```
%%% td0.pl
%%%
%%% Top Down Grammar Interpreter
%%% This grammar (borrowed in part from some of Dale Gerdemann's
%%% teaching materials) is ambiguous. Try parsing
%%%    [i,saw,her,duck]
%%% for example.

:- use_module( library(lists), [append/3] ).

%%%---( Grammar )------------------------------------------

rule( s, [np,vp] ).
rule( np, [det,n] ).
rule( np, [pn] ).
rule( vp, [v] ).
rule( vp, [v,np] ).
rule( vp, [v,s] ).

word( the, det ).
word( villain, n ).
word( hero, n ).
word( duck, n ).
word( pursue, v ).
word( saw, v ).
word( duck, v ).
word( her, det ).
word( i, pn ).
word( her, pn ).

%%%---( Recognizer )---------------------------------------

rec( String ):-
        td( [s], String ).

%%% accept
td( [], [] ).
%%% scan
td( [PreTermCat|RestStack], [Word|RestString] ):-
        word( Word, PreTermCat ),
        td( RestStack, RestString ).
%%% expand
td( [NonTermCat|RestStack], String ):-
        rule( NonTermCat, RightHandSide ),
        multipush( RightHandSide, RestStack, NewStack ),
        td( NewStack, String ).



multipush( RightHandSide, RestStack, NewStack ):-
        append( RightHandSide, RestStack, NewStack ).
```

Figure 1: Program `td0.pl`: Top-down recognizer.

```prolog
%%% bu.pl
%%%
%%% Bottom up (shift-reduce) grammar interpreter.
%%%
%%% Remember that things on the stack are in the
%%% reverse order from the right hand sides of
%%% the rules!

:- use_module( library(lists), [append/3,reverse/2] ).

%%%---( Grammar )------------------------------------------

rule( s, [np,vp] ).
rule( np, [det,n] ).
rule( np, [pn] ).
rule( vp, [v] ).
rule( vp, [v,np] ).
rule( vp, [v,s] ).

word( the, det ).
word( villain, n ).
word( hero, n ).
word( duck, n ).
word( pursue, v ).
word( saw, v ).
word( duck, v ).
word( her, det ).
word( i, pn ).
word( her, pn ).

%%%---( Recognizer )---------------------------------------

rec( String ):-
        bu( [], String ).

%%% accept
bu( [s], [] ).
%%% reduce
bu( StackIn, String ):-
        multipop_reversed( StackIn, StackOut, RHS ),
        rule( NonTermCat, RHS ),
        bu( [NonTermCat|StackOut], String ).
%%% shift
bu( StackIn, [Word|String] ):-
        word( Word, PreTermCat ), % this is just a shortcut
        bu( [PreTermCat|StackIn], String ).

multipop_reversed( StackIn, StackOut, RHS ):-
        append( RHSReversed, StackOut, StackIn ),
        reverse( RHSReversed, RHS ).
```

Figure 2: Program bu.pl: Bottom-up recognizer.

```
%%% tdparse.pl
%%%
%%% Top Down Parser
%%%

rule( s, [np,vp] ).
rule( np, [det,n] ).
rule( np, [pn] ).
rule( vp, [v] ).
rule( vp, [v,np] ).
rule( vp, [v,s] ).

word( the, det ).
word( villain, n ).
word( hero, n ).
word( duck, n ).
word( pursue, v ).
word( saw, v ).
word( duck, v ).
word( her, det ).
word( i, pn ).
word( her, pn ).


parse( String, Tree ):-
        td( [Tree:s], String ).

%%% accept
td( [], [] ).
%%% scan
td( [(PreTermCat/Word):PreTermCat|RestStack], [Word|RestString] ):-
        word( Word, PreTermCat ),
        td( RestStack, RestString ).
%%% expand
td( [(NonTermCat/SubTrees):NonTermCat|RestStack], String ):-
        rule( NonTermCat, RightHandSide ),
        multipush( RightHandSide, SubTrees, RestStack, NewStack ),
        td( NewStack, String ).

%%% multipush( RHS, RHS_ParseTrees, StackIn, StackOut )
%%% ------------------------------------------------------
%%% When we push the elements of the RHS onto the stack, we have
%%% to attach their (mostly uninstantiated) parse trees to them.
%%% We also need to collect a list of those sub-parse-trees
%%% which will become the children of the node which we are
%%% currently expanding.
multipush( [], [], Stack, Stack ).
multipush( [Cat|RestCats],      % rule
           [Cat/CSubs|RestSubs], % subtrees of parent category
           Stack0,               % stack in
           [(Cat/CSubs):Cat|Stack1] % stack out
         ):-
        multipush( RestCats, RestSubs, Stack0, Stack1 ).
```

Figure 3: Program `tdparse.pl`: Top-down parser.

```
%%% buparse2.pl
%%%
%%% Bottom up (shift-reduce) parser, plus grammar.
%%%
%%% Version 2: Partially evaluated the stack implementation
%%% (except for multipop/4).
%%%

rule( s, [np,vp] ).
rule( np, [det,n] ).
rule( np, [pn] ).
rule( vp, [v] ).
rule( vp, [v,np] ).
rule( vp, [v,s] ).

word( the, det ).
word( villain, n ).
word( hero, n ).
word( duck, n ).
word( pursue, v ).
word( saw, v ).
word( duck, v ).
word( her, det ).
word( i, pn ).
word( her, pn ).

parse( String, Tree ):-
        bu( [], String, Tree ).

%%% accept
bu( [Tree:s], [], Tree ).
%%% reduce
bu( StackIn, String, Tree ):-
        multipop( RHS, RHSTrees, StackIn, StackMid ),
        rule( NonTermCat, RHS ),
        bu( [(NonTermCat/RHSTrees):NonTermCat|StackMid], String, Tree ).
%%% shift
bu( StackIn, [Word|String], Tree ):-
        word( Word, PreTermCat ), % this is just a shortcut
        bu( [(PreTermCat/Word):PreTermCat|StackIn], String, Tree ).

%%%---( Stack Implementation )-------------------------------------
%%%
%%% This ``coroutines'' the calls to append/3 and reverse/2
%%% that were used in the definition of multipop_reversed/3
%%% in file bu.pl

multipop( CatList, TreeList, Stack0, Stack ):-
        multipop_aux( [], CatList, [], TreeList, Stack0, Stack ).

multipop_aux( CatList, CatList, TreeList, TreeList, Stack, Stack ).
multipop_aux( CL0, CL, TL0, TL, [T:C|S1], S ):-
        multipop_aux( [C|CL0], CL, [T|TL0], TL, S1, S ).
```

Figure 4: Program buparse2.pl: Bottom-up parser.