-
    s(Synset_id,_W_num1,Word1,_Ss_type1,_Sense_number1,_Tag_count1),
    s(Synset_id,_W_num2,Word2,_Ss_type2,_Sense_number2,_Tag_count2).
    %Word1 \= Word2.
gloss(Word1,Gloss):-
    s(Synset_id,_W_num1,Word1,_Ss_type1,_Sense_number1,_Tag_count1),
    g(Synset_id,Gloss).
```

Prolog Wordnet Examples

```
directHypernym(Word1,Word2):-
    s(Synset_id1,_W_num1,Word1,_Ss_type1,_Sense_number1,_Tag_count1),
    hyp(Synset_id1,Synset_id2),
    s(Synset_id2,_W_num2,Word2,_Ss_type2,_Sense_number2,_Tag_count2).
antonym(Word1,Word2):-
    s(Synset_id1,W_num1,Word1,_Ss_type1,_Sense_number1,_Tag_count1),
    ant(Synset_id1,W_num1,Synset_id2,W_num2),
    s(Synset_id2,W_num2,Word2,_Ss_type2,_Sense_number2,_Tag_count2).
```

XSB Prolog Wordnet Examples

```
:-index(hyp/2,[1,2]).
hypernymSynsets(S1,S2):-
  hyp(S1,S2). % direct hypernym sets
:- table(hypernymSynsets/2).
hypernymSynsets(S1,S2):-
  hyp(S1,S3),
  hypernymSynsets(S3,S2). % multiple-levels hypernym sets
hypernym(Word1,Word2):-
  s(Synset_id1,_W_num1,Word1,_Ss_type1,_Sense_number1,_Tag_count1),
  hypernymSynsets(Synset_id1,Synset_id2),
  s(Synset_id2,_W_num2,Word2,_Ss_type2,_Sense_number2,_Tag_count2).
```