LIN 177 Homework 7

Hardy Jones
999397426
Professor Ojeda
Winter 2015

1. The process uses partial reduplication to create pluralized verbs. The second to last phone is duplicated in the morph to create the plural form.

```
samoan([[m, a], [t, e]], [verb, singular]).
samoan([[n, o], [f, o]], [verb, singular]).
samoan([[g, a], [l, u], [e]], [verb, singular]).
samoan([[t, a], [n, u]], [verb, singular]).
samoan([[a], [l, o], [f, a]], [verb, singular]).
samoan([[t, a], [o], [t, o]], [verb, singular]).
samoan([[a, t], [a], [m, a], [?, i]], [verb, singular]).

samoan(VerbPlural, [verb, plural]) :-
    samoan(VerbSingular, [verb, singular]),
    append(Initial, [Repeat, Last], VerbSingular),
    append(Initial, [Repeat, Repeat, Last], VerbPlural).
```

```
?- samoan(X, Y).
X = [[m, a], [t, e]],
Y = [verb, singular] ;
X = [[n, o], [f, o]],
Y = [verb, singular] ;
X = [[g, a], [l, u], [e]],
Y = [verb, singular] ;
X = [[t, a], [n, u]],
Y = [verb, singular] ;
X = [[a], [l, o], [f, a]],
Y = [verb, singular] ;
X = [[t, a], [o], [t, o]],
Y = [verb, singular] ;
X = [[a, t], [a], [m, a], [?, i]],
Y = [verb, singular] ;
X = [[m, a], [m, a], [t, e]],
Y = [verb, plural] ;
X = [[n, o], [n, o], [f, o]],
Y = [verb, plural] ;
X = [[g, a], [l, u], [l, u], [e]],
Y = [verb, plural] ;
X = [[t, a], [t, a], [n, u]],
Y = [verb, plural] ;
X = [[a], [l, o], [l, o], [f, a]],
Y = [verb, plural] ;
X = [[t, a], [o], [o], [t, o]],
Y = [verb, plural] ;
X = [[a, t], [a], [m, a], [m, a], [?, i]],
Y = [verb, plural] ;
false.
```

2. The process uses mutation to create yiddish words from english words. Everything up until the first vowel of the morph is replaced by `schm`.

```
vowel(a).
vowel(e).
vowel(i).
vowel(o).
vowel(u).

english([h, o, u, s, e], [noun]).
english([t, r, o, u, s, e, r, s], [noun]).
english([s, m, o, o, t, h], [noun]).
english([s, p, r, i, n, k, l, e, r], [noun]).
english([a, r, t, i, s, t], [noun]).

yiddish(Yiddish, [noun]) :-
    english(English, [noun]),
    drop_consonants(English, Rest),
    append([s, c, h, m], Rest, Yiddish).

drop_consonants([], []).
drop_consonants([C|Cs], [C|Cs]) :-
    vowel(C).
drop_consonants([C|Cs], Rest) :-
    not(vowel(C)),
    drop_consonants(Cs, Rest).
```

```
?- yiddish(X, Y).
X = [s,c,h,m,o,u,s,e],
Y = [noun] ;
X = [s,c,h,m,o,u,s,e,r,s],
Y = [noun] ;
X = [s,c,h,m,o,o,t,h],
Y = [noun] ;
X = [s,c,h,m,i,n,k,l,e,r],
Y = [noun] ;
X = [s,c,h,m,a,r,t,i,s,t],
Y = [noun] ;
false.
```

3. The instantiator clause should not be changed as it would then have to understand the underlying form of the language. Rather the underlying form should generate possible morphs and the instantiator should restrict to well formed morphs based on the rules of the language.

```
:- ['Programs/Programs/Unix/entailment.swipl'].
:- ['Programs/Programs/Unix/fullproperties.swipl'].

underenglish([P1,P2,P3,P4],[adjective]):-
    phone(P1), not(voi(P1)), not(dnt(P1)), lab(P1),
    phone(P2), mid(P2), not(bck(P2)), not(ctr(P2)),
    phone(P3), snt(P3), not(nas(P3)), not(alv(P3)), pal(P3),
    phone(P4), not(snt(P4)), voi(P4), not(cnt(P4)), alv(P4).

underenglish([P1,P2,P3],[adjective]):-
    phone(P1), not(voi(P1)), dnt(P1), lab(P1),
    phone(P2), mid(P2), not(bck(P2)), not(ctr(P2)),
    phone(P3), not(nas(P3)), alv(P3), pal(P3).

underenglish([P1,P2,P3,P4,P5,P6,P7],[adjective]):-
    phone(P1), not(voi(P1)), cnt(P1), cor(P1), not(sib(P1)),
    phone(P2), hih(P2), not(bck(P2)), not(tns(P2)),
    phone(P3), nas(P3),
    phone(P4), not(voi(P4)), vel(P4),
    phone(P5), not(cns(P5)), not(str(P5)),
    phone(P6), not(snt(P6)), voi(P6), not(cnt(P6)), lab(P6),
    phone(P7), snt(P7), not(nas(P7)), alv(P7), not(pal(P7)).

underenglish([P1,P2,P3,P4,P5,P6],[adjective]):-
    phone(P1), not(voi(P1)), not(cnt(P1)), alv(P1),
    phone(P2), hih(P2), not(bck(P2)), not(tns(P2)),
    phone(P3), not(voi(P3)), not(dnt(P3)), lab(P3),
    phone(P4), hih(P4), not(bck(P4)), not(tns(P4)),
    phone(P5), not(voi(P5)), vel(P5),
    phone(P6), snt(P6), not(nas(P6)), alv(P6), not(pal(P6)).

underenglish([P1,P2,P3,P4,P5,P6,P7,P8], [adjective]):-
    phone(P1), not(nas(P1)), alv(P1), pal(P1),
    phone(P2), not(bck(P2)), tns(P2),
    phone(P3), snt(P3), not(nas(P3)), alv(P3), not(pal(P3)),
    phone(P4), mid(P4), not(bck(P4)), not(ctr(P4)),
    phone(P5), snt(P5), not(nas(P5)), not(alv(P5)), pal(P5),
    phone(P6), not(voi(P6)), not(cnt(P6)), alv(P6),
    phone(P7), hih(P7), not(bck(P7)), not(tns(P7)),
    phone(P8), not(snt(P8)), voi(P8), not(cnt(P8)), alv(P8).

underenglish([P1,P2,P3,P4,P5,P6],[adjective]):-
    phone(P1), not(voi(P1)), not(cnt(P1)), pal(P1),
    phone(P2), mid(P2), not(bck(P2)), not(ctr(P2)),
    phone(P3), snt(P3), not(nas(P3)), not(alv(P3)), pal(P3),
    phone(P4), nas(P4),
    phone(P5), not(snt(P5)), voi(P5), cnt(P5), pal(P5),
    phone(P6), not(snt(P6)), voi(P6), not(cnt(P6)), alv(P6).

underenglish([P1,P2,P3,P4],[adjective]):-
    phone(P1), not(voi(P1)), vel(P1),
    phone(P2), snt(P2), not(nas(P2)), alv(P2), not(pal(P2)),
    phone(P3), not(bck(P3)), tns(P3),
    phone(P4), not(nas(P4)), alv(P4), pal(P4).

underenglish([P1,P2],[affix,negative]):-
    phone(P1),ctr(P1),str(P1),
    phone(P2),nas(P2).

underenglish(A,[adjective,negative]):-
    underenglish(B,[affix,negative]),
    underenglish(C,[adjective]),
    %% If the first is a consonant, then it doesn't matter what we do.
    nth1(1, C, C1),
    phone(C1), cns(C1),
    append(B,C,A).

underenglish(A,[adjective,negative]):-
    underenglish(B,[affix,negative,vowel]),
    underenglish(C,[adjective]),
    %% If he first is not a consonant,
    %% Then we need to use the negative vowel affix.
    nth1(1, C, C1),
    phone(C1), not(cns(C1)),
    append(B,C,A).

%% available
underenglish([P1,P2,P3,P4,P5,P1,P7,P1,P5], [adjective]):-
    phone(P1), not(cns(P1)), not(str(P1)),
    phone(P2), lab(P2), ant(P2), cnt(P2), voi(P2),
    phone(P3), mid(P3), not(bck(P3)), not(ctr(P3)),
    phone(P4), hih(P4), not(bck(P4)), not(tns(P4)),
    phone(P5), snt(P5), not(nas(P5)), alv(P5), not(pal(P5)),
    phone(P7), not(snt(P7)), voi(P7), not(cnt(P7)), lab(P7).

%% ending
underenglish([P1,P2,P3,P1,P5], [adjective]):-
    phone(P1), hih(P1), not(bck(P1)), not(tns(P1)),
    phone(P2), nas(P2), alv(P2), not(pal(P2)),
    phone(P3), not(snt(P3)), voi(P3), not(cnt(P3)), alv(P3),
    phone(P5), nas(P5), vel(P5).

%% even
underenglish([P1,P2,P3,P4], [adjective]):-
    phone(P1), not(bck(P1)), tns(P1),
    phone(P2), lab(P2), ant(P2), cnt(P2), voi(P2),
    phone(P3), hih(P3), not(bck(P3)), not(tns(P3)),
    phone(P4), nas(P4), alv(P4), not(pal(P4)).

%% ordered
underenglish([P1,P2,P3,P4,P2,P3], [adjective]):-
    phone(P1), str(P1), bck(P1), mid(P1),
    phone(P2), not(nas(P2)), alv(P2), pal(P2),
    phone(P3), not(snt(P3)), voi(P3), not(cnt(P3)), alv(P3),
    phone(P4), hih(P4), not(bck(P4)), not(tns(P4)).

%% uttered
```

```
underenglish([P1,P2,P3,P4,P5], [adjective]):-
    phone(P1), ctr(P1), str(P1),
    phone(P2), not(voi(P2)), not(cnt(P2)), alv(P2),
    phone(P3), hih(P3), not(bck(P3)), not(tns(P3)),
    phone(P4), not(nas(P4)), alv(P4), pal(P4),
    phone(P5), not(snt(P5)), voi(P5), not(cnt(P5)), alv(P5).

%% Only for vowel negations.
underenglish([P1,P2],[affix,negative,vowel]):-
    phone(P1), ctr(P1), str(P1),
    phone(P2), nas(P2), alv(P2), not(pal(P2)).

english(A,B):-
    underenglish(A,B),
    (nas(N), cns(C), nextto(N,C,A)) => homorganic(N,C).

homorganic(A,B):-
    lab(A) <=> lab(B),
    dnt(A) <=> dnt(B),
    alv(A) <=> alv(B),
    pal(A) <=> pal(B),
    vel(A) <=> vel(B).
```

```
?- english(X, [adjective, negative]).

X = [ʌ,m,p,e,j,d] ;

X = [ʌ,n,t,ɪ,p,ɪ,k,l] ;

X = [ʌ,ŋ,k,l,i,ɹ] ;

X = [ʌ,ɱ,f,e,ɹ] ;

X = [ʌ,n̜,θ,ɪ,ŋ,k,ə,b,l] ;

X = [ʌ,ɲ,č,e,j,ɲ,ʒ,d] ;

X = [ʌ,n̪,ɹ,i,l,e,j,t,ɪ,d] ;

X = [ʌ,n,ə,v,e,ɪ,l,ə,b,ə,l] ;

X = [ʌ,n,ɪ,n,d,ɪ,ŋ] ;

X = [ʌ,n,i,v,ɪ,n] ;

X = [ʌ,n,o,ɹ,d,ɪ,ɹ,d] ;

X = [ʌ,n,ʌ,t,ɪ,ɹ,d] ;

false.
```

4.

```
:- ['Programs/Programs/Unix/entailment.swipl'].

vow(P) :- name(P,[121]).
vow(P) :- name(P,[248]).
vow(P) :- name(P,[230]).
vow(P) :- name(P,[105]).
vow(P) :- name(P,[101]).
vow(P) :- name(P,[117]).
vow(P) :- name(P,[111]).
vow(P) :- name(P,[097]).

per(P) :- name(P,[121]).
per(P) :- name(P,[248]).
per(P) :- name(P,[230]).
per(P) :- name(P,[117]).
per(P) :- name(P,[111]).
per(P) :- name(P,[097]).

bck(P) :- name(P,[117]).
bck(P) :- name(P,[111]).
bck(P) :- name(P,[097]).

hih(P) :- name(P,[121]).
hih(P) :- name(P,[105]).
hih(P) :- name(P,[117]).

low(P) :- name(P,[230]).
low(P) :- name(P,[097]).

underfinnish([c,V1,c,V2]) :-
    vow(V1),
    vow(V2).

underfinnish(UnderFinnish) :-
    underfinnish(Stem, [stem]),
    last(Stem, V1),
    per(V1), (low(V1) ; not(hih(V1)), not(low(V1))),
    per(V2), low(V2),
    append(Stem, [s, s, V2], UnderFinnish).

underfinnish([t, a, l, o], [stem]).
underfinnish([k, y, l, æ], [stem]).

finnish(A):-
    underfinnish(A),
    (per(B), per(C), member(B,A), member(C,A)) => (bck(B) <=> bck(C)).
```

```
?- finnish(X).
X = [c,y,c,y] ;
X = [c,y,c,ø] ;
X = [c,y,c,æ] ;
X = [c,y,c,i] ;
X = [c,y,c,e] ;
X = [c,ø,c,y] ;
X = [c,ø,c,ø] ;
X = [c,ø,c,æ] ;
X = [c,ø,c,i] ;
X = [c,ø,c,e] ;
X = [c,æ,c,y] ;
X = [c,æ,c,ø] ;
```

```
X = [c,æ,c,æ] ;
X = [c,æ,c,i] ;
X = [c,æ,c,e] ;
X = [c,i,c,y] ;
X = [c,i,c,ø] ;
X = [c,i,c,æ] ;
X = [c,i,c,i] ;
X = [c,i,c,e] ;
X = [c,i,c,u] ;
X = [c,i,c,o] ;
X = [c,i,c,a] ;
X = [c,e,c,y] ;
X = [c,e,c,ø] ;
X = [c,e,c,æ] ;
X = [c,e,c,i] ;
X = [c,e,c,e] ;
X = [c,e,c,u] ;
X = [c,e,c,o] ;
X = [c,e,c,a] ;
X = [c,u,c,i] ;
X = [c,u,c,e] ;
X = [c,u,c,u] ;
X = [c,u,c,o] ;
X = [c,u,c,a] ;
X = [c,o,c,i] ;
X = [c,o,c,e] ;
X = [c,o,c,u] ;
X = [c,o,c,o] ;
X = [c,o,c,a] ;
X = [c,a,c,i] ;
X = [c,a,c,e] ;
X = [c,a,c,u] ;
X = [c,a,c,o] ;
X = [c,a,c,a] ;
X = [t,a,l,o,s,s,a] ;
X = [k,y,l,æ,s,s,æ] ;
false.
```

5.
```
front(i).
front(e).

high(i).
high(u).

asturian([p, V, l], [stem]) :- front(V).
asturian([k, V, s], [stem]) :- front(V).

asturian(Asturian, [noun, count]) :-
    asturian(Stem, [stem]),
    member(Vowel, Stem),
    front(Vowel), high(Vowel),
    append(Stem, [u], Asturian).
```

```
asturian(Asturian, [noun, mass]) :-
    asturian(Stem, [stem]),
    member(Vowel, Stem),
    front(Vowel), not(high(Vowel)),
    append(Stem, [o], Asturian).
```

```
?- asturian(X, Y), member(noun, Y).
X = [p,i,l,u],
Y = [noun,count] ;
X = [k,i,s,u],
Y = [noun,count] ;
X = [p,e,l,o],
Y = [noun,mass] ;
X = [k,e,s,o],
Y = [noun,mass] ;
false.
```

Extra Credit It will not make a difference for the outcome of the grammar, but it will change how we implement the grammar. We still have to ensure exclusive disjunction, we just use `vib(r)` to formulate it.

As the footnote on page 179 states, we would have to ensure exactly what the change from condition (c) to (d) states.

We create the first grammar as so:

```
:- ['Programs/Programs/Unix/entailment.swipl'].

liq(l).
liq(r).

vib(r).

latin([m, o, r], [noun]).
latin([m, o, l], [noun]).

latin([a, R, i, s], [affix]) :- liq(R).

latin(Latin, [adjective]) :-
    latin(Noun, [noun]),
    Affix = [a, R, i, s],
    latin(Affix, [affix]),
    last(Noun, L),
    not(vib(R) <=> vib(L)),
    append(Noun, Affix, Latin).
```

```
?- latin(X, [adjective]).
X = [m,o,r,a,l,i,s] ;
X = [m,o,l,a,r,i,s] ;
false.
```

And the can extend to the second grammar by making a `dissimilate` predicate that handles the cases as so:

```
:- ['Programs/Programs/Unix/entailment.swipl'].

liq(l).
liq(r).

vib(r).

latin([m, o, r], [noun]).
latin([m, o, l], [noun]).
%% Add a couple of nouns.
latin([s, p, i, r, i, t, u], [noun]).
latin([a, n, e, c, d, o, t], [noun]).

latin([a, R, i, s], [affix]) :- liq(R).

latin(Latin, [adjective]) :-
    latin(Noun, [noun]),
    Affix = [a, R, i, s],
    latin(Affix, [affix]),
    last(Noun, L),
    dissimilate(L, R),
    append(Noun, Affix, Latin).

dissimilate(L, R) :-
    liq(L), liq(R),
    not(vib(L) <=> vib(R)).
dissimilate(L, R) :-
    not(liq(L)),
    not(vib(R)).
```

```
?- latin(X, [adjective]).
X = [m,o,r,a,l,i,s] ;
X = [m,o,l,a,r,i,s] ;
X = [s,p,i,r,i,t,u,a,l,i,s] ;
X = [a,n,e,c,d,o,t,a,l,i,s] ;
false.
```