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

\[ <s> ::= a \ b \mid a <s> b \]

Grammar generates / recognises

Sentence = a a a b b b

(c) 2011 P. Fodor (CS Stony Brook) & Wikipedia
COMMAND SEQUENCES FOR A ROBOT

- up
- up up down up down

CORRESPONDING BNF GRAMMAR

- \texttt{<move>} ::= \texttt{<step>} | \texttt{<step>} \texttt{<move>}
- \texttt{<step>} ::= \texttt{up} | \texttt{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

(c) 2011 P.Fodor (CS Stony Brook) & Wikipedia
The cat scares the mouse.
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

- Avoid parse tree, encode meaning directly in DCG
move( Dist)  -->  step( Dist).

move( Dist)  -->  

  step( D1), move( D2), \{Dist \text{ 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

(c) 2011 P. Fodor (CS Stony Brook) & Wikipedia
## SOME MEANINGS IN LOGIC

<table>
<thead>
<tr>
<th>Sentence</th>
<th>Formalised meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>“John paints”</td>
<td>$\text{paints}(\text{john})$</td>
</tr>
<tr>
<td>“John likes Annie”</td>
<td>$\text{likes}(\text{john},\text{annie})$</td>
</tr>
</tbody>
</table>
SOME MEANINGS IN LOGIC

- Sentence

- "A man paints"

- Formalised meaning

- \( \text{exists}(X, \text{man}(X) \text{ and } \text{paints}(X)) \)

- Note: "paints" is intransitive verb, "likes" is trans. verb

(c) 2011 P. Fodor (CS Stony Brook) & Wikipedia
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].
- ...

(c) 2011 P.Fodor (CS Stony Brook) & Wikipedia
INCORPORATING MEANING

- % “john” means “john”
- proper_noun(john) -> [john].
- % “paints” means “paints(X)”
- intran_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 context-free grammars
• Context-sensitive grammars can also be expressed with DCGs, by providing extra arguments

\[ s \rightarrow \text{symbols}(\text{Sem},a), \text{symbols}(\text{Sem},b), \text{symbols}(\text{Sem},c). \]

\[ \text{symbols}(\text{end},\_)) \rightarrow \text{[]} . \]

\[ \text{symbols}(s(\text{Sem},S)) \rightarrow [S], \text{symbols}(\text{Sem},S). \]
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

\[
\begin{align*}
\text{sentence} & \rightarrow \text{noun\_phrase}, \text{verb\_phrase}. \\
\text{noun\_phrase} & \rightarrow \text{det}, \text{noun}. \\
\text{verb\_phrase} & \rightarrow \text{verb}, \text{noun\_phrase}. \\
\text{det} & \rightarrow [\text{the}]. \\
\text{det} & \rightarrow [\text{a}]. \\
\text{noun} & \rightarrow [\text{bat}]. \\
\text{noun} & \rightarrow [\text{cat}]. \\
\text{verb} & \rightarrow [\text{eats}]. \\
\end{align*}
\]

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

(c) 2011 P.Fodor (CS Stony Brook) & Wikipedia
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.
np --> [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)

\[
\begin{align*}
s(s(NP,VP)) & \rightarrow np(Num,NP), \ vp(Num,VP). \\
np(Num,np(PN)) & \rightarrow pn(Num,PN). \\
np(Num,NP) & \rightarrow \\
& \quad d(Det), \\
& \quad n(Num,N), \\
& \quad rel(Num,Rel), \\
& \quad \{\text{build}_np(Det,N,Rel,NP)\}. /* \text{embedded Prolog goal} */ \\
\text{* Prolog rules for build}_np */ \\
build\_np(Det,N,rel(nil),np(Det,N)). \\
build\_np(Det,N,rel(RP,VP),np(Det,N,rel(RP,VP))). \\
vp(Num,\vp(TV,NP)) & \rightarrow \\
& \quad tv(Num,Tv), \\
& \quad np(\_,NP). \\
vp(Num,\vp(IV)) & \rightarrow iv(Num,IV). \\
rel(\_,\text{rel(nil)}) & \rightarrow []. \\
rel(Num,\text{rel(RP,VP)}) & \rightarrow \\
& \quad \text{rpn(RP)}, \ vp(Num,VP). \\
\end{align*}
\]

?- \(s(\text{Parse}\_\text{form},'The\ boy\ who\ sits\ reads\ the\ book',[[]])\).
Parse\_form=\(s(np(d(\text{the}),n(\text{boy}),rel(\text{rpn(who),vp(iv(sits)))),\ vp(tv(\text{reads}),np(d(a),n(\text{book})))))\)
Cut (logic programming)

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

- **Green cut**
  
  ```prolog
  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**
  
  ```prolog
  gamble(X) :- gotmoney(X),!.
  gamble(X) :- gotcredit(X).
  ```
Prolog Wordnet

- Wordnet: [http://wordnet.princeton.edu/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](http://www.ai.uga.edu/mc/pronto/Witzig.pdf)
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).
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).

(c) 2011 P.Fodor (CS Stony Brook) & Wikipedia