text processing in Prolog

Yves Lespérance
Adapted from Peter Roosen-Runge

Prolog representation for parsing text

- want to parse natural language text
- one way to represent grammar rules:
  sentence --> noun_phrase, verb_phrase.
  stands for
  sentence(X):- append(Y,Z,X), noun_phrase(Y), verb_phrase(Z).
  determiner --> [the].
  stands for
  determiner([the]).
- must guess how to split the sequence, inefficient; let constituent parsers decide

a better representation

- sentence(S0,S):- noun_phrase(S0,S1), verb_phrase(S1,S).
- determiner([the | S],S).
- 1st argument is sequence to parse and 2nd argument is what is left after removing it
- Rule means "there is a sentence between S0 and S if …"
- ?-sentence([the, boy, drinks, the, juice], []). succeeds
- ?-noun_phrase([the, boy, drinks, the, juice], R). succeeds with R = [drinks, the, juice]

definite clause grammar (DCG) notation

sentence --> noun_phrase,verb_phrase.
stands for
sentence(S0,S):- noun_phrase(S0,S1), verb_phrase(S1,S).
determiner --> [the].
stands for
determiner([the|S],S).

enforcing constraints between constituents

- suppose we want to enforce number agreement
- can add extra argument to pass this info between constituents
- noun_phrase(N) --> determiner(N), noun(N).
- noun(singular) --> [boy].
- noun(plural) --> [boys].
- determiner(singular) --> [a].
- ?- noun_phrase(N,[a, boys],[]). fails
- ?- noun_phrase(N,[a, boy],[]). succeeds with N = singular

returning a parse tree or interpretation

- Extra arguments can also be used to return a parse tree or interpretation
- noun_phrase(np(D,N)) --> determiner(D), noun(N).
- determiner(determiner(a)) --> [a].
- noun(noun(boy)) --> [boy].
- ?- noun_phrase(PT,[a, boy],[]). succeeds with PT = np(determiner(a),noun(boy))

adding extra tests

- can invoke predicates for tests or interpretation by putting between {}
- don't match input tokens
- e.g. accessing a lexicon
- noun(N,noun(W)) --> [W],

  {is_noun(W,N)}.
- is_noun(boy,singular).

grammar writing tips

- good grammars:
  - are very modular
  - achieve broad coverage with small number of rules
- collecting a corpus of examples can help design and test grammar
- identify patterns built out of certain types of constituents

Prolog & text processing

- Prolog good for analyzing and generating text
- parsing involves *pattern-matching*
- text & parse-trees are *recursive* data structures
- text patterns involve *many alternatives*, backtracking is helpful
- *steadfast* predicates can analyze and generate