# Project 1

## *Resolution.*

## 1.a)

My example:

1. Every giraffe or elephant is tall.
2. Every tall animal eats leaves.
3. Every young giraffe will eat leaves.
4. Elephants don't like leaves.
5. Monica is a young giraffe.

Question:

Does Monica eat leaves?

KB:

1. $\forall x((Giraffe(x) \vee Elephant(x)) \rightarrow Tall(x))$
2. $\forall x((Tall(x) \rightarrow (Leaves(x) \vee Tress(x))$
3. $\forall x((Young(x) \wedge Giraffe(x)) \rightarrow Leaves(x))$
4. $\forall x(Elephant(x) \rightarrow \neg Leaves(x))$
5. $Giraffe(Monica) \wedge Young(Monica)$

Question

$Leaves(Monica)$ ?

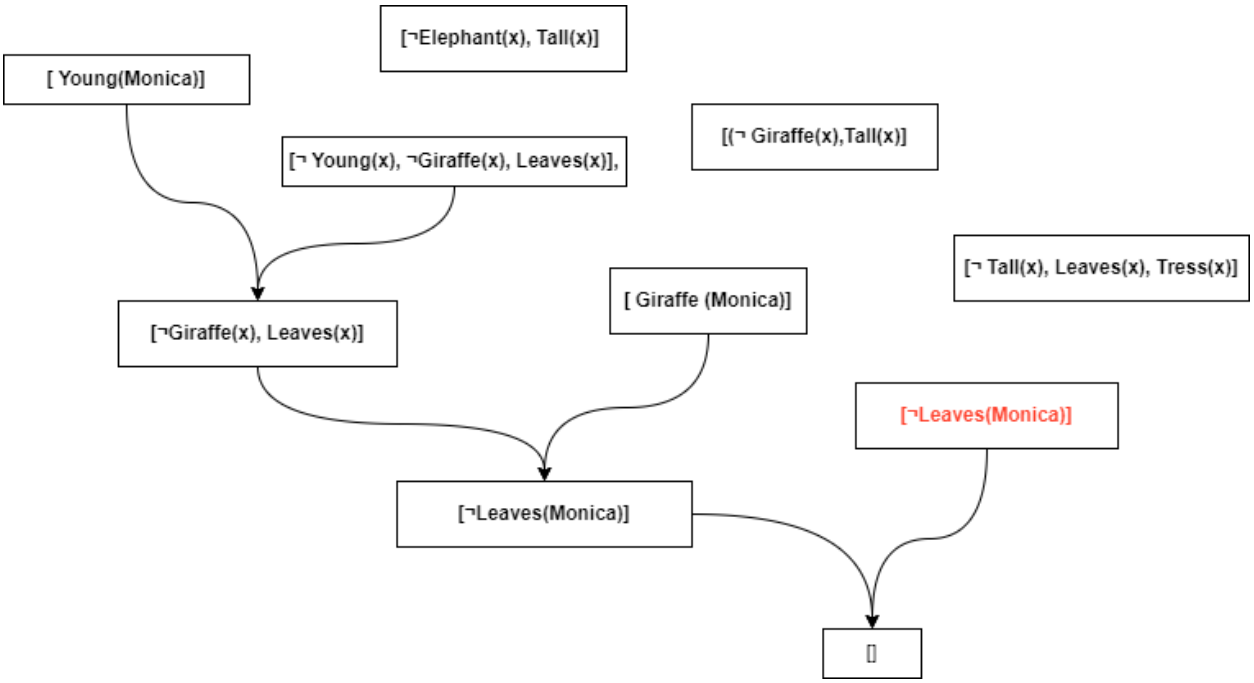
## 1.b)

Translate to CNF:

1. $\forall x((Giraffe(x) \vee Elephant(x)) \rightarrow Tall(x)) = \neg (Giraffe(x) \vee Elephant(x)) \vee Tall(x) = (\neg Giraffe(x) \vee \neg Elephant(x)) \vee Tall(x) \Rightarrow [(\neg Giraffe(x), Tall(x)], [\neg Elephant(x), Tall(x)]$,
2. $\forall x((Tall(x) \rightarrow (Leaves(x) \vee Tress(x)) = \neg Tall(x) \vee (Leaves(x) \vee Tress(x)) \Rightarrow [\neg Tall(x), Leaves(x), Tress(x)]$
3. $\forall x((Young(x) \wedge Giraffe(x)) \rightarrow Leaves(x)) = \neg(Young(x) \wedge Giraffe(x)) \vee Leaves(x) \Rightarrow [\neg Young(x), \neg Giraffe(x), Leaves(x)]$
4. $\forall x(Elephant(x) \rightarrow \neg Leaves(x)) = \neg Elephant(x) \vee \neg Leaves(x) \Rightarrow [\neg Elephant(x), \neg Leaves]$
5. $Giraffe(Monica) \wedge Young(Monica) \Rightarrow [Giraffe(Monica)], [Young(Monica)]$

**CNF** : [(¬ Giraffe(x),Tall(x)],[¬Elephant(x), Tall(x)], [¬ Tall(x), Leaves(x), Tress(x)], [¬ Young(x), ¬Giraffe(x), Leaves(x)], [¬ Elephant(x), ¬ Leaves], [ Giraffe (Monica)],[ Young(Monica)]

**Q**: [Leaves(Monica)]⇒not(Q) ⇒[¬Leaves(Monica)]




# 1.c)

## My example

Input→res([[n(Giraffe),Tall],[n(Elephant),Tall],[n(Tall),Leaves,Tress],[n(Young),n(Giraffe),Leaves],[n(Elephant),n(Leaves)],[Giraffe],[Young]]).
Output→unsatisfiable

The unsatisfiable response is because we also added the negated question to the list.

More examples taken from the course 3.

## Page 19:

Input→ res([[Toddler],[n(Toddler),Child],[n(Child),n(Male),Boy],[n(Child),n(Female),Girl],[Female],[n(Girl)]]).

Output→unsatisfiable

## Page 21:

Input→ res([[Rain,Sun],[n(Sun),Mail],[n(Sleet),Mail],[n(Rain),Mail],[n[n(Rain)]]).
Output→unsatisfiable

## Page 29:

Input→ res([[n(HardWorker)],[n(Student),HardWorker],[n(GradStudent),Student],[GradStudent]]).
Output→unsatisfiable

# 1.d)

```
apply_res_on_file
```

[[n(a),b],[c,d],[n(d),b],[n(b)],[n(c),b],[e],[a,b,n(f),f]]

unsatisfiable

[[n(b),a],[n(a),b,e],[a,n(e)],[n(a)],[e]]

unsatisfiable

[[n(a),b],[c,f],[n(c)],[n(f),b],[n(c),b]]

satisfiable

[[a,b],[n(a),n(b)],[c]]

satisfiable

# *SAT SOLVER  DAVIS PUTNAM.*

_____

input-->[[toddler],[n(toddler),child],[n(child),n(male),boy],[n(infant),child],[n(child),n(female),girl],[female],[girl]]

max appearance:[child/true,girl/true,toddler/true,n(male)/true,female/true]

least balanced:[girl/true,child/true,toddler/true,n(male)/true,female/true]

_____

input-->[[toddler],[n(toddler),child],[n(child),n(male),boy],[n(infant),child],[n(child),n(female),girl],[female],[n(girl)]]

max appearance: false

least balanced: false

_____

input-->[[n(a),b],[c,d],[n(d),b],[n(c),b],[n(b)],[e],[a,b,n(f),f]]

max appearance: false

least balanced: false

_____

input-->[[n(b),a],[n(a),b,e],[e],[a,n(e)],[n(a)]]

max appearance: false

least balanced: false

_____

input-->[[n(a),n(e),b],[n(d),e,n(b)],[n(e),f,n(b)],[f,n(a),e],[e,f,n(b)]]

max appearance:[e/true,n(a)/true,f/true]

least balanced:[f/true,n(a)/true,n(d)/true]

_____

input-->[[a,b],[n(a),n(b)],[n(a),b],[a,n(b)]]

max appearance: false

least balanced: false

_____

The strategies used for selecting the atom we're by the most frequent literal in clauses and the least balanced literal in clauses.

# CODE
# RESOLUTION

```
negate(n(A),A).
negate(A,n(A)).

same(X, X).
same([A| B], [C| D]):-
   A=C,
   same(B,D).

check_same(A,B):-
   sort(A,A_sorted),
   sort(B,B_sorted),
   same(A_sorted,B_sorted).

merge_lists_no_duplicates(L1, L2, R):-
   append(L1,L2,R1),
   list_to_set(R1, R).

subtract_helper(_,[],R1,R1).
subtract_helper([C,D],[[H1,H2]|T],R1,R2):-
   subtract(C,[H1],R3),
   subtract(D,[H2],R4),
   merge_lists_no_duplicates(R3,R4,R5),
   append(R1,[R5],R1_NEW),
   subtract_helper([C,D],T,R1_NEW,R2).

remove_duplicates([],[]).
remove_duplicates(X,Y) :-
   setof(Z,member(Z,X),Y).

comb_of_2(_,[]):-true.
comb_of_2([X|T],[X|Comb]):-
   comb_of_2(T,Comb).
comb_of_2([_|T],[X|Comb]):-
   comb_of_2(T,[X|Comb]).

fnv([],_,R1,R1).
fnv([H|T],L2,R1,R2):-
   negate(H,N),
   member(N,L2),
   append(R1,[[H,N]],R1_NEW),
   fnv(T,L2,R1_NEW,R2).
fnv([H|T],L2,R1,R2):-
   fnv(T,L2,R1,R2).

append_helper(KB,[],KB).
append_helper(KB,TO_APPEND,R):-
```

```
    append(KB,TO_APPEND,R).

resolution([C,D],  R3):-
    fnv(C,D,[],R2),
    subtract_helper([C,D],R2,[],R3).

res_on_all([],KB,KB).
res_on_all([H|T],KB,R):-
    resolution(H,R2),
    append_helper(KB,R2,KB_NEW),
    res_on_all(T,KB_NEW,R).

res(KB):-
    member([], KB),
    write("unsatisfiable"),!.
res(KB):-
    findall([A,B],comb_of_2(KB,[A,B]),ALL_COMB),
    res_on_all(ALL_COMB,KB,R),
    remove_duplicates(R,KB_NEW_NO_DUPLICATES),
    (
    check_same(KB,KB_NEW_NO_DUPLICATES)
    ->
    write("satisfiable");
    res(KB_NEW_NO_DUPLICATES)
    ),!.

apply_res_on_all([]).
apply_res_on_all([H|T]):-
    write(H),
    nl,
    res(H),
    nl,
    apply_res_on_all(T).

read_file(S,[]) :-
    at_end_of_stream(S).
read_file(S,[L|R]) :-
    not(at_end_of_stream(S)),
    read(S,L),
    read_file(S,R).

apply_res_on_file:-
    open('ex1_data.txt', read, S),
    read_file(S, L),
    apply_res_on_all(L),
    close(S).
```

# *SAT SOLVER  DAVIS PUTNAM.*

```
negate(n(A),A).
negate(A,n(A)).

minus(A,B,R):-
   R is A-B.

merge_lists_of_lists([],R1,R1).
merge_lists_of_lists([H|T],R1,R):-
   append(R1,H,R1_NEW),
   merge_lists_of_lists(T,R1_NEW,R).

merge_lists_of_lists_no_dup([],R1,R1).
merge_lists_of_lists_no_dup([H|T],R1,R):-
   append(R1,H,R1_NEW),
   list_to_set(R1_NEW, R1_NEW2),
   merge_lists_of_lists_no_dup(T,R1_NEW2,R).

nooc([],_,0).
nooc([H|T],E,N):-
   H=E,
   nooc(T,E,N1),
   N is N1+1.
nooc([H|T],E,N):-
   nooc(T,E,N).

make_frequency_vec([],_,TMP,TMP).
make_frequency_vec([H|T],LD,TMP,R):-
   nooc(LD,H,OC),
   append(TMP,[[H,OC]],TMP_NEW),
   make_frequency_vec(T,LD,TMP_NEW,R).

make_balance_frequency_vec([],_,TMP,TMP):-!.
make_balance_frequency_vec([H|T],LD,TMP,R):-
   nooc(LD,H,OC),
   negate(H,H_N),
   nooc(LD,H_N,OC_N),
   minus(OC,OC_N,EX),
   abs(EX,EX_ABS),
   append(TMP,[[H,EX_ABS]],TMP_NEW),
   make_balance_frequency_vec(T,LD,TMP_NEW,R),!.

max_app_element([[A,B]],A).
max_app_element([[A,B],[A1,B1]|T],M) :-
   B>=B1,
   max_app_element([[A,B]|T],M),!.
max_app_element([[A,B],[A1,B1]|T],M) :-
   B<B1,
   max_app_element([[A1,B1]|T],M),!.
```

```
all_possible_non_negated_literals([],[]):-!.
all_possible_non_negated_literals([H|T],[H_NEG|R]):-
    negate(H,H_NEG),
    H_NEG\=n(H),
    all_possible_non_negated_literals(T,R).
all_possible_non_negated_literals([H|T],[H|R]):-
    negate(H,H_NEG),
    H_NEG==n(H),
    all_possible_non_negated_literals(T,R).

elem_with_most_app(L,R):-
    merge_lists_of_lists_no_dup(L,[],LND),
    merge_lists_of_lists(L,[],LD),
    make_frequency_vec(LND,LD,[],R2),
    max_app_element(R2,R),!.

elem_least_balanced(L,R):-
    merge_lists_of_lists_no_dup(L,[],LND),
    merge_lists_of_lists(L,[],LD),
    all_possible_non_negated_literals(LND,LND_NN),
    list_to_set(LND_NN,LND_NN_SET),
    make_balance_frequency_vec(LND_NN_SET,LD,[],R2),
    max_app_element(R2,R),!.


reduction(L, C, R):-
    negate(C,C_N),
    member(C_N,L),
    subtract(L,[C_N],R),
    !.
reduction(L, C, "IN"):-
    member(C,L),
    !.
reduction(L,C,L):-
    !.

reduction_with_all([],_,TMP,R):-
    subtract(TMP,["IN"],R),!.
reduction_with_all([H|T],LIT,TMP,R):-
    reduction(H,LIT,R1),
    append(TMP,[R1],TMP_NEW),
    reduction_with_all(T,LIT,TMP_NEW,R),!.

eliminate_negated_elems([],[]):-!.
eliminate_negated_elems([H|T],R):-
    negate(H,H_NEG),
    H_NEG\=n(H),!,
    eliminate_negated_elems(T,R).
eliminate_negated_elems([H|T],[H|R]):-
    negate(H,H_NEG),
    H_NEG==n(H),!,
```

```
    eliminate_negated_elems(T,R).

dp([],[]):-
    !.
dp(L,_):-
    member([],L),
    !,
    fail.
dp(L,[(LIT/true)|S]):-
    elem_with_most_app(L,LIT),
    reduction_with_all(L,LIT,[],L1),
    dp(L1,S),
    !.
dp(L,[(LIT/false)|S]):-
    elem_with_most_app(L,LIT),
    negate(LIT,LIT_N),
    reduction_with_all(L,LIT_N,[],L1),
    dp(L1,S),
    !.

dp_2([],[]):-
    !.
dp_2(L,_):-
    member([],L),
    !,
    fail.
dp_2(L,[(LIT/true)|S]):-
    elem_least_balanced(L,LIT),
    reduction_with_all(L,LIT,[],L1),
    dp(L1,S),
    !.
dp_2(L,[(LIT/false)|S]):-
    elem_least_balanced(L,LIT),
    negate(LIT,LIT_N),
    reduction_with_all(L,LIT_N,[],L1),
    dp(L1,S),
    !.

apply_dp_on_all([]).
apply_dp_on_all([H|T]):-
    write('input-->'),
    write(H),
    nl,
    (
    dp(H,R)
    ->
    write('max appearance:'),write(R);
    write('max appearance: false')
    ),
    nl,
    (
    dp_2(H,R2)
```

```
    ->
    write('least balanced:'), write(R2);
    write('least balanced: false')
    ),
    nl,
    write('_____________________________'),
    nl,
    apply_dp_on_all(T).

read_file(S,[]) :-
    at_end_of_stream(S).
read_file(S,[L|R]) :-
    not(at_end_of_stream(S)),
    read(S,L),
    read_file(S,R).

apply_dp_on_file:-
    write('_____________________________'),
    nl,
    open('ex2_data.txt', read, S),
    read_file(S, L),
    apply_dp_on_all(L),
    close(S).
```