Computers Playing Jeopardy!
Stony Brook University
Prolog Example: Reachability

edge(1,2).
edge(2,3).
edge(2,4).

reachable(X,Y) :- edge(X,Y).
reachable(X,Y) :- edge(X,Z), reachable(Z, Y).
Prolog Example: Reachability

| ?- reachable(X,Y). |

X = 1
Y = 2;  Type a semi-colon repeatedly

X = 2
Y = 3;

X = 2
Y = 4;

X = 1
Y = 3;

X = 1
Y = 4;

no

| ?- halt.  Command to Exit XSB |
Data types

- An **atom** is a general-purpose name with no inherent meaning.
- **Numbers** can be floats or integers.
- A **compound term** is composed of an atom called a "functor" and a number of "arguments", which are again terms: `tree(node(a),tree(node(b),node(c))))`
- Special cases of compound terms:
  - **Lists**: ordered collections of terms: `[], [1,2,3], [a,1,X|T]`
  - **Strings**: A sequence of characters surrounded by quotes is equivalent to a list of (numeric) character codes: “abc”, “to be, or not to be”
Representation of Lists

- List is handled as binary tree in Prolog
  
  [Head | Tail]  OR  
  .(Head, Tail)  

  - Where Head is an atom and Tail is a list  
  - We can write [a,b,c] or .(a,.(b,.(c,[]))).
Matching

- Given two terms, they are identical or the variables in both terms can have same objects after being instantiated
  - date(D,M,2006) unification date(D1,feb,Y1)
    - D=D1, M=feb, Y1=2006
- General Rule to decide whether two terms, S and T match are as follows:
  - If S and T are constants, S=T if both are same object
  - If S is a variable and T is anything, T=S
  - If T is variable and S is anything, S=T
  - If S and T are structures, S=T if
    - S and T have same functor
    - All their corresponding arguments components have to match
Append example

```prolog
append([], L, L).
append([X|L], M, [X|N]) :- append(L, M, N).
```

append([1,2],[3,4],X)?
append([],L,L).
append([X|L],M,[X|N]) :- append(L,M,N).

append([1,2],[3,4],X)?

X=1, L=[2], M=[3,4], A=[X|N]
Append example

append([],L,L).
append([X|L],M,[X|N]) :- append(L,M,N).

append([2],[3,4],N)?
append([1,2],[3,4],X)?  X=1, L=[2], M=[3,4], A=[X|N]
Append example

append([], L, L).
append([X|L], M, [X|N’]) :- append(L, M, N’).

append([2], [3,4], N)? X=2, L=[], M=[3,4], N=[2|N’]
append([1,2], [3,4], X)? X=1, L=[2], M=[3,4], A=[1|N]
Append example

append([],L,L).
append([X|L],M,[X|N']) :- append(L,M,N').

append([],[3,4],N')?
append([2],[3,4],N)?
append([1,2],[3,4],X)?
Append example

\[
\text{append}([], L, L).
\]

\[
\text{append}([X|L], M, [X|N']) \ :- \ \text{append}(L, M, N').
\]

\begin{tabular}{|l|l|}
\hline
append([], [3, 4], N')? & L = [3, 4], N' = L \\
\hline
append([2], [3, 4], N)? & X=2, L=[], M=[3, 4], N=[2|N'] \\
\hline
append([1, 2], [3, 4], X)? & X=1, L=[2], M=[3, 4], A=[1|N] \\
\hline
\end{tabular}
### Append example

```prolog
append([], L, L).
append([X|L], M, [X|N']) :- append(L, M, N').
```

#### Example:

- **append([], [3,4], N')?**
  - `A = [1|N]
  - `N = [2|N']
  - `N' = L
  - `L = [3,4]
  - Answer: `A = [1,2,3,4]

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>append([], [3,4], N')?</td>
<td><code>L = [3,4], N' = L</code></td>
</tr>
<tr>
<td>append([2], [3,4], N)?</td>
<td>`X=2, L=[], M=[3,4], N=[2</td>
</tr>
<tr>
<td>append([1,2], [3,4], X)?</td>
<td>`X=1, L=[2], M=[3,4], A=[1</td>
</tr>
</tbody>
</table>
Quicksort Example

\[
\text{partition([], _, [], [])}.
\]
\[
\text{partition([X|Xs], Pivot, Smalls, Bigs) :-}
\]
\[
\begin{align*}
\quad & ( \quad X @< \text{Pivot} \quad \rightarrow \\
\quad & \quad \text{Smalls} = [X|\text{Rest}], \\
\quad & \quad \text{partition(Xs, Pivot, Rest, Bigs)} \\
\quad & ; \quad \text{Bigs} = [X|\text{Rest}], \\
\quad & \quad \text{partition(Xs, Pivot, Smalls, Rest)} \\
\quad ) .
\end{align*}
\]
\[
\text{quicksort([]) } \quad \rightarrow \quad [] .
\]
\[
\text{quicksort([X|Xs]) } \quad \rightarrow \\
\quad \{ \quad \text{partition(Xs, X, Smaller, Bigger)} \quad \}, \\
\quad \text{quicksort(Smaller), [X], quicksort(Bigger)} .
\]
More Examples

member(X,[X | R]).
member(X,[Y | R]) :- member(X,R)

- $X$ is a member of a list whose first element is $X$.
- $X$ is a member of a list whose tail is $R$ if $X$ is a member of $R$.

?- member(2,[1,2,3]).
Yes

?- member(X,[1,2,3]).
X = 1 ;
X = 2 ;
X = 3 ;
No
More Examples

\[ \text{select}(X,[X | R],R) \].
\[ \text{select}(X,[F | R],[F | S]) :- \text{select}(X,R,S). \]

- When \( X \) is selected from \([X | R]\), \( R \) results.
- When \( X \) is selected from the tail of \([X | R]\), \([X | S]\) results, where \( S \) is the result of taking \( X \) out of \( R \).

?- select(X,[1,2,3],L).
X=1  L=[2,3] ;
X=2  L=[1,3] ;
X=3  L=[1,2] ;
No
append([],X,X).
append([X|Y],Z,[X|W]) :- append(Y,Z,W).

?- append([1,2,3],[4,5],X).
X=[1,2,3,4,5]
Yes
More Examples

reverse([X | Y], Z, W) :- reverse(Y, [X | Z], W).
reverse([], X, X).

?- reverse([1, 2, 3], [], X).
X = [3, 2, 1]
Yes
More Examples

perm([],[]).
perm([X | Y],Z) :- perm(Y,W), select(X,Z,W).

?- perm([1,2,3],P).
P = [1,2,3] ;
P = [2,1,3] ;
P = [2,3,1] ;
P = [1,3,2] ;
P = [3,1,2] ;
P = [3,2,1]
More Examples

- **Sets**

  union([X|Y],Z,W) :- member(X,Z),  union(Y,Z,W).
  union([X|Y],Z,[X|W]) :- \+ member(X,Z), union(Y,Z,W).
  union([],Z,Z).

  intersection([X|Y],M,[X|Z]) :- member(X,M), intersection(Y,M,Z).
  intersection([X|Y],M,Z) :- \+ member(X,M), intersection(Y,M,Z).
  intersection([],M,[]).
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

Generation

\[
\begin{array}{c}
\text{a} \\
\text{s} \\
\text{b}
\end{array}
\]

Recognition

\[
\begin{array}{c}
\text{a} \\
\text{s} \\
\text{b}
\end{array}
\]

Sentence = a a a b b b

@ Ivan Bratko

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

- up
- up up down up down

CORRESPONDING BNF GRAMMAR

- \(<move> ::= <step> | <step> <move>\)
- \(<step> ::= up | down\)
CORRESPONDING DCG GRAMMAR

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

The cat scares the mouse.

det | noun | verb | det | noun

noun_phrase | noun_phrase

verb_phrase

sentence
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].

?- sentence(A,B).
?- trace,sentence([the,cat,scares,the,mouse],A).

(c) 2011 P.Fodor (CS Stony Brook) & Wikipedia
A SIMPLE NATURAL LANGUAGE DCG

sentence(N)  -->  noun_phrase(N), verb_phrase(N).
verb_phrase(N)  -->  verb(N), noun_phrase.
noun_phrase(N)  -->  determiner, noun(N).
determiner  -->  [ the].
noun(singular)  -->  [ cat].
noun(plural)  -->  [ cats].
noun(singular)  -->  [ mouse].
verb(singular)  -->  [ scares].
verb(plural)  -->  [ scare].
?- sentence(A,B).
?- trace,sentence([the,cat,scares,thes,mouse],A).
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!
A SIMPLE NATURAL LANGUAGE DCG

sentence(N,s(X,Y))  -->  noun_phrase(N,X),
                    verb_phrase(N,Y).

verb_phrase (N,vp(X,Y)) -->  verb(N,X),
                        noun_phrase(_,Y).

noun_phrase(N,np(X,Y))  -->  determiner(X), noun(N,Y).

determiner(det(the))  -->  [ the].
noun(singular,noun(cat))  -->  [ cat].
noun(plural,noun(cats))  -->  [ cats].
noun(singular,noun(mouse))  -->  [ mouse].
verb(singular,verb(scares))  -->  [ scares].
verb(plural,verb(scare))  -->  [ scare].

?- sentence(A,B,C,D).
A SIMPLE NATURAL LANGUAGE DCG

sentence(N) --> noun_phrase(N),
               verb_phrase(N).

verb_phrase(N) --> verb(N),
                noun_phrase(_).

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

determiner --> [ the].
noun(singular) --> [ cat].
noun(plural) --> [ cats].
noun(singular) --> [ mouse].
verb(singular) --> [ scares].
verb(plural) --> [ scare].

?- sentence(A,B,C).
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).

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

<table>
<thead>
<tr>
<th>Sentence</th>
<th>Formalised meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>John paints</code></td>
<td>$\text{paints}(\text{john})$</td>
</tr>
<tr>
<td><code>John likes Annie</code></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
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)”

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 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{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

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.
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)

```prolog
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)))))
```

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