

Very basic NLP with LPA PROLOG

Dimitri PISSARENKO
dimitri.pissarenko@gmx.net

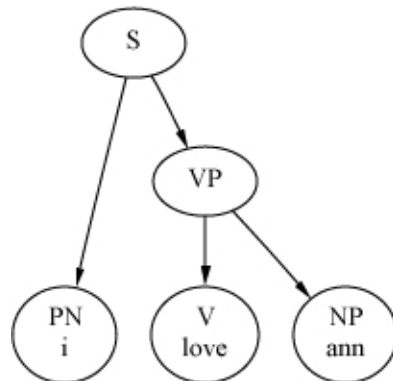
February 22, 2002

1 Sample sentence

Throughout the text, a very simple sentence will be used, which equals to the PROLOG list `[i,love,ann]`. Its structure corresponds to

$S \rightarrow PN VP .$
 $PN \rightarrow i$
 $VP \rightarrow V NP$
 $V \rightarrow love$
 $NP \rightarrow ann$

or, as a syntax tree



2 Doing NLP with PROLOG

LPA PROLOG has built-in support for NLP. This means, that LPA PROLOG system relieves the programmer from implementing the actual parsing routines. The only 2 things the programmer has to do in order to use the LPA PROLOG NLP support are

- to define the grammar

- to define the lexicon.

Grammar describes the structure of the sentences. The lexicon is the collection of morphemes (words) that can be recognised by the parser being developed. In our simple case, the grammar is

$$\begin{aligned} S &\rightarrow \text{PN VP} . \\ \text{VP} &\rightarrow \text{V NP} \end{aligned}$$

and the lexicon

$$\begin{aligned} \text{PN} &\rightarrow \text{i} \\ \text{V} &\rightarrow \text{love} \\ \text{NP} &\rightarrow \text{ann} \end{aligned}$$

When we write these rules in the DCG notation (definite clause grammar), we obtain

```
s(s(Pn,Vp)) --> pn(Pn),vp(Vp).
vp(vp(V,Np)) --> v(V),np(Np).
```

for the grammar and

```
pn(pn(i)) --> [i].
v(v(love)) --> [love].
np(np(ann)) --> [ann].
```

for the lexicon. For further details on the DCG notation in general and LPA PROLOG built-in NLP support see [Matthews, 1998, pp185–214] and [Shalfield, 2001, pp56–70] respectively.

3 Parsing

Having defined the grammar and lexicon, we can now try to parse our phrase and generate the corresponding parse tree. This is done by entering the following predicate:

```
| ?- phrase(s(X), [i, love, ann]).
```

Upon executing this command,

```
| ?- phrase(s(X), [i, love, ann]).
X = s(pn(i), vp(v(love), np(ann)))
```

```
| ?-
```

is printed out. This value of X is the parse tree in the PROLOG notation.

4 Displaying trees

Due to the pitiable circumstance that most of us are humans and not PROLOG cyborgs, the parse tree extracted in previous step is not very easy to comprehend. Contrary to PROLOG cyborgs, we prefer graphical representations to lists. For visualising trees one may use the `ppt/1` predicate by Gustaf Neumann (see appendix C). The `ppt/1` has only one argument (arity of one) that contains the tree to be displayed and prints the tree to the standard output stream. So, entering

```
| ?- phrase(s(X), [i, love, ann]), ppt(X).
```

produces the following output

```
| ?- phrase(s(X), [i, love, ann]), ppt(X).
      s
      |
      .----- .
      |         |
      pn        vp
      |         |
      .         .----- .
      |         |         |
      i         v         np
                |         |
                .         .
                |         |
                love    ann
```

```
X = s(pn(i), vp(v(love), np(ann)))
```

```
| ?-
```

References

- C. Matthews. *An Introduction to Natural Language Processing Through Prolog*. Addison Wesley Longman, 1998.
- R. Shalfield. *WIN-PROLOG Programming Guide*. Logic Programming Associates Ltd, 2001.

A Who is Ann?

In this section you see a second example of a PROLOG program that parses a sentence. The sentence used here is `[ann, means, artificial, neural, network]`.

```
1 /**
2  * who_is_ann.pl
3  *
4  * Demonstration of LPA PROLOG built-in NLP support
```

```

5  *
6  * Dimitri PISSARENKO, Universtity of Derby in Austria
7  *
8  * February 21, 2002
9  *
10 **/
11
12 ?- consult('ppt.pl'). /* "include" the predicates for displaying trees */
13
14 /* Definition of the grammar for the sentence [ann,means,artificial,neural,network] */
15 s(s(N,Vp)) -->
16   n(N),vp(Vp).
17 vp(vp(V,Np)) --> v(V), np(Np).
18 np(np(Adj0,Adj1,N)) --> adj(Adj0), adj(Adj1), n(N).
19
20 /* Definition of the lexicon for the sentence [ann,means,artificial,neural,network] */
21 n(n(ann)) --> [ann].
22 v(v(means)) --> [means].
23 adj(adj(artificial)) --> [artificial].
24 adj(adj(neural)) --> [neural].
25 n(n(network)) --> [network].
26
27 /* Predicate start invokes parsing the sentence and printing the */
28 /* resulting parse tree                                     */
29
30 start :-
31   phrase(s(X),[ann,means,artificial,neural,network]),
32   write('Parse tree (for PROLOG cyborgs): '), write(X), nl,
33   write('Parse tree:'), nl, ppt(X).
34
35 ?-start.

```

B i_love_ann.pl

```

1 /**
2  * i_love_ann.pl
3  *
4  * Demonstration of LPA PROLOG built-in NLP support
5  *
6  * Dimitri PISSARENKO, Universtity of Derby in Austria
7  *
8  * February 21, 2002
9  *
10 **/
11
12 ?- consult('ppt.pl'). /* "include" the predicates for displaying trees */
13
14 /* Definition of the grammar for the sentence [i,love,ann] */

```

```

15 s(s(Pn,Vp)) -->
16   pn(Pn), /* pronoun */
17   vp(Vp). /* verb phrase */
18
19 vp(vp(V,Np)) -->
20   v(V), /* verb */
21   np(Np). /* noun phrase */
22
23 /* Definition of the lexicon for the sentence [i,love,ann] */
24 pn(pn(i)) --> [i].
25 v(v(love)) --> [love].
26 np(np(ann)) --> [ann].
27
28 /* Predicate start invokes parsing the sentence and printing the */
29 /* resulting parse tree                                     */
30
31 start :-
32   phrase(s(X),[i,love,ann]),
33   write('Parse tree (for PROLOG cyborgs): '), write(X), nl,
34   write('Parse tree:'), nl, ppt(X).
35
36 ?-start.

```

C ppt.pl

```

1 /*
2   ppt/1,
3   printing Prolog terms in a tree layout for typewriter output.
4
5   Written in Spring 1985 -- Gustaf Neumann
6
7   (c)1985 Gustaf Neumann, Wirtschaftsuniversitaet Wien,
8   Augasse 2-6, 1090 Vienna, Austria *222/31 336-4533.
9   email: neumann@wu-wien.ac.at or neumann@awiwuw11.bitnet
10
11  Permission is granted to use, copy and distribute this program as long
12  (1) this note remains intact,
13  (2) no fees are charged and
14  (3) no further restrictions are imposed.
15
16  The following predicates are not defined within this program:
17  - length(List,Length),
18  - tab(Exp),
19  - append(A,B,AB).
20  Do not try to print infinite trees :-).
21
22  To show, what this program does, issue the goal: examples.
23 */
24 ?-op(100,xfy,:).
25

```

```

26 examples:- example(X), ppt(X), nl, nl, write(X), nl,
27     wait_for_input, fail.
28 examples.
29
30 example(sin(alpha)/cos(beta-phi)+cos(beta)*tan(360-alpha)).
31 example(buch(titel(wirtschaftsinformatik1),autor(hans_robert, hansen))).
32 example((a:-b,c,d)).
33 %example((ppt(X,Y):-Body):- clause(ppt(X,Y),Body).
34 example(sentence(np(proper(gustaf)),vp(v(likes),np(proper(prolog)))).
35 example(sentence(np(det(that),n(man),rel(that,vp(iv(whistle))),
36     vp(tv(tunes),np(det(nil),n(pianos),rel(nil)))).
37 example(wirtschaftsinformatik(leitung(hans_robert),
38     sekretariat(virly,anita),
39     assistenten(lore,rony,goeha,gu,margret,andy,stessi))).
40
41
42 /*****
43 *           top level predicate ppt/1           *
44 *****/
45 ppt(Term):- ppt(Term,arc).
46 ppt(Term,Arc) :-
47     number_vars(Term,0,_),      /* ground all variables in Term */
48     {pos(Term,Pos,C,0-Right)},  /* compute hor. positions of nodes */
49     {inv([Pos],[_]:_,H:T,s)},  /* invert structure for printing */
50     posdiff(-(72-Right)//2,0,Tab), /* compute hor. tab for centering */
51     {print_tree(H:T,[C],Tab,Arc)}.
52                                     /* print tree in line printer mode */
53
54 /*****
55 *           Compute Positions of Nodes           *
56 *****/
57 pos(Head,t(Head,Rel,L,[],O)-[], Nc, NO-Nn):- /* leaf node */
58     atomic(Head), !,
59     string_length(Head,L), Nn is NO+L,
60     Rel is L//2,          /* middle of the node */
61     Nc is (NO+Nn)//2.    /* center over node */
62 pos(X,t(Head,Rel,L,Centers,Adj)-A, Nc, NO-N2):- /* non-leaf node */
63     X =.. [Head|Args],
64     pos_list(Args,A,Centers,NO-N1),
65     string_length(Head,L), posdiff(N1-NO,L,Error),
66     Adj is (Error+((N1-NO) mod 2))//2,
67     N2 is N1+Error,
68     Rel is L//2,          /* middle of the node */
69     Nc is (NO+N2)//2.
70
71 pos_list([], [], [], N-N).
72 pos_list([H], [A], [Center], N-N1) :- !, pos(H,A,Center,N-N1).
73 pos_list([H|T], [A|Args], [C|Centers], NO-Nn):-
74     pos(H, A, C, NO-N1),
75     N2 is N1+2, pos_list(T,Args,Centers,N2-Nn).

```

```

76
77 string_length(X,L):- atomic(X), name(X,S), length(S,L).
78
79 posdiff(Expr,L,Adj):- Adj is L-Expr, Adj > 0, !.
80 posdiff(_,_ ,0).
81
82 /*****
83
84 *                invert tree                *
85 *****/
86 inv([Node-Sons|Brothers],List:Deep,[Node|List1]:Deep2,_):-
87     inv(Brothers,List:Deep,List1:Deep1,b),
88     inv(Sons,Deep1,Deep2,s).
89 inv([],[]: [], [],s).
90 inv([],[]: [], []:_ ,b).
91 inv([],E,E,_).
92
93 /*****
94 *                print tree                *
95 *****/
96 print_tree(Node:Deep, Centers, Tab, Arc) :-
97     tab(Tab), print_list(Node,0,Centers,Cd), nl,
98     {( Arc == noarc
99     ;   Deep == []
100    ;   tab(Tab), marks(Centers,Node,0),
101        tab(Tab), horarc(Node,0,_), nl,
102        tab(Tab), marks(Cd,0)
103    }},
104     print_tree(Deep,Cd,Tab,Arc).
105 print_tree([],[],_ ,_).
106
107 print_list([t(H,Rel,L,Cd,Adj)|R], P0, [C|Centers], Ca) :-
108     P is C-Rel, tab(P-P0), write(H), Pn is P+L,
109     print_list(R,Pn,Centers,Cr),
110     add_to(Cd,Adj,Cda), {append(Cda,Cr,Ca)}.
111 print_list([],_ , [], []).
112
113 /*****
114 *                draw arcs                *
115 *****/
116 marks([], [],_):- nl.
117 marks([H|T],[t(_,_ ,_ , [] ,_ )|R],E) :- !, tab(H-E), write(' '),marks(T,R,H+1).
118 marks([H|T],[_ |R], E) :- tab(H-E), write('|'),marks(T,R,H+1).
119
120 marks([],_):- nl.
121 marks([H|T],E) :- tab(H-E), write('|'), marks(T,H+1).
122
123 horarc([], A,A).
124 horarc([t([],_ ,_ ,_ ,_ )|R],P,P2) :- !, horarc(R,P,P2).
125 horarc([t(_,_ ,_ ,Cd,Adj)|R],P,P2) :- line(Cd,Adj,P,P1), horarc(R,P1,P2).

```

```

126
127 line([], _,E,P) :- P is E.
128 line([H], A,E,P) :- !, tab(H+A-E), write(' '), P is H+A+1.
129 line([H|T],A,E,P) :- tab(H+A-E), write(' '), line_([H|T],A,H+A+1,P).
130
131 line_([], _,E,P) :- P is E.
132 line_([H], A,E,P) :- line_to(H+A-E), P is H+A+1.
133 line_([_,T|Tt],A,E,P) :- line_to(T+A-E), write(' '), line_([T|Tt],A,T+A+1,P).
134
135 line_to(Exp) :- L is Exp, line_to_(L,'-').
136 line_to_(L,_) :- L < 1.
137 line_to_(L,C) :- L >= 1, write(C), L1 is L-1, line_to_(L1,C).
138
139 add_to([],_, []).
140 add_to([H|T],A,[Ha|Ta]) :- Ha is H+A, add_to(T,A,Ta).
141
142 /*****
143 *          misc utility predicates          *
144 *****/
145 {G} :- G,!.
146
147 wait_for_input :- get0(_).
148 %wait_for_input :- system([clrscrn,more]).
149
150 number_vars(Term,NO,N1) :-
151     var(Term), !,
152     name(NO,Digits), name('V',[C]),
153     name(Term,[C|Digits]),
154     N1 is NO+1.
155 number_vars(Term,NO,NO) :-
156     atomic(Term), !.
157
158 number_vars(Term,NO,N1) :-
159     Term =.. [_|Args],
160     number_list(Args,NO,N1).
161
162 number_list([],NO,NO).
163 number_list([H|T],NO,N2) :- number_vars(H,NO,N1), number_list(T,N1,N2).

```