Declarative Programming

5: natural language processing using DCGs
Definite clause grammars: context-free grammars in Prolog

one non-terminal on left-hand side

non-terminal defined by rule produces syntactic category

sentences generated by grammar are lists of terminals:
the lazy turtle sleeps, Achilles beats the turtle, the rapid turtle beats Achilles
Definite clause grammars: parse trees for generated sentences

- **sentence**
  - **noun_phrase**
    - **article**
      - *the*
    - **adjective**
      - *rapid*
    - **noun**
      - *turtle*
  - **verb_phrase**
    - **transitive_verb**
      - *beats*
    - **noun_phrase**
      - **proper_noun**
        - *Achilles*
Definite clause grammars: top-down construction of parse trees

Start with NT and repeatedly replace NTS on right-hand side of an applicable rule until sentence is obtained as a list of terminals.
DCG rules and Prolog clauses: equivalence

**sentence**

[the, rapid, turtle, beats, achilles]

**grammar rule**

sentence → noun_phrase, verb_phrase

verb → [sleeps]

**equivalent Prolog clause**

sentence(S) :-
    noun_phrase(NP),
    verb_phrase(VP),
    append(NP, VP, S).

verb([sleeps]).

*S is a sentence if some first part belongs to the noun_phrase category and some second part to the verb_phrase category*

**parsing**

?- sentence([the, rapid, turtle, beats, achilles])
DCG rules and Prolog clauses: 
built-in equivalence without append/3

Grammar rule:
sentence --> noun_phrase, verb_phrase

Equivalent Prolog clause:
sentence(L,L0) :- noun_phrase(L,L1), verb_phrase(L1,L0).

 Parsing:
?- phrase(sentence, L)

L consists of a sentence followed by L0

starting non-terminal

built-in meta-predicate calling sentence(L,[])
DCG rules and Prolog clauses: summary and expressivity

- non-terminals can have arguments
- goals can be put into the rules
- no need for deterministic grammars
- a single formalism for specifying syntax, semantics parsing and generating
Expressivity of DCG rules: non-terminals with arguments - plurality

sentence --> noun_phrase(N), verb_phrase(N).
noun_phrase(N) --> article(N), noun(N).
verb_phrase(N) --> intransitive_verb(N).
article(singular) --> [a].
article(singular) --> [the].
article(plural) --> [the].
noun(singular) --> [turtle].
noun(plural) --> [turtles].
intransitive_verb(singular) --> [sleeps].
intransitive_verb(plural) --> [sleep].

phrase(sentence, [a, turtle, sleeps]). % yes
phrase(sentence, [the, turtles, sleep]). % yes
phrase(sentence, [the, turtles, sleeps]). % no

arguments unify to express plurality agreement
Expressivity of DCG rules: non-terminals with arguments - parse trees

sentence(s(NP,VP)) --> noun_phrase(NP), verb_phrase(VP).
noun_phrase(np(N)) --> proper_noun(N).
noun_phrase(np(Art,Adj,N)) --> article(Art), adjective(Adj), noun(N).
noun_phrase(np(Art,N)) --> article(Art), noun(N).
verb_phrase(vp(IV)) --> intransitive_verb(IV).
verb_phrase(vp(TV,NP)) --> transitive_verb(TV), noun_phrase(NP).
article(art(the)) --> [the].
adjective(adj(lazy)) --> [lazy].
adjective(adj(rapid)) --> [rapid].
proper_noun(pn(achilles)) --> [achilles].
noun(n(turtle)) --> [turtle].
intransitive_verb(iv(sleeps)) --> [sleeps].
transitive_verb(tv(beats)) --> [beats].

?-phrase(sentence(T), [achilles, beats, the, lazy, turtle])
T = s(np(pn(achilles)),
  vp(tv(beats),
    np(art(the),
      adj(lazy),
      n(turtle))))
Expressivity of DCG rules: goals in rule bodies

\[\text{numeral}(N) \rightarrow n1\_999(N).\]
\[\text{numeral}(N) \rightarrow n1\_9(N1), \text{[thousand]}, n1\_999(N2), \{N \text{ is } N1*1000+N2\}.\]
\[\text{n1}\_999(N) \rightarrow n1\_99(N).\]
\[\text{n1}\_999(N) \rightarrow n1\_9(N1), \text{[hundred]}, n1\_99(N2), \{N \text{ is } N1*100+N2\}.\]
\[\text{n1}\_99(N) \rightarrow n0\_9(N).\]
\[\text{n1}\_99(N) \rightarrow n10\_19(N).\]
\[\text{n1}\_99(N) \rightarrow n20\_90(N).\]
\[\text{n1}\_99(N) \rightarrow n20\_90(N1), n1\_9(N2), \{N \text{ is } N1+N2\}.\]
\[\text{n0}\_9(0) \rightarrow \[].\]
\[\text{n0}\_9(N) \rightarrow n1\_9(N).\]
\[\text{n1}\_9(1) \rightarrow \text{[one]}.\]
\[\text{n1}\_9(2) \rightarrow \text{[two]}.\]
\[\text{n10}\_19(10) \rightarrow \text{[ten]}.\]
\[\text{n10}\_19(11) \rightarrow \text{[eleven]}.\]
\[\text{n20}\_90(20) \rightarrow \text{[twenty]}.\]
\[\text{n20}\_90(30) \rightarrow \text{[thirty]}.\]

\(X\_Y(N)\) if \(N\) is a number in \([X..Y]\).
Interpretation of natural language: 
syntax and semantics

Syntax:
- sentence --> determiner, noun, verb_phrase
- sentence --> proper_noun, verb_phrase
- verb_phrase --> [is], property
- property --> [a], noun
- property --> [mortal]
- determiner --> [every]
- proper_noun --> [socrates]
- noun --> [human]

Semantics:
- [every, human, is, mortal]

Interpret a sentence: assign a clause to it

mortal(X):- human(X)

Represents meaning of sentence
Interpretation of natural language: interpreting sentences as clauses (I)

**proper_noun(socrates) --> [socrates]**
the meaning of the proper noun ‘Socrates’ is the term socrates

**property(X=>mortal(X)) --> [mortal]**.
the meaning of the property ‘mortal’ is a mapping from terms to literals containing the unary predicate mortal

**operator X=>L: term X is mapped to literal L**

**verb_phrase(M) --> [is], property(M).**
the meaning of a phrase (proper noun - verb) is a clause with empty body and of which the head is obtained by applying the meaning of the verb phrase to the meaning of the proper noun

**sentence([(L:-true)]) --> proper_noun(X), verb_phrase(X=>L).**

**singleton clause list, cf. determiner ‘some’**

?-phrase(sentence(C), [socrates,is,mortal]).
C = [(mortal(socrates):- true)]
Interpretation of natural language: interpreting sentences as clauses (II)

sentence(C) → determiner(M1,M2,C),
          noun(M1),
          verb_phrase(M2).

noun(X=human(X)) → [human].

determiner(X=\(B\), X=\(H\), \([H:- B]\)) → [every].

?-phrase(sentence(C), [every,human,is,mortal])
C = \([\text{mortal}(X):- \text{human}(X)\])

the meaning of a determined sentence with determiner ‘every’ is a clause with the same variable in head and body
determiner(\text{sk} \Rightarrow \text{H1}, \text{sk} \Rightarrow \text{H2},
\text{(H1:}\neg\text{-true}), (\text{H1:}\neg\text{-true})) \rightarrow \text{[some]}.

?-\text{phrase(sentence(C), [some, humans, are, mortal])}
C = [(\text{human(sk)}:\neg\text{-true}), (\text{mortal}\text{(sk)}:\neg\text{-true})]

the meaning of a determined sentence with determiner ‘some’ are two clauses about the same individual (i.e., skolem constant)
Interpretation of natural language: relational nature illustrated

?-phrase(sentence(C),S).
C = human(X):-human(X)
S = [every,human,is,a,human];
C = mortal(X):-human(X)
S = [every,human,is,mortal];
C = human(socrates):-true
S = [socrates,is,a,human];
C = mortal(socrates):-true
S = [socrates,is,mortal];

?-phrase(sentence(Cs), [D,human,is,mortal]).
D = every, Cs = [(mortal(X):-human(X))];
D = some, Cs = [(human(sk):-true),(mortal(sk):-true)]
Interpretation of natural language: complete grammar with plurality agreement

:- op(600, xfy, '=>').

sentence(C) --> determiner(N, M1, M2, C), noun(N, M1),
verb_phrase(N, M2).
sentence([(L:- true)]) --> proper_noun(N, X),
verb_phrase(N, X=>L).
verb_phrase(s, M) --> [is], property(s, M).
verb_phrase(p, M) --> [are], property(p, M).
property(N, X=>mortal(X)) --> [mortal].
property(s, M) --> noun(s, M).
property(p, M) --> noun(p, M).
determiner(s, X=>B , X=>H, [(H:- B)]) --> [every].
determiner(p, sk=>H1, sk=>H2, [(H1 :- true),(H2 :- true)]) -->[some].
proper_noun(s, socrates) --> [socrates].
noun(s, X=>human(X)) --> [human].
noun(p, X=>human(X)) --> [humans].
noun(s, X=>living_being(X)) --> [living], [being].
noun(p, X=>living_being(X)) --> [living], [beings].
Interpretation of natural language:
shell for building up and querying rule base

**Grammar for queries**

question(Q) --> [who], [is], property(s,X=>Q)
question(Q) --> [is], proper_noun(N,X), property(N,X=>Q)
question((Q1,Q2)) --> [are], [some], noun(p,sk=>Q1), property(p,sk=>Q2)

**Shell**

nl_shell(RB) :- get_input(Input), handle_input(Input,RB).

handle_input(stop,RB) :- !.
handle_input(show,RB) :- !, show_rules(RB), nl_shell(RB).
handle_input(Sentence,RB) :- phrase(sentence(Rule),Sentence), nl_shell([Rule|RB]).
handle_input(Question,RB) :- phrase(question(Query),Question), prove_rb(Query,RB),! transform(Query,Clauses), phrase(sentence(Clauses),Answer), show_answer(Answer), nl_shell(RB).
show_answer('no'), nl_shell(RB).

add new rule

question that can be solved
transform instantiated query (conjuncted literals) to list of clauses with empty body
generate nl
show_rules([]).
show_rules([R|Rs]) :-
    phrase(sentence(R),Sentence),
    show_answer(Sentence),
    show_rules(Rs).
get_input(Input) :-
    write('? '),read(Input).
show_answer(Answer) :-
    write('!' ),write(Answer), nl.

show_answer(Answer) :- write('!' ),nl.
get_input(Input) :- write('?' ),read(Input).

transform((A,B),[(A:-true)|Rest]):-!,
    transform(B,Rest).
transform(A,[(A:-true)]).
Interpretation of natural language:
shell for building up and querying rule base - interpreter

prove(true,RB) :- !.
prove((A,B),RB) :- !,
    prove(A,RB),prove(B,RB).
prove(A,RB) :-
    find_clause((A:-B),RB),
    prove(B,RB).

find_clause(C,[R|Rs]) :-
    copy_element(C,R).
find_clause(C,[R|Rs]) :-
    find_clause(C,Rs).

copy_element(X,Ys) :- element(X1,Ys),
    copy_term(X1,X).

copy_term(+In, -Out)
Create a version if In with renamed (fresh) variables and unify it to Out.

handy when storing rule base in list
finds a clause in the rule base, but without instantiating its variables (rule can be used multiple times, rules can share variables)
Interpretation of natural language:
shell for building up and querying rule base - example

? [every,human,is,mortal]
? [socrates,is,a,human]
? [who,is,mortal]
! [socrates,is,mortal]
? [some,living,beings,are,humans]
? [are,some,living,beings,mortal]
! [some,living,beings,are,mortal]

built-in repeat/1 succeeds indefinitely

shell :- repeat, get_input(X), handle_input(X).
handle_input(stop) :- !.
handle_input(X) :- /* handle */, fail.

possible improvement: apply idiom of failure-driven loop to avoid memory issues

causes backtracking to repeat literal