```
%%%%%%%%%%
%%% NLP %%%
%%%%%%%%%%

%% Parsing %%

sentence(s(VP)) --> verb_phrase(VP).
sentence(s(VP)) --> verb_phrase(VP), dot.


noun_phrase(np(Adjp, Noun)) --> adj_phrase(Adjp),
noun(Noun).
noun_phrase(np(Det,Noun))--> det(Det),noun(Noun).
noun_phrase(np(Noun))--> noun(Noun).
noun_phrase(np(Noun,Conj,NP)) -->
noun(Noun),conj(Conj),noun_phrase(NP).

verb_phrase(vp(Propnoun,Verb, NP)) -->
propnoun(Propnoun),verb(Verb), noun_phrase(NP).
verb_phrase(vp(Propnoun,Verb, Det,NP)) -->
propnoun(Propnoun),verb(Verb),det(Det), noun_phrase(NP).

adj_phrase(adjp(Adj)) --> adj(Adj).
adj_phrase(adjp(Comma,Adjp))--
>comma(Comma),adj_phrase(Adjp).
adj_phrase(adjp(Conj,Adjp)) -->
conj(Conj),adj_phrase(Adjp).
adj_phrase(adjp(Adj, Adjp)) --> adj(Adj), adj_phrase(Adjp).


conj(c(and))-->[and].

verb(v(want)) --> [want].

det(d(a)) --> [a].

propnoun(pn(i))-->[i].

comma(comma(',')) -->[','].
%%adj(a()) --> [].
adj(a(spicy)) --> [spicy].


%%noun(n()) -->[].
noun(n(food)) -->[food].
```

```
noun(n(american)) -->[american].
noun(n(burger)) -->[burger].
noun(n(wings)) -->[wings].
noun(n(steak)) -->[steak].
noun(n(beef)) -->[beef].
noun(n(chineese)) -->[chineese].

dot --> ['.'].
dot -->[].



%%%%%%%%%%
%%%Utils%%%
%%%%%%%%%%

read_word_list(Ws) :-
    read_line_to_codes(user_input,Cs),
    atom_codes(A,Cs),
    tokenize_atom(A,Ws).

parse():-
read_word_list(IN),sentence(Parse,IN,[]),write(Parse).

%%%%%%%%%%%%%%%%%%
%%% Origin Sets %%%
%%%%%%%%%%%%%%%%%%

%% Origin sets are represented as [type, [hyp1, hyp2, ...]]
where the hyps provide
%% the justification for believing a fact, and the type is
either hyp or der. A hyp
%% type is always associated with a singleton list of hyps
(the term itself). A der
%% type is associated with on or more hypotheses from which
it is derivable. A term
%% may have one or more origin sets (stored in a list).

% Merge a single pair of OSes.
merge_os_pair([_, OS1], [_, OS2], [der, Result]) :-
    append(OS1, OS2, OS3),
    list_to_set(OS3, Result).

% Base case - only one pair to merge.
```

```
merge_os([OS1], [OS2], [Result]) :- merge_os_pair(OS1, OS2,
Result).
% TODO: Multiple OSes to merge.
%merge_os([OS1|Rest1], [OS2|Rest2], Result) :-

to_der_os([], []).
to_der_os([[_,OS]|Rest], [[der,OS]|PartResult]) :-
to_der_os(Rest, PartResult).

% In removing terms you need to remove all of the origin
sets with a specific hyp in them
% but a term may have more than one OS, so this removes
just the offending ones.
remove_oses_with_hyp(_, [], []).
remove_oses_with_hyp(Term, [[_, OS]|Rest], NewOS) :-
    member(Term, OS), !, remove_oses_with_hyp(Term, Rest,
NewOS).
remove_oses_with_hyp(Term, [[Type, OS]|Rest], [[Type,
OS]|NewOS]) :-
    remove_oses_with_hyp(Term, Rest, NewOS).


%%%%%%%%%%%%%%%%%%%%%
%%% Knowledge Base %%%
%%%%%%%%%%%%%%%%%%%%%

%% A knowledge base is a list of the form [[term
originset], [term originset], ...]
%% These rules allow building a KB without doing any
inference.

assert_hyp(Term, OldKB, [[Term, [[hyp, [Term]]]]|OldKB]).

assert_hyps([Term], OldKB, NewKB) :- assert_hyp(Term,
OldKB, NewKB).
assert_hyps([Term|Rest], OldKB, NewKB) :-
    assert_hyps(Rest, OldKB, PartKB),
    assert_hyp(Term, PartKB, NewKB).

%% Removing a hyp from the KB involves removing everything
that has that hyp as one of
%% its origin set members.

unassert_hyp(_, [], []).
```

```
unassert_hyp(Term, [[Term,_]|KBRest], NewKB) :-
unassert_hyp(Term, KBRest, NewKB).
unassert_hyp(Term, [[OtherTerm, OS]|KBRest], NewKB) :-
remove_oses_with_hyp(Term, OS, NewOS),
    (NewOS = [] ->
        unassert_hyp(Term, KBRest, NewKB);
        unassert_hyp(Term, KBRest, PartialKB), NewKB =
[[OtherTerm, NewOS]|PartialKB]).

% When two terms are in the KB which are identical, merge
them.
% - If they have different OSes, merge the OSes.
% - If they have the same OS, do nothing.
merge_term(T, T, T).
merge_term([Term, OS1], [Term, OS2], [Term, NewOS]) :-
merge_os(OS1, OS2, NewOS).


merge_kbs(KB1, KB2, Result) :- append_kbs(KB1, KB2,
Result).

% Appending KBs is a 2-step process. First, just append
them. Then, term-by-term,
% walk through the resulting KB looking for duplicate
terms. Merge them.

append_kbs(KB1, KB2, Result) :-
    append(KB1, KB2, KB3), !,
    merge_duplicates(KB3, Result).

merge_duplicates([[First, FirstOS]|Rest],
[MergeTerm|Result]) :-
    member([First, OtherOS], Rest),
    merge_term([First, FirstOS], [First, OtherOS],
MergeTerm),
    delete(Rest, [First, OtherOS], NewRest),
    merge_duplicates(NewRest, Result).
merge_duplicates([[First, FirstOS]|Rest], [[First,
FirstOS]|Result]) :-
    merge_duplicates(Rest, Result).
merge_duplicates([], []).



%%%%%%%%%%%%%%%%%%%%%
```

```
%%%   Model Code   %%%
%%%%%%%%%%%%%%%%%%%%%

test:-
assert_hyps([if(category(american),not(category(mexican))),
if(category(mexican),not(category(american)))],[],KB).
```