Language processing

One of the original applications for Prolog was language processing. Language processing involves finding the meaning of a sequence of words. For example, the meaning of the sentence “The capital of Sweden is Stockholm” could be represented as capital(Stockholm, Sweden). An intermediate step to finding the meaning involves identifying the grammatical structure of the sentence. In this example sentence, the grammatical structure is a noun phrase (“The capital of Sweden”), a verb (“is”), and proper noun (“Stockholm”). Identifying the grammatical structure is called parsing.

We will look at how Prolog can be used for parsing, and take a glimpse at how to write a program that understands natural language.

Prolog grammar rules

Prolog has a special kind of rule that can be used to write down the grammar for a language. Grammar rules are written using −→ instead of :−, but are otherwise similar to ordinary rules.

For example, here is a simplistic grammar for English written in Prolog:

sentence −→ noun_phrase, verb_phrase.
noun_phrase −→ determiner, noun.
noun_phrase −→ proper_name.
verb_phrase −→ transitive_verb, noun_phrase.
verb_phrase −→ intransitive_verb.
determiner −→ [every].
determiner −→ [a].
noun −→ [man].
noun −→ [woman].
proper_name −→ [john].
transitive_verb −→ [loves].
intransitive_verb −→ [lives].

The words in square brackets match words from the input sentence. To use the grammar, you need to provide Prolog with the starting grammar rule and the input sentence. The built-in phrase predicate is used to do this:

?- phrase( sentence, [every, man, loves, a, woman] ).
yes

This tells us that “every man loves a woman” is a syntactically correct sentence, according to the rules of the grammar. Every such sentence must start with a noun_phrase and end with a verb_phrase. The noun
phrase in this sentence is “every man” and the verb phrase is “loves a woman”. “Every man” is a determiner (“every”) followed by a noun ("man"); this sequence is matched by the first rule for verb_phrase. “Loves a woman” is a verb phrase. It starts with “loves”, which is a transitive_verb, followed by the noun_phrase “a woman”.

Building parse trees

Just testing whether a sentence conforms to the syntax of the language is not especially interesting. We would like to extract the grammatical structure of the sentence. To do this, we augment each rule with a term that describes the rule along with the component parts:

sentence( s(NP,VP) ) ---> noun_phrase(NP), verb_phrase(VP).
noun_phrase( np(D,N) ) ---> determiner(D), noun(N).
verb_phrase( vp(V,NP) ) ---> transitive_verb(V), noun_phrase(NP).
verb_phrase( vp(V) ) ---> intransitive_verb(V).
determiner( every ) ---> [every].
determiner( a ) ---> [a].
noun( man ) ---> [man].
noun( woman ) ---> [woman].
proper_name( john ) ---> [john].
transitive_verb( loves ) ---> [loves].
intransitive_verb( lives ) ---> [lives].

Now we can ask

?- phrase( sentence(S), [every, man, loves, a, woman] ).
S = s(np(every, man), vp(loves, np(a, woman)));

This grammar can also be used in reverse, to generate a sentence from a parse term:

?- phrase( sentence(s(np(every, man), vp(loves, np(a, woman)))), W).
W = [every, man, loves, a, woman]

Further processing can then be performed on the parse term, to perform an appropriate action.

— John Hamer, 21 June 2012