Assignment 1
Knowledge Representation and Reasoning

Iordache Adrian-Răzvan

November 2020

1 Resolution

In this exercise will use as KB (Knowledge Base) the next assumptions:

1. Anyone who is smart enough and conscientious and lucky is successful.
2. If you are disciplined and wise you're conscientious.
3. If you're smart does not imply you're wise.
4. If you're hard working and you're wise then you're smart enough.
5. Being wise means making good decisions.
6. Everyone can get lucky.
7. Adrian is disciplined and hard working.
8. Adrian makes good decisions.

Question: Is Adrian successful?

1.1 Representation of the KB in FOL

1. $(isSmartEnough(X) \bigwedge isConscientious(X) \bigwedge isLucky(X)) \implies isSuccessful(X)$
2. $(isDisciplined(X) \bigwedge isWise(X)) \implies isConscientious(X)$
3. $(isSmart(X) \implies isWise(X)) \bigvee (isSmart(X) \implies \neg isWise(X))$
4. $(isHardWorking(X) \bigwedge isWise(X)) \implies isSmartEnough(X)$
5. $isWise(X) \implies makeGoodDecisions(X)$
6. $isLucky(X)$
7. $isDisciplined(adrian) \bigwedge isHardWorking(adrian)$
8. $makesGoodDecisions(adrian)$

Question: $isSuccessful(adrian)$

1.1.1 Conversion to Conjunctive Normal Form

1. $(\neg isSmartEnough(X) \bigvee \neg isConscientious(X) \bigvee \neg isLucky(X) \bigvee isSuccessful(X))$
2. $(\neg isDisciplined(X) \bigvee \neg isWise(X) \bigvee isConscientious(X))$
3. $(\neg isSmart(X) \bigvee isWise(X) \bigvee \neg isWise(X))$
4. $(\neg isHardWorking(X) \bigvee \neg isWise(X) \bigvee isSmartEnough(X))$
5. $(\neg isWise(X) \bigvee makeGoodDecisions(X))$
6. $isLucky(X)$
7. $isDisciplined(adrian)$
8. $isHardWorking(adrian)$
9. $makesGoodDecisions(adrian)$

1.1.2 Input to Resolution Algorithm

1. $[\neg isSmartEnough(X), \neg isConscientious(X), \neg isLucky(X), isSuccessful(X)]$
2. $[\neg isDisciplined(X), \neg isWise(X), isConscientious(X)]$
3. $[\neg isSmart(X), isWise(X), \neg isWise(X)]$
4. $[\neg isHardWorking(X), \neg isWise(X), isSmartEnough(X)]$
5. $[\neg isWise(X), makeGoodDecisions(X)]$
6. $[isLucky(X)]$
7. $[isDisciplined(adrian)]$
8. $[isHardWorking(adrian)]$
9. $[makesGoodDecisions(adrian)]$
10. $[\neg isSuccessful(adrian)]$

1.2 Manual prove that the question is entailed from KB, by applying Resolution

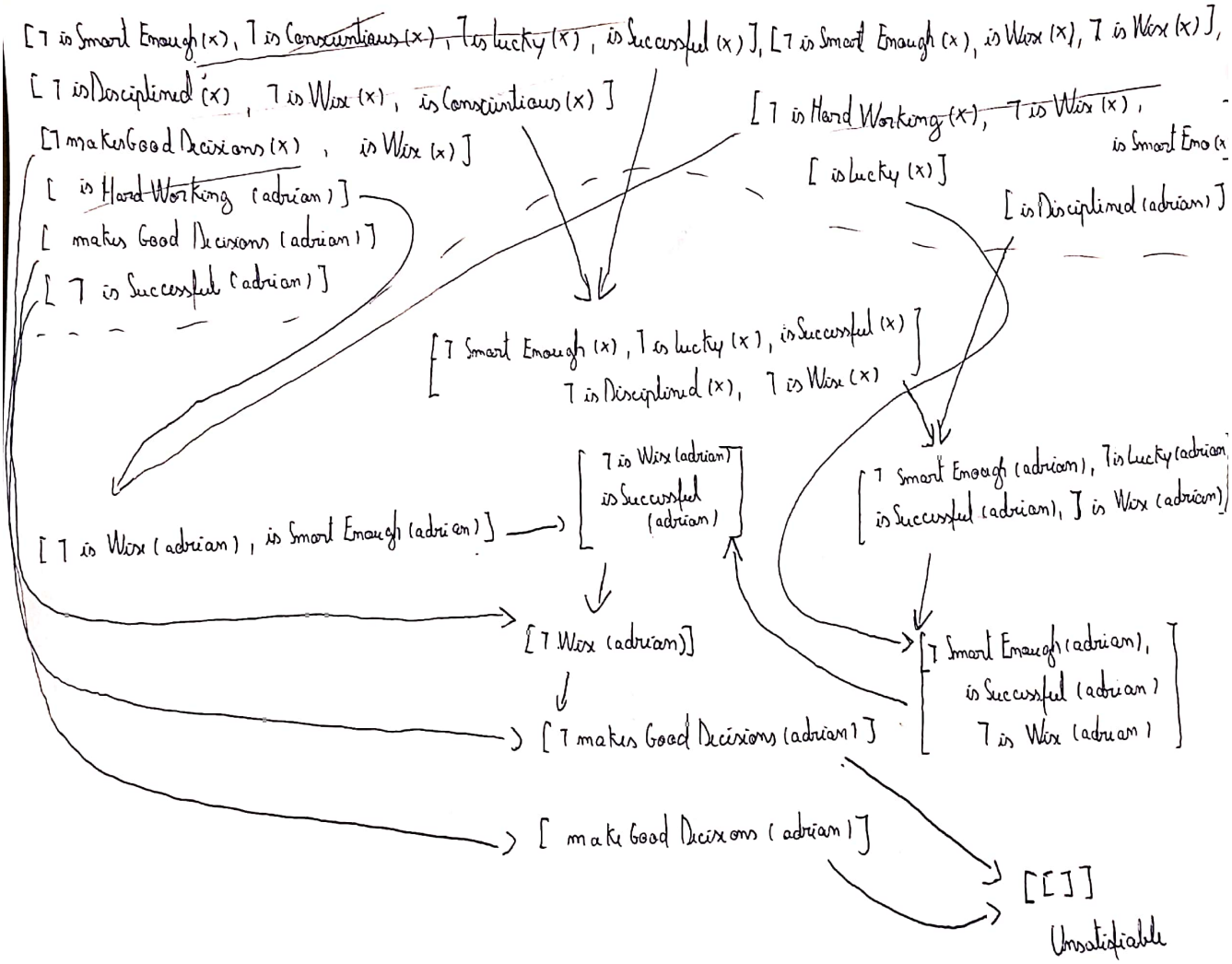

Figure 1: Prove that the question is entailed from KB

1.3 Output of the algorithm:

1.3.1 Raw Output with intermediate steps for input clauses as variables

$Input\ Clauses$:
$[[n(isSmartEnough(\_15636)), n(isConscientious(\_15636)), n(isLucky(\_15636)), isSuccessful(\_15636)],$
$[n(isDisciplined(\_15636)), n(isWise(\_15636)), isConscientious(\_15636)],$
$[n(isSmartEnough(\_15636)), isWise(\_15636), n(isWise(\_15636))],$
$[n(isHardWorking(\_15636)), n(isWise(\_15636)), isSmartEnough(\_15636)],$
$[n(makeGoodDecisions(\_15636)), isWise(\_15636)], [isLucky(\_15636)],$
$[isHardWorking(adrian)], [isDisciplined(adrian)],$
$[makeGoodDecisions(adrian)], [n(isSuccessful(adrian))]]$
$Opposite\ Clauses\ Selected$:
$[n(isDisciplined(_15636)), n(isWise(_15636)), isConscientious(_15636)]$
$[n(isSmartEnough(\_15636)), n(isConscientious(\_15636)), n(isLucky(\_15636)), isSuccessful(\_15636)]$
$Resulted\ Clause$:
$[n(isDisciplined(\_15636)), n(isWise(\_15636)), n(isSmartEnough(\_15636)), n(isLucky(\_15636)), isSuccessful(\_15636)]$
$Opposite\ Clauses\ Selected$:
$[isDisciplined(adrian)]$
$[n(isDisciplined(adrian)), n(isWise(adrian)), n(isSmartEnough(adrian)), n(isLucky(adrian)), isSuccessful(adrian)]$
$Resulted\ Clause$:
$[n(isWise(adrian)), n(isSmartEnough(adrian)), n(isLucky(adrian)), isSuccessful(adrian)]$
$Opposite\ Clauses\ Selected$:
$[isHardWorking(adrian)]$
$[n(isHardWorking(adrian)), n(isWise(adrian)), isSmartEnough(adrian)]$
$Resulted\ Clause$:
$[n(isWise(adrian)), isSmartEnough(adrian)]$
$Opposite\ Clauses\ Selected$:
$[isLucky(adrian)]$
$[n(isWise(adrian)), n(isSmartEnough(adrian)), n(isLucky(adrian)), isSuccessful(adrian)]$
$Resulted\ Clause$:
$[n(isWise(adrian)), n(isSmartEnough(adrian)), isSuccessful(adrian)]$
$Opposite\ Clauses\ Selected$:
$[n(isWise(adrian)), isSmartEnough(adrian)]$
$[n(isWise(adrian)), n(isSmartEnough(adrian)), isSuccessful(adrian)]$
$Resulted\ Clause$:
$[n(isWise(adrian)), isSuccessful(adrian)]$
$Opposite\ Clauses\ Selected$:
$[n(isWise(adrian)), isSuccessful(adrian)]$
$[n(isSuccessful(adrian))]$
$Resulted\ Clause$:
$[n(isWise(adrian))]$
$Opposite\ Clauses\ Selected$:
$[n(makeGoodDecisions(adrian)), isWise(adrian)]$
$[n(isWise(adrian))]$
$Resulted\ Clause$:
$[n(makeGoodDecisions(adrian))]$
$Opposite\ Clauses\ Selected$:
$[makeGoodDecisions(adrian)]$
$[n(makeGoodDecisions(adrian))]$
$Resulted\ Clause$: []
$Answer$: $Unsatisfiable$

The output above is very complicated and hard to follow, so will use this just as a proof of concept, but the explanation of the procedure will be done on the same input but with atoms instead of variables in clauses. With this will assume that the unification step is not necessary to explain.

1.3.2 Raw Output with intermediate steps for input clauses as atoms

$? - main\_resolution.$
$Input\ Clauses:$
$[[n(smart), n(conscient), n(lucky), succes],$
$[n(disciplined), n(wise), conscient],$
$[n(smart), wise, n(wise)],$
$[n(hardWork), n(wise), smart],$
$[n(goodDecisions), wise],$
$[lucky], [disciplined],$
$[hardWork], [goodDecisions], [n(succes)]]$
$Opposite\ Clauses\ Selected:$
$[n(disciplined), n(wise), conscient][n(smart), n(conscient), n(lucky), succes]$
$Resulted\ Clause: [n(disciplined), n(wise), n(smart), n(lucky), succes]$
$Opposite\ Clauses\ Selected:$
$[disciplined][n(disciplined), n(wise), n(smart), n(lucky), succes]$
$Resulted\ Clause:$
$[n(wise), n(smart), n(lucky), succes]$
$Opposite\ Clauses\ Selected:$
$[goodDecisions][n(goodDecisions), wise]$
$Resulted\ Clause: [wise]$
$Opposite\ Clauses\ Selected: [hardWork][n(hardWork), n(wise), smart]$
$Resulted\ Clause: [n(wise), smart]$
$Opposite\ Clauses\ Selected: [lucky][n(wise), n(smart), n(lucky), succes]$
$Resulted\ Clause: [n(wise), n(smart), succes]$
$Opposite\ Clauses\ Selected: [n(wise), smart][n(wise), n(smart), succes]$
$Resulted\ Clause: [n(wise), succes]$
$Opposite\ Clauses\ Selected: [n(wise), succes][n(succes)]$
$Resulted\ Clause: [n(wise)]$
$Opposite\ Clauses\ Selected: [wise][n(wise)]$
$Resulted\ Clause: []$
$Answer: Unsatisfiable$

1.3.3 Procedure Explanations

From the last example we can observe how the clauses are selected and how we obtain the final result.

Step 1: Will remove the trivial clauses, the ones that contain opposite literals such as [a, n(a)].

Step 2: Will extract and sort all literals from all the clauses.

Step 3: In the next phase will iterate through all literals trying to find a positive clause (that contains the variable e.g. [a]) and a negative clause (that contains the negation of the variable e.g. [n(a)]) for the same literal.

Step 4: If we don't find at least a literal that has a positive and a negative clause will say that the KB is Satisfiable and the Question we've asked it's not logically entailed from the KB.

Step 5: If we find the positive and negative case, will delete the positive and the negative literal and will add to the initial set of clauses the union of the positive and negative clauses after the removal step. And go back to "Step 1" until [] is deduced or "Step 4" is reached.

1.4 Results for given examples:

- $[[n(a), b], [c, d], [n(d), b], [n(c), b], [n(b)]]$ - Satisfiable
- $[[n(b), a], [n(a), b, e], [e], [a, n(e)], [n(a)]]$ - Unsatisfiable
- $[[n(a), b], [c, f], [n(f), b], [n(c), b], [n(c)]]$ - Satisfiable
- $[[a, b], [n(a), n(b)]]$ - Satisfiable

2 SAT solver – The Davis Putnam procedure

For the implementation of the Davis Putnam will use three strategies for the selection of the atom to perform the • (dot) operation.

1. Choose each literal sorted in lexical order.
2. Choose the most frequent literal in the set of clauses.
3. Choose the least frequent literal in the set of clauses.

2.1 Test Cases for Davis-Putnam Procedure

- $[[toddler], [n(toddler), child], [n(child), n(male), boy], [n(infant), child], [n(child), n(female), girl], [female], [girl]]$
- $[[toddler], [n(toddler), child], [n(child), n(male), boy], [n(infant), child], [n(child), n(female), girl], [female], [n(girl)]]$
- $[[n(a), b], [c, d], [n(d), b], [n(c), b], [n(b)]]$
- $[[n(b), a], [n(a), b, e], [e], [a, n(e)], [n(a)]]$
- $[[n(a), n(e), b], [n(d), e, n(b)], [n(e), f, n(b)], [f, n(a), e], [e, f, n(b)]]$
- $[[a, b], [n(a), n(b)], [n(a), b], [a, n(b)]]$

2.1.1 Results on the Davis-Putnam (Literals sorted in lexical order)

- Satisfiable - $[(boy, True), (child, True), (female, True), (girl, True), (infant, True), (male, True), (toddler, True)]$
- Unsatisfiable
- Unsatisfiable
- Unsatisfiable
- Satisfiable - $[(a, True), (b, True), (d, True), (e, True), (f, True)]$
- Unsatisfiable

2.1.2 Results on the Davis-Putnam (Most Frequent Literal First)

- Satisfiable - $[(child, True), (female, True), (girl, True), (toddler, True), (boy, True)]$
- Unsatisfiable
- Unsatisfiable
- Unsatisfiable
- Satisfiable - $[(e, True), (b, True), (f, True)]$
- Unsatisfiable

2.1.3 Results on the Davis-Putnam (Least Frequent Literal First)

- Satisfiable - $[(boy, True), (infant, True), (male, True), (female, True), (girl, True), (toddler, True), (child, True)]$
- Unsatisfiable
- Unsatisfiable
- Unsatisfiable
- Satisfiable - $[(d, True), (a, True), (f, True), (b, True), (e, True)]$
- Unsatisfiable

As we can observe the method that uses the most frequent literal first is more efficient because with fewer steps reaches the correct answer in all test cases. The values that are not in the final list are not essential and can be truth or false.

And between picking literals in sorted order and the least frequent method it seems to be no difference in performance.

3 Project Implementation

```
read_pipeline_resolution(X) :-
see('/home/adrian/Desktop/Python/Personal/Master/Master-Projects/First-Year/Knowledge-Represe
read(X), seen.

read_pipeline_davis_putman(X) :-
see('/home/adrian/Desktop/Python/Personal/Master/Master-Projects/First-Year/Knowledge-Represe
read(X), seen.


is_trivial_clause([], 'No').

is_trivial_clause(Clause, 'Yes') :-
member(Lit, Clause),
member(n(Lit), Clause).

is_trivial_clause(_, 'No').

remove_trivial_clauses([], []).
remove_trivial_clauses([Clause | Clauses], Filtered) :-
is_trivial_clause(Clause, 'Yes'),
remove_trivial_clauses(Clauses, Filtered).

remove_trivial_clauses([Clause | Clauses], [Clause | Filtered]) :-
is_trivial_clause(Clause, 'No'),
remove_trivial_clauses(Clauses, Filtered).



get_literals_from_clause_([], Literals, Literals).
```

```
get_literals_from_clause_([n(Lit) | Lits], Literals, Aux) :-
get_literals_from_clause_(Lits, Literals, [Lit | Aux]).

get_literals_from_clause_([Lit | Lits], Literals, Aux) :-
get_literals_from_clause_(Lits, Literals, [Lit | Aux]).

get_literals_from_clause([], []).
get_literals_from_clause(Clause, Literals) :-
get_literals_from_clause_(Clause, Literals, []).

get_literals_from_clauses_([], Literals, Aux) :-
sort(Aux, Literals).

get_literals_from_clauses_([Clause | Clauses], Literals, Aux) :-
get_literals_from_clause(Clause, Lits),
append(Lits, Aux, Union),

get_literals_from_clauses_(Clauses, Literals, Union).

get_literals_from_clauses([], []).
get_literals_from_clauses(Clauses, Lits) :-
get_literals_from_clauses_(Clauses, Lits, []).


find_literal_in_clauses([], _, []).
find_literal_in_clauses([Clause | _], Lit, Clause) :-
member(Lit, Clause), !.

find_literal_in_clauses([Clause | Clauses], Lit, Result) :-
not(member(Lit, Clause)),
find_literal_in_clauses(Clauses, Lit, Result).

find_resolvent(_, [], [], [], _).
find_resolvent(Clauses, [Lit | _], PositiveClause, NegativeClause, Lit) :-
find_literal_in_clauses(Clauses, Lit, PositiveClause),
find_literal_in_clauses(Clauses, n(Lit), NegativeClause),
PositiveClause \= [], NegativeClause \= [], !.

find_resolvent(Clauses, [_ | Lits], PositiveClause, NegativeClause, Lit) :-
find_resolvent(Clauses, Lits, PositiveClause, NegativeClause, Lit).

resolution([], "Satisfiable") :- !.

resolution(Clauses, 'Unsatisfiable') :- member([], Clauses), !.

resolution(Clauses, 'Satisfiable') :-
remove_trivial_clauses(Clauses, Filtered),
get_literals_from_clauses(Filtered, Lits),
find_resolvent(Filtered, Lits, PositiveClause, NegativeClause, _),
PositiveClause = [], NegativeClause = [].
```

```
resolution(Clauses, Answer) :-
remove_trivial_clauses(Clauses, Filtered),
get_literals_from_clauses(Filtered, Lits),
find_resolvent(Filtered, Lits, PositiveClause, NegativeClause, Chosen),
PositiveClause \= [], NegativeClause \= [],
delete(PositiveClause, Chosen, RemovedPos),
delete(NegativeClause, n(Chosen), RemovedNeg),
union(RemovedPos, RemovedNeg, Result),
delete(Filtered, PositiveClause, RemovedPositives),
delete(RemovedPositives, NegativeClause, RemovedNegatives),
union(RemovedNegatives, [Result], NewClauses),
resolution(NewClauses, Answer).


main_resolution :-
read_pipeline_resolution(Clauses),
write("Input Clauses: "), nl, write(Clauses), nl,
resolution(Clauses, Answer),
write("Answer: "), write(Answer), nl.




clauses_with_literal(_, [], []).
clauses_with_literal(Lit, [Clause | Clauses], Result) :-
not(member(Lit, Clause)),
clauses_with_literal(Lit, Clauses, Result).

clauses_with_literal(Lit, [Clause | Clauses], [Clause | Result]) :-
member(Lit, Clause),
clauses_with_literal(Lit, Clauses, Result).

delete_from_clauses(_, [], []).
delete_from_clauses(Lit, [Clause | Clauses], [Removed | Result]) :-
delete(Clause, Lit, Removed),
delete_from_clauses(Lit, Clauses, Result).

clauses_without_literal(_, [], []).
clauses_without_literal(Lit, [Clause | Clauses], Result) :-
member(Lit, Clause),
clauses_without_literal(Lit, Clauses, Result).

clauses_without_literal(Lit, [Clause | Clauses], Result) :-
member(n(Lit), Clause),
clauses_without_literal(Lit, Clauses, Result).

clauses_without_literal(Lit, [Clause | Clauses], [Clause | Result]) :-
```

```
not(member(Lit, Clause)),
not(member(n(Lit), Clause)),
clauses_without_literal(Lit, Clauses, Result).

negation(n(Lit), Lit) :- !.
negation(Lit, n(Lit)).


dot(Clauses, Lit, OutputClauses) :-
negation(Lit, LitN),
clauses_without_literal(Lit, Clauses, Without),
clauses_with_literal(LitN, Clauses, With),
delete_from_clauses(LitN, With, Removed),
union(Without, Removed, OutputClauses).


davis_putman([], _, [], "Satisfiable") :- !.

davis_putman(Clauses, _, _, "Unsatisfiable") :- member([], Clauses), !.

davis_putman(Clauses, [Lit | Lits], [(Lit, "True") | Solution], "Satisfiable") :-
dot(Clauses, Lit, OutputClauses),
davis_putman(OutputClauses, Lits, Solution, "Satisfiable"), !.


davis_putman(Clauses, [Lit | Lits], [(Lit, "False") | Solution], "Satisfiable") :-
dot(Clauses, n(Lit), OutputClauses),
davis_putman(OutputClauses, Lits, Solution, "Satisfiable"), !.

davis_putman(_, _, [], "Unsatisfiable").


find_frequency_literal(_, [], 0).
find_frequency_literal(Lit, [Clause | Clauses], Freq) :-
member(Lit, Clause),
find_frequency_literal(Lit, Clauses, Aux),
Freq is Aux + 1.

find_frequency_literal(Lit, [Clause | Clauses], Freq) :-
member(n(Lit), Clause),
find_frequency_literal(Lit, Clauses, Aux),
Freq is Aux + 1.

find_frequency_literal(Lit, [Clause | Clauses], Freq) :-
not(member(Lit, Clause)),
not(member(n(Lit), Clause)),
find_frequency_literal(Lit, Clauses, Freq).


find_frequency(_, [], []).
find_frequency(Clauses, [Lit | Lits], [(Lit, Freq) | Result]) :-
```

```
find_frequency_literal(Lit, Clauses, Freq),
find_frequency(Clauses, Lits, Result).


extract_literals([], []).
extract_literals([(Lit, _) | Rest], [Lit | Extracted]) :-
extract_literals(Rest, Extracted).




main_davis_putman :-
read_pipeline_davis_putman(Clauses),
remove_trivial_clauses(Clauses, Filtered),
get_literals_from_clauses(Filtered, Lits),
davis_putman(Filtered, Lits, Solution, Ans),
write(Ans), nl, write(Solution), nl.



main_davis_putman_max_frequency :-
read_pipeline_davis_putman(Clauses),
remove_trivial_clauses(Clauses, Filtered),
get_literals_from_clauses(Filtered, Lits),
find_frequency(Filtered, Lits, Preprocessed),
sort(2, @>=, Preprocessed, Sorted),
extract_literals(Sorted, SortedLiterals),
davis_putman(Filtered, SortedLiterals, Solution, Ans),
write(Ans), nl, write(Solution), nl.


main_davis_putman_min_frequency :-
read_pipeline_davis_putman(Clauses),
remove_trivial_clauses(Clauses, Filtered),
get_literals_from_clauses(Filtered, Lits),
find_frequency(Filtered, Lits, Preprocessed),
sort(2, @=<, Preprocessed, Sorted),
extract_literals(Sorted, SortedLiterals),
davis_putman(Filtered, SortedLiterals, Solution, Ans),
 write(Ans), nl, write(Solution), nl.
```