Project I

Dăscălescu Dana
Groupe 507 - Artificial Intelligence

December 23, 2022

The procedures for solving both problems have been implemented as in the course-presented versions from Brachman R. J. Levesque H. J. (2004). *Knowledge representation and reasoning.* Morgan Kaufmann.

I followed the coding guidelines for Prolog for both exercises outlined in *Coding guidelines for prolog - covingtoninnovations.com. (n.d.). Retrieved December 11, 2022, from https://covingtoninnovations.com/mc/plcoding.pdf.*

The following resources were consulted:

- *Prolog. SWI. (n.d.). Retrieved December 11, 2022, from https://www.swi-prolog.org/*
- *Dtonhofer. (2020, December 21). Prolog_notes/maplist_3_examples.MD at master · dtonhofer/prolog_notes. GitHub. Retrieved December 11, 2022, from https://github.com/dtonhofer/prolog_notes/blob/master/swipl_notes/about_maplist/maplist_3_examples.md*

1. Resolution We are required to create our own Knowledge Base (KB) and a Question (logically entailed from the KB), expressed in natural language.

a) Represent your KB in FOL, using a vocabulary that you must define.

1. Every Formula 1 fan is either a Mercedes or a Ferrari owner.
2. Every Mercedes owner is a tough person.
3. Any Ferrari owner has a bold attitude.
4. Any tough person is a risk-taker.
5. George is a Formula 1 fan.
6. (Question) If George is not a risk-taker, then he has a bold attitude.

We start by identifying the essential entities and defining the vocabulary:

- *Constant symbols* - function symbols with arity=0:
  Person: george
- Description of basic types of objects:

$$\text{Unary predicates} \begin{cases} Tough(x) \\ BoldAttitude(x) \\ RiskTaker(x) \\ Fan(x) \end{cases}$$

- Set of attributes of objects:

$$\text{Unary predicates} \begin{cases} MercedesOwner(x) \\ FerrariOwner(x) \end{cases}$$

We will write each sentence as a well-formed formula in first-order predicate calculus. The above sentences written in FOL:

1. $\forall x.(Fan(x) \wedge (MercedesOwner(x) \vee FerrariOwner(x)))$
2. $\forall x.(MercedesOwner(x) \supset Tough(x))$
3. $\forall x.(FerrariOwner(x) \supset BoldAttitude(x))$
4. $\forall x.(Tough(x) \supset RiskTaker(x))$
5. $Fan(george)$
6. $\neg RiskTaker(george) \supset BoldAttitude(george)$

b) By applying Resolution, we will prove "manually" that the question is logically entailed from KB.

We want to determine that the Question is logically entailed from KB, which is equivalent to $KB \cup \{\neg Question\}$ is unsatisfiable.

Firstly, we will transform the sentences in KB and the negated Question into CNF and denote the resulting set of clauses with S. Then, we will determine if S is unsatisfiable by applying Resolution and searching for a derivation of the empty clause.

The procedure for converting propositional formulas to CNF (replacing $\subset$ and $\equiv$ with their respective formulas; moving $\neg$ inward such that it appears in front of an atom; distributing $\wedge$ over $\vee$; collecting terms), as well as the handling of variables and quantifiers, adheres to the guidelines from *Course 3, Slides 6, 21-25.*

The following conversions were applied to rewrite as Conjuctive Normal Form clauses:

1. $\forall x.(Fan(x) \wedge (MercedesOwner(x) \vee FerrariOwner(x)))$

   $[Fan(x)]$
   $[MercedesOwner(x), FerrariOwner(x)]$

2. $\forall x.(MercedesOwner(x) \supset Tough(x))$

   $\forall x.(\neg MercedesOwner(x) \vee Tough(x))$

   $[\neg MercedesOwner(x), Tough(x)]$

3. $\forall x.(FerrariOwner(x) \supset BoldAttitude(x))$

   $\forall x.(\neg FerrariOwner(x) \vee BoldAttitude(x))$

   $[\neg FerrariOwner(x), BoldAttitude(x)]$

4. $\forall x.(Tough(x) \supset RiskTaker(x))$

   $\forall x.(\neg Tough(x) \vee RiskTaker(x))$

   $[\neg Tough(x), RiskTaker(x)]$

5. Fan(george)

6. $\neg(\neg RiskTaker(george) \supset BoldAttitude(george))$

   $\neg(\neg\neg RiskTaker(george) \vee BoldAttitude(george))$

   $\neg RiskTaker(george) \wedge \neg BoldAttitude(george)$

   $[\neg RiskTaker(george)]$

   $[\neg BoldAttitude(george)]$

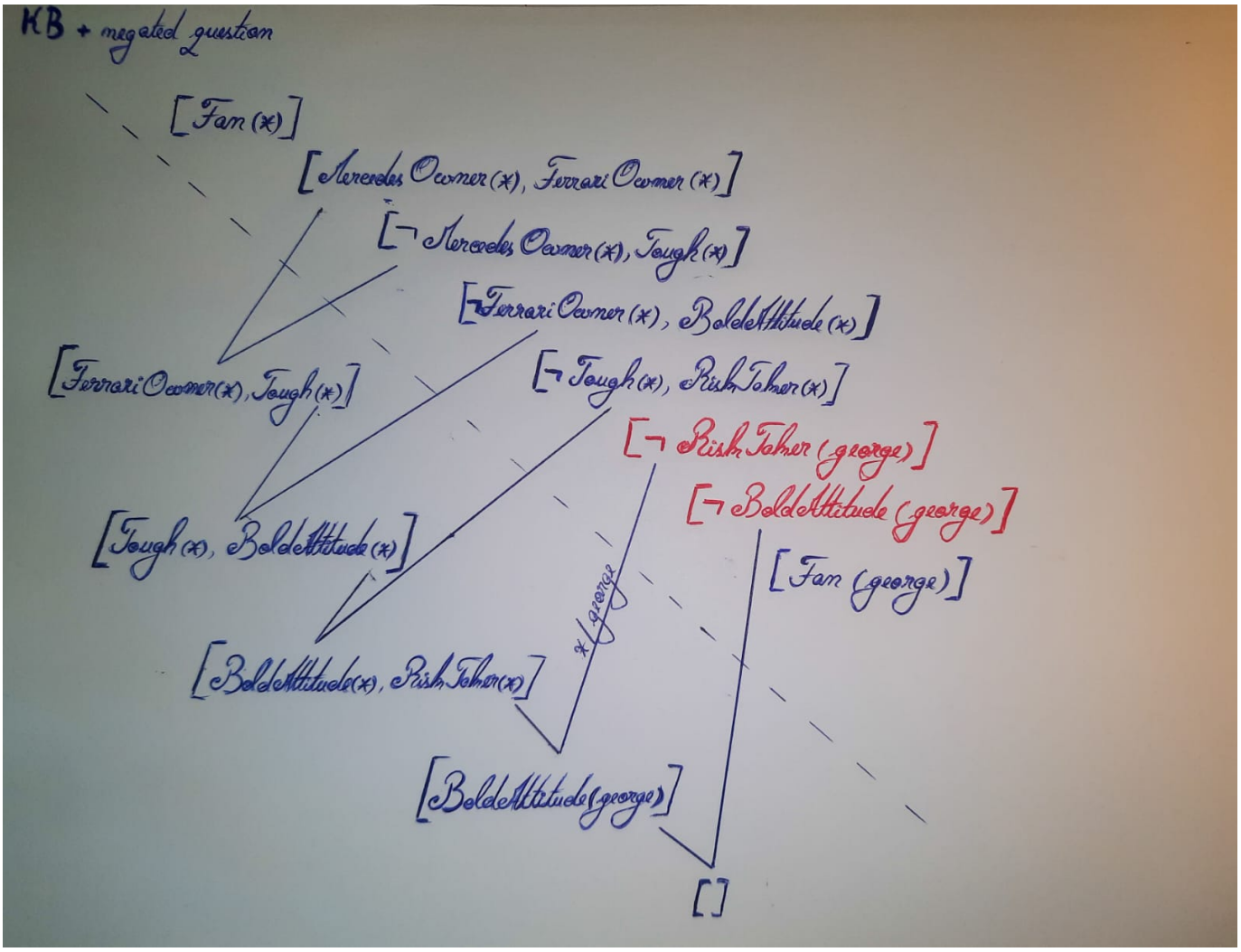

Figure 1: Resolution

In conclusions, we demonstrated that $KB \cup \neg Question$ is unsatisfiable (see Figure 1), implying that the Question is logically entailed from KB.

c) Prove "automatically" that the Question is logically entailed from Knowledge Base (by implementing Resolution in FOL).

The following search optimizations have been implemented:

- We eliminate any clauses containing a pure literal (appearing in a sentence when its negation does not exist anywhere else). Elimination of pure literals is a common heuristic employed by various satisfiability algorithms.

- We remove all tautologies from the clauses (clauses which contain both ρ and its negation, $\neg\rho$).
- We eliminate clauses for which another clause with a subset of literals already exists (possible after a substitution).

Before applying the resolution procedure, we also chose a specified order each time to increase the chances of deriving the empty clause. For this, we utilized the **sort/2** predicate.

d) Use your implementation of the Resolution for the propositional case, for the following sets of propositional clauses, written in CNF:

- $[[\neg a, b], [c, d], [\neg d, b], [\neg b], [\neg c, b], [e], [a, b, \neg f, f]]$
  A: Unsatisfiable
- $[[\neg b, a], [\neg a, b, e], [a, \neg e], [\neg a], [e]]$
  A: Unsatisfiable
- $[[\neg a, b], [c, f], [\neg c], [\neg f, b], [\neg c, b]]$
  A: Satisfiable
- $[[a, b], [\neg a, \neg b], [c]]$
  A: Satisfiable

2. SAT solver - The Davis Putnam We are required to implement the Davis-Putman SAT procedure. For a set of clauses S written in CNF, the procedure will display YES or NOT, depending on whether or not S is Satisfiable. In the case of YES, the procedure will also display the truth values assigned to the literals. We will choose two different strategies for selection of the atom to perform the $\cdot$ operation.

I applied two strategies to select an atom p: one in which the atom appears in the most clauses and another in which the atom appears in the fewest clauses.

The outcomes are shown above for the finite sets of propositional clauses written in CNF:

1. $[[toddler], [\neg toddler, child], [\neg child, \neg male, boy], [\neg infant, child], [\neg child, \neg female, girl], [female], [girl]]$
   (a) Strategy I: [(n(child),false),(girl,true),(n(male),true),(toddler,true),(female,true)]
   (b) Strategy II: [(boy,true),(female,true),(toddler,true),(n(child),false),(girl,true)]
2. $[[toddler], [\neg toddler, child], [\neg child, \neg male, boy], [\neg infant, child], [\neg child, \neg female, girl], [female], [\neg girl]]$
   (a) Strategy I: NO
   (b) Strategy II: NO
3. $[[\neg a, b], [c, d], [\neg d, b], [\neg c, b], [\neg b], [e], [a, b, \neg f, f]]$
   (a) Strategy I: NO
   (b) Strategy II: NO
4. $[[\neg b, a], [\neg a, b, e], [e], [a, \neg e], [\neg a]]$
   (a) Strategy I: NO
   (b) Strategy II: NO

5. $[[\neg a, \neg e, b], [\neg d, e, \neg b], [\neg e, f, \neg b], [f, \neg a, e], [e, f, \neg b]]$

   (a) Strategy I: [(n(b),true),(n(a),true)]

   (b) Strategy II: [(b,true),(n(a),true),(n(d),true),(e,true),(f,true)]

6. $[[a, b], [\neg a, \neg b], [\neg a, b], [a, \neg b]]$

   (a) Strategy I: NO

   (b) Strategy II: NO

As expected, the first strategy (choosing the atom that appears in the most clauses) is faster since more clauses are eliminated with each iteration.

Code

```
exercise1 :-
    % open('P1_1d_input4.txt', read, InputFile),
    open('P1_1c_input.txt', read, InputFile),
    read(InputFile,KB), close(InputFile),
    sort(KB,ClausesSorted),
    get_unique_literals(ClausesSorted,UniqueLiterals), eliminate_pure_clauses(ClausesSorted,UniqueLiterals
    eliminate_tautologies(ResultedClauses,ResultedClauses2),
    eliminate_subsumed_clauses(ResultedClauses2,ResultedClauses3),
    resolution(ResultedClauses3).


eliminate_pure_clauses([],_,[]).
eliminate_pure_clauses([H|T],UniqueLiterals,ResultingClauses) :-
    member(Lit,H), negate(Lit,NegatedLit), not(member(NegatedLit,UniqueLiterals)), !,
    eliminate_pure_clauses(T,UniqueLiterals,Aux), ResultingClauses=Aux;
    eliminate_pure_clauses(T,UniqueLiterals,Aux), ResultingClauses=[H|Aux].


eliminate_tautologies([],[]).
eliminate_tautologies([H|T],ResultingClauses) :-
    member(X,H), member(n(Y),H), X==Y, !, eliminate_tautologies(T,Aux), ResultingClauses=Aux;
    eliminate_tautologies(T,Aux), ResultingClauses=[H|Aux].


eliminate_subsumed_clauses(Clauses,ResultingClauses) :-
    member(Clause1,Clauses), member(Clause2,Clauses), Clause1\==Clause2,
    is_subsumed_clause(Clause1,Clause2), select(Clause2,Clauses,ResultingClauses2),
    eliminate_subsumed_clauses(ResultingClauses2,ResultingClauses);
    ResultingClauses=Clauses.


is_subsumed_clause([],_).
is_subsumed_clause([H|T],Clause2) :-
    contains_literal(H,Clause2), is_subsumed_clause(T,Clause2).


contains_literal(Lit,[Lit|_]).
```

```
contains_literal(Lit,[_|T]) :-
    contains_literal(Lit,T).


resolution([]) :-
    write("Satisfiable").
resolution(S) :-
    member([],S) -> write("Unsatisfiable");

    member(Clause1,S), member(Clause2,S), Clause1\==Clause2,
    select(Lit,Clause1,Aux1), select(n(Lit),Clause2,Aux2),
    get_resolvent(Aux1,Aux2,Resolvent),
    append(S,[Resolvent],UnionKB), sort(UnionKB,KB),
    get_unique_literals(KB,UniqueLiterals), eliminate_pure_clauses(KB,UniqueLiterals,ResultedClauses),
    eliminate_tautologies(ResultedClauses,KBFiltered), eliminate_subsumed_clauses(KBFiltered,NewKB),
    not(S=NewKB),
    resolution(NewKB);

    write("Satisfiable").


get_resolvent(Clause1,Clause2,Resolvent) :-
    append(Clause1,Clause2,Reunion), sort(Reunion,Resolvent).



exercise2 :-
    open('P1_2_i_input.txt',read,InputFile), read(InputFile,KB), close(InputFile),
    sort(KB,ClausesCNF),
    (   dp(ClausesCNF,Answer,Output) ->
        write("Satisfiable? "), write(Answer), write("\n"),
        (   Answer="YES" -> write(Output))
    ).


dp([],"YES",[]).
dp(S,"NO",_) :-
    member([],S),!.
dp(S,SAT,Values) :-
    choose_literal_strategy1(S,Literal),
    dot_operation(S,Literal,ResultedClauses),
    dp(ResultedClauses,Ans,Out),
    (   Ans="YES" -> SAT="YES", append([(Literal,"true")],Out,Values),!;

        negate(Literal,NegatedLiteral), dot_operation(S,NegatedLiteral,ResultedClauses2),
        dp(ResultedClauses2,SAT,Out2),
        append([(Literal,"false")],Out2,Values)    ).


choose_literal_strategy1(Clauses,ChosenLiteral) :-
    get_unique_literals(Clauses,UniqueLiterals),
    make_pairs(Clauses,UniqueLiterals,ListPairs),
    sort(0,@>,ListPairs,SortedListPairs),
    second(SortedListPairs,DescendingOrderLiterals),
    get_head_list(DescendingOrderLiterals,ChosenLiteral).
```

```
choose_literal_strategy2(Clauses,ChosenLiteral) :-
    get_unique_literals(Clauses,UniqueLiterals),
    make_pairs(Clauses,UniqueLiterals,ListPairs),
    sort(0,@<,ListPairs,SortedListPairs),
    second(SortedListPairs,DescendingOrderLiterals),
    get_head_list(DescendingOrderLiterals,ChosenLiteral).


make_pairs(Clauses,Literals,ListPairs) :-
    maplist(
        {Clauses}/[Lit,Freq]>>get_literal_frequency(Lit,Clauses,Freq),
         Literals,LiteralsCount),
    make_list_pairs(LiteralsCount,Literals,ListPairs).


make_list_pairs([],_,[]).
make_list_pairs(_,[],[]).
make_list_pairs([Head1|Tail1],[Head2|Tail2],[(Head1,Head2)|Rest]) :-
    make_list_pairs(Tail1,Tail2,Rest).


get_unique_literals(Clauses,UniqueLiterals) :-
    flatten(Clauses,LiteralsList), sort(LiteralsList,UniqueLiterals).


get_literal_frequency(Literal,Clauses,Frequency) :-
    get_clauses_containing_literal(Clauses,Literal,SubsetClauses),
    count_clauses(SubsetClauses,Frequency).


get_clauses_containing_literal(Clauses,Literal,ResultingClauses) :-
    include(
        {Literal}/[Clause] >> member(Literal,Clause),
        Clauses,
        ResultingClauses).


count_clauses([],0).
count_clauses([_|T],N):-
    count_clauses(T,Aux), N is Aux+1.


second([],[]).
second([(_,Y)|T],[Y|Rest]) :-
    second(T,Rest).


get_head_list([H|_],H).


dot_operation(SetClauses,Literal,ResultedClauses) :-
    include({Literal}/[Clause]>>not(member(Literal,Clause)),
            SetClauses,
            FilteredClauses),
    negate(Literal,LiteralNegated), include({LiteralNegated}/[Clause]>>not(member(LiteralNegated,Clause)),
```

```
                                        FilteredClauses,
                                        SubsetClauses1),
    include({LiteralNegated}/[Clause]>>member(LiteralNegated,Clause),
            SetClauses,
            FilteredClauses2),
    maplist({LiteralNegated}/[Clause,ClauseMinusLit]>>delete(Clause,LiteralNegated,ClauseMinusLit),
            FilteredClauses2,
            SubsetClauses2),
    union(SubsetClauses1,SubsetClauses2,ResultedClauses).


negate(n(X),X) :- !.
negate(X,n(X)).
```