Logic programming IV

Henrik Boström
Stockholm University

- Definite clause grammars (DCGs)

Grammar rules

```
<s> ::= a b
<s> ::= a <s> b
```

Backus-Naur form

<s> is a non-terminal symbol (can be replaced)
a and b are terminal symbols (can not be replaced)

```
<s>
a <s> b
a a <s> b b
a a a b b b
```

Definite Clause Grammar (DCG)

```
s --> [a,b].
s --> [a], s, [b].
```

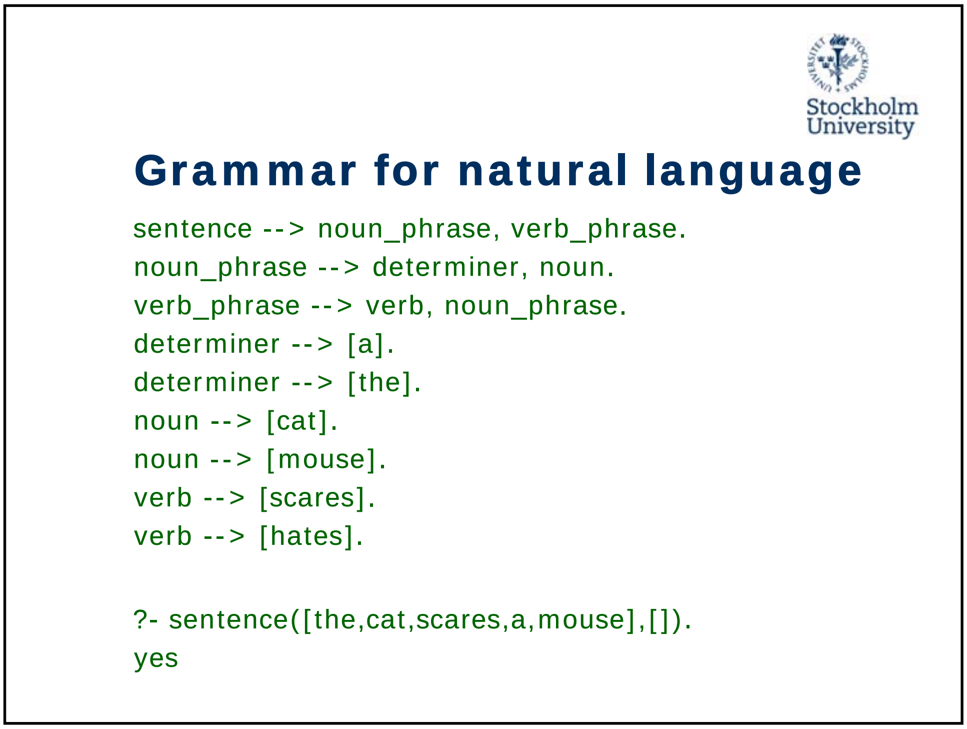
The string a a b b can be represented by the lists:

[a,a,b,b] and []
[a,a,b,b,c] and[c]
[a,a,b,b,1,0,1] and[1,0,1]

```
?- s([a,a,b,b],[]).
yes
?- s([a,a,b],[]).
no
```

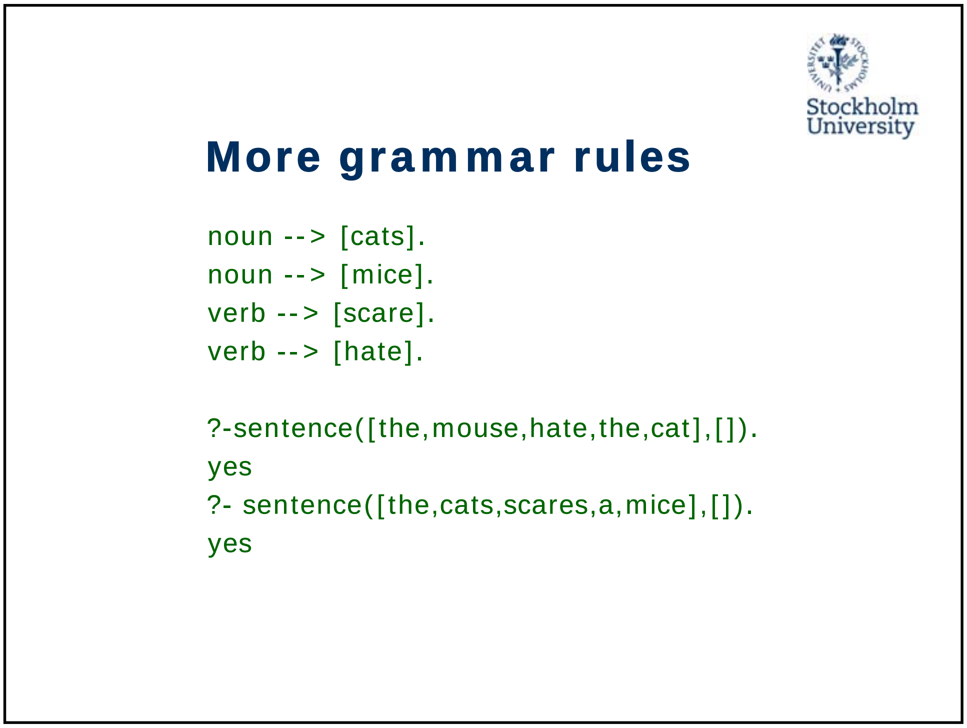
DCGs are translated into clauses

```
n --> n1, n2, ..., nn.
n(List1,Rest):-
        n1(List1,List2), n2(List2,List3), ..., nn(Listn,Rest).

n --> [t1], n2, [t3], n4, [t5].
n([t1|List1],Rest):-
        n2(List1,[t3|List2]), n4(List2,[t5|Rest]).

n --> [t1], n2, [t3], n4, [t5], {p6}.
n([t1|List1],Rest):-
        n2(List1,[t3|List2]), n4(List2,[t5|Rest]), p6.
```



Grammar for natural language

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

?- sentence([the,cat,scares,a,mouse],[]).
yes
```



More grammar rules

```
noun --> [cats].
noun --> [mice].
verb --> [scare].
verb --> [hate].

?-sentence([the,mouse,hate,the,cat],[]).
yes
?- sentence([the,cats,scares,a,mice],[]).
yes
```

Stockholm University

Grammar rules with arguments

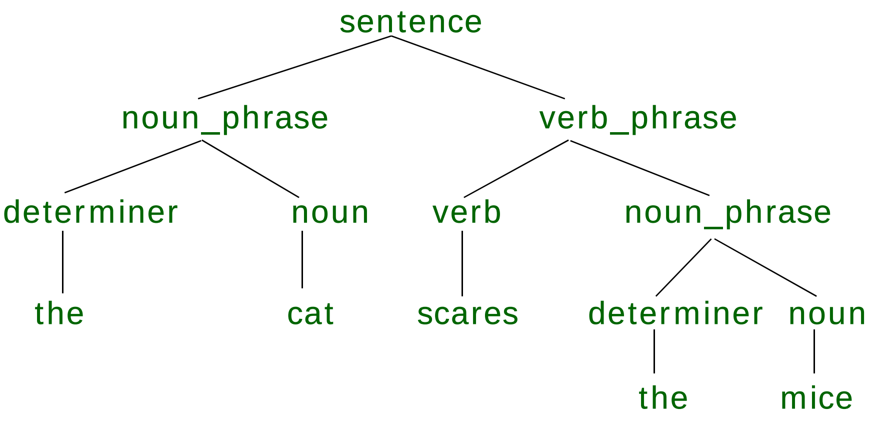
```
sentence --> noun_phrase(No), verb_phrase(No).
noun_phrase(No) --> determiner(No), noun(No).
verb_phrase(No1) --> verb(No1), noun_phrase(No2).
determiner(singular) --> [a].
determiner(_) --> [the].
noun(singular) --> [cat].
noun(plural) --> [cats].
...
verb(singular) --> [scares].
verb(plural) --> [scare].
...
```

Stockholm University

Grammar rules with arguments

```
?- sentence([the,mice,hate,the,cat],[]).
yes
?- sentence([the,mice,hates,the,cat],[]).
no
?- sentence([the,What,hates,the,cat],[]).
What = cat;
What = mouse;
no
```

Parse tree


sentence

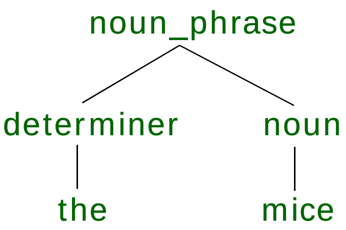
noun_phrase verb_phrase

determiner noun verb noun_phrase

the cat scares determiner noun

the mice

Parse tree in Prolog


noun_phrase

determiner noun

the mice

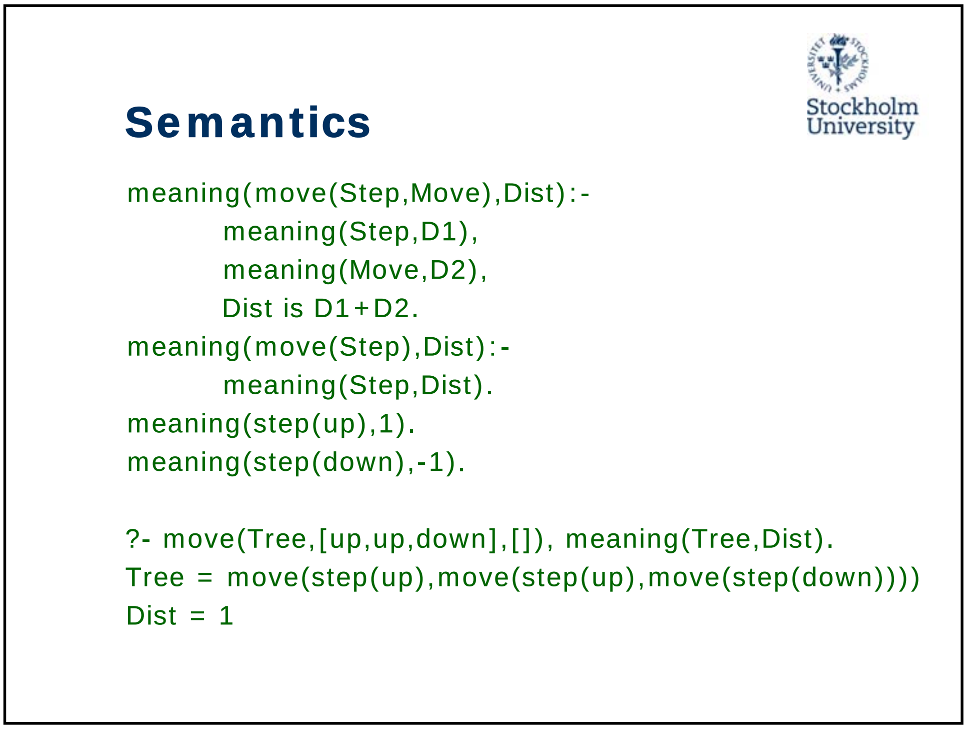
noun_phrase(determiner(the),noun(mice))

Grammar with parse tree

```
sentence(sentence(NP,VP)) -->
        noun_phrase(No,NP), verb_phrase(No,VP).

noun_phrase(No,noun_phrase(Det,Noun)) -->
        determiner(No,Det), noun(No,Noun).

verb_phrase(No1,verb_phrase(Verb,NP)) -->
        verb(No1,Verb), noun_phrase(No2,NP).

determiner(singular,determiner(a)) --> [a].
determiner(_,determiner(the)) --> [the].

noun(singular,noun(cat)) --> [cat].
noun(plural,noun(cats)) --> [cats].
...
```

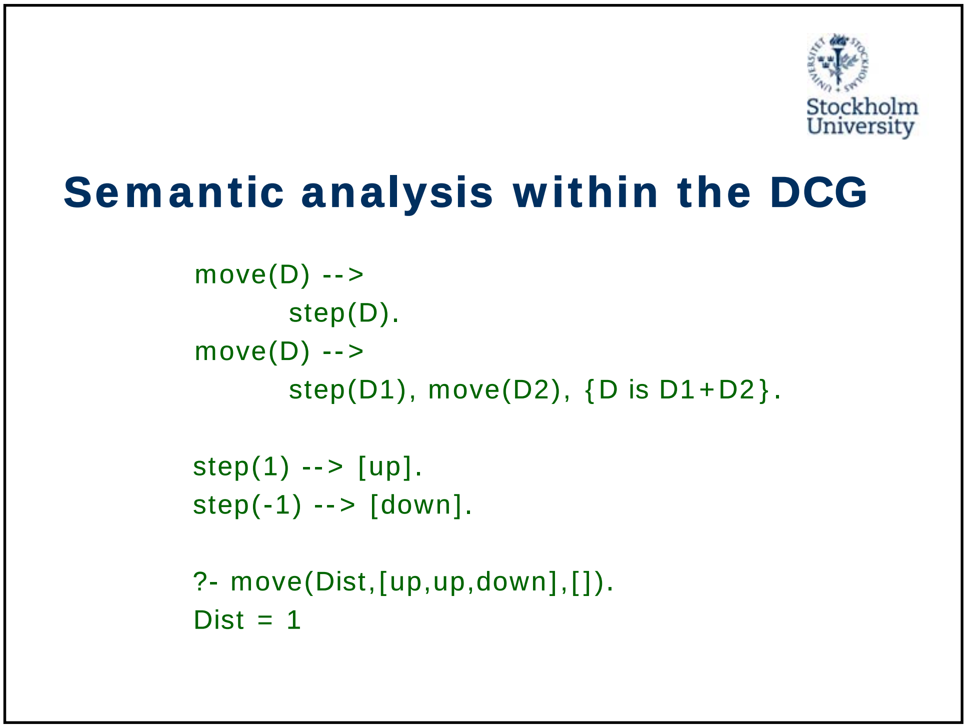
Grammar with parse tree

```
move(move(Step)) --> step(Step).
move(move(Step,Move)) --> step(Step), move(Move).

step(step(up)) --> [up].
step(step(down)) --> [down].
```



Semantics

```
meaning(move(Step,Move),Dist):-
        meaning(Step,D1),
        meaning(Move,D2),
        Dist is D1+D2.
meaning(move(Step),Dist):-
        meaning(Step,Dist).
meaning(step(up),1).
meaning(step(down),-1).

?- move(Tree,[up,up,down],[]), meaning(Tree,Dist).
Tree = move(step(up),move(step(up),move(step(down))))
Dist = 1
```



Semantic analysis within the DCG

```
move(D) -->
        step(D).
move(D) -->
        step(D1), move(D2), {D is D1+D2}.

step(1) --> [up].
step(-1) --> [down].

?- move(Dist,[up,up,down],[]).
Dist = 1
```

Stockholm University

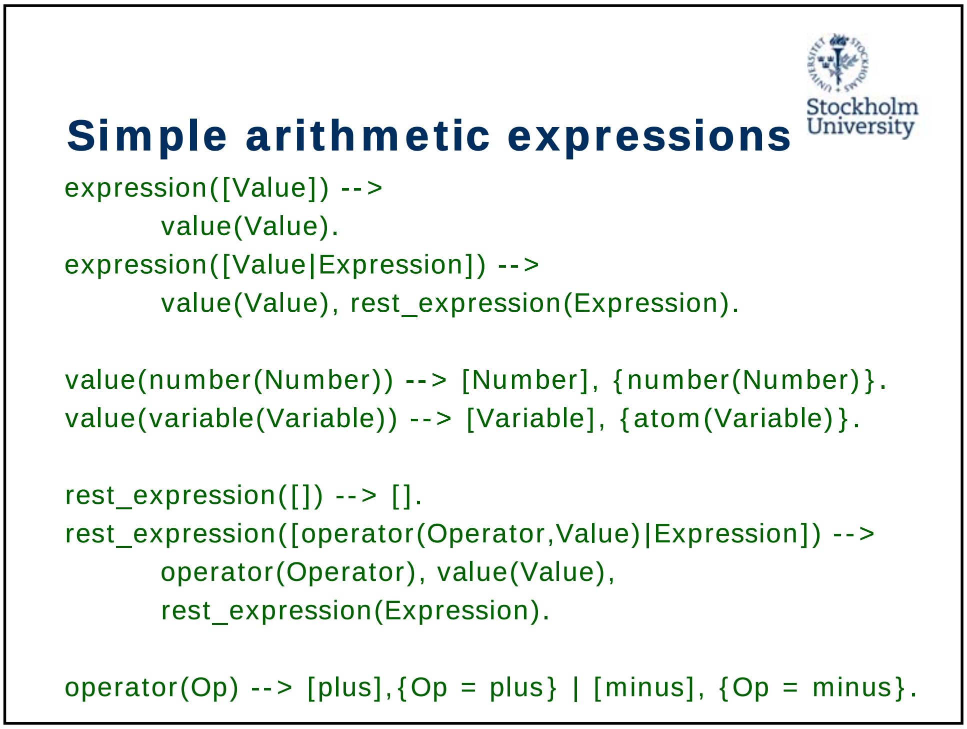
Semantic analysis within the DCG

```
move(D1,D2) -->
        step(D3), {D2 is D1+D3}.
move(D1,D2) -->
        step(D3), {D4 is D1+D3}, move(D4,D2).

step(1) --> [up].
step(-1) --> [down].

?- move(0,Dist,[up,up,down],[]).
Dist = 1
```

Stockholm University

Simple arithmetic expressions

```
?- expression(E,[3,minus,4,plus,2],[]) ,
   evaluate(E,[],0,V).

E = [number(3),operator(minus,number(4)),
      operator(plus,number(2))]
V = 1

?- expression(E,[a,minus,b,plus,2,minus,c],[]) ,
   evaluate(E,[(a,7),(b,6)],0,V).

E = [variable(a),operator(minus,variable(b)),
      operator(plus,number(2)), operator(minus,variable(c))]
V = 3
```



Simple arithmetic expressions

```
expression([Value]) -->
        value(Value).
expression([Value|Expression]) -->
        value(Value), rest_expression(Expression).

value(number(Number)) --> [Number], {number(Number)}.
value(variable(Variable)) --> [Variable], {atom(Variable)}.

rest_expression([]) --> [].
rest_expression([operator(Operator,Value)|Expression]) -->
        operator(Operator), value(Value),
        rest_expression(Expression).

operator(Op) --> [plus],{Op = plus} | [minus], {Op = minus}.
```