crypto_nlp_i.pro

```
:-consult('io.pro').
:-consult('gv.pro').
:-consult('crypto.pro').
:-consult('combosets.pro').

sentence(Problem) --> simpleproblemcommand(Problem), [.].
sentence(Problem) --> randomproblemcommand(Problem), [.].
sentence(Problem) --> simpleproblemquery(Problem),[?].
simpleproblemcommand(Problem) -->
        [use], numberzzz(N1,N2,N3,N4,N5), [to], [make], goal(G),

{Problem=problem(number(N1,N2,N3,N4,N5),goal(G)),establishCryptoProblem(number
s(N1,N2,N3,N4,N5),goal(G))}.
simpleproblemcommand(Problem) -->
        [write], goal(G), [in], [terms], [of],numberzzz(N1,N2,N3,N4,N5),

{Problem=problem(number(N1,N2,N3,N4,N5),goal(G)),establishCryptoProblem(number
s(N1,N2,N3,N4,N5),goal(G))}.
randomproblemcommand(Problem) -->
        [use], [whatever], [to], [make], [whatever],
        {generateRandomCryptoProblem, valueOf(problem,Problem)}.
simpleproblemquery(Problem) -->
        [can], [you], [make], goal(G), separator,numberzzz(N1,N2,N3,N4,N5),

{Problem=problem(number(N1,N2,N3,N4,N5),goal(G)),establishCryptoProblem(number
s(N1,N2,N3,N4,N5),goal(G))}.
separator --> [from].
separator --> [with].
goal(G) --> number(G).
numberzzz(N1,N2,N3,N4,N5) --> number(N1), [and], number(N2), [and], number(N3),
[and], number(N4), [and], number(N5).
numberzzz(N1,N2,N3,N4,N5) --> number(N1), number(N2), number(N3), number(N4),
[and], number(N5).
numberzzz(1,2,3,4,5) --> [the], [first], [five], [positive], [numbers].
numberzzz(0,1,2,3,4) --> [numbers], [zero], [through], [four].
numberzzz(1,2,3,4,5) --> [numbers], [one], [through], [five].
numberzzz(2,3,4,5,6) --> [numbers], [two], [through], [six].
```

```
numberzzz(3,4,5,6,7) --> [numbers], [three], [through], [seven].
numberzzz(4,5,6,7,8) --> [numbers], [four], [through], [eight].
numberzzz(5,6,7,8,9) --> [numbers], [five], [through], [nine].
numberzzz(1,3,5,7,9) --> [the], [odd], [numbers].
numberzzz(N1,N1,N1,N1,N1) --> [five], pluralnumber(N1).
numberzzz(N1,N1,N1,N1,N2) --> [four], pluralnumber(N1), [and], [one], number(N2).
numberzzz(N1,N2,N2,N2,N2) --> [one], number(N1),[and], [four], pluralnumber(N2).
numberzzz(N1,N1,N2,N2,N2) --> [two], pluralnumber(N1), [and], [three],
pluralnumber(N2).
numberzzz(N1,N1,N1,N2,N2) --> [three], pluralnumber(N1), [and], [two],
pluralnumber(N2).
numberzzz(N1,N1,N2,N2,N3) --> [two], pluralnumber(N1), [and], [two],
pluralnumber(N2), [and], [one], number(N3).
numberzzz(N1,N2,N2,N3,N3) --> [one], number(N1),[and], [two], pluralnumber(N2),
[and], [two], pluralnumber(N3).
numberzzz(N1,N1,N2,N3,N3) --> [two], pluralnumber(N1), [and], [one], number(N2),
[and], [two], pluralnumber(N3).
number(0) --> [zero].
number(1) --> [one].
number(2) --> [two].
number(3) --> [three].
number(4) --> [four].
number(5) --> [five].
number(6) --> [six].
number(7) --> [seven].
number(8) --> [eight].
number(9) --> [nine].
pluralnumber(0) --> [zeros].
pluralnumber(1) --> [ones].
pluralnumber(2) --> [twos].
pluralnumber(3) --> [threes].
pluralnumber(4) --> [fours].
pluralnumber(5) --> [fives].
pluralnumber(6) --> [sixes].
pluralnumber(7) --> [sevens].
pluralnumber(8) --> [eights].
pluralnumber(9) --> [nines].

%Testing the augmented DCG -- the parser.
```

```
%-----------------------------------------

parser :-
read_sentence(S),
sentence(Problem,S,[]),
write(Problem),nl,
parser.

parser :-
write('Not a sentence...'),nl,
parser.

%The Interpreter
%-----------------------------------------


interpreter :-
read_sentence(S),
sentence(Problem,S,[]),
try_to_solve_exhaustively(Problem),
interpreter.

interpreter :-
write('Not a sentence ...'),nl,
interpreter.

%Exhaustive problem solver
%----------------------------------------

try_to_solve_exhaustively(Problem) :-
eraseProblem,
eraseSolution,
Problem = problem(numbers(N1,N2,N3,N4,N5),goal(G)),
establishCryptoProblem(numbers(N1,N2,N3,N4,N5),goal(G)),
tryToSolveProblemDecompositionally,
solution(Solution),
displayResult(Solution),nl.

try_to_solve_exhaustively(_) :-
```

```
write('No solution to this one!'),nl.

tryToSolveProblemDecompositionally :-
problem(numbers(N1,N2,N3,N4,N5),goal(G)),
crypto(N1,N2,N3,N4,N5,G,Expression),
recordSolution(Expression).
tryToSolveProblemDecompositionally.
```