KRR – project 1

Eduard George Stan

407 – Artificial Intelligence

Part I: Resolution Algorithm

The resolution algorithm has the purpose of proving the truth or falsity of a statement. The implementation presented in this project report uses prolog to make the resolution algorithm possible in a recursive manner.

Basically, we have a knowledge base (KB), which means a list of clauses. The algorithm searches for an empty clause to conclude that the whole statement is Unsatisfiable (False). If it does not find the empty clause, it applies resolution between the clauses, and adds the resolvent clause to the KB, if it is not already there. The algorithm works recursively until there are no more unique resolvents to add to the KB. If the algorithm can't find the empty clause, it stops and returns Satisfiable (True).

A brief explanation of the main predicates that are present in the code:

1. *Neg/2* - checks if a parameter is the negation of the other parameter.
2. *resolvent/3* - generates the resolvent, given two clauses.
3. *res/1* - computes resolution. It checks for an empty clause in the KB. If the function does not find an empty clause, it computes resolution until it can't be done any further.

There are several optimizations to make for the resolution algorithm. The ones that were implemented for this project are done by removing:

- Pure clauses.
- Tautologies.
- Subsumed clauses.

Their implementation will be discussed during the oral examination.

Example of knowledge base using natural language:

1. Alex is a PhD student.
2. Every python developer who has 5 years of experience is a senior.
3. Every senior has a pet.
4. Every PhD student that is also a python developer has 5 years of experience.
5. Alex is a Python developer.

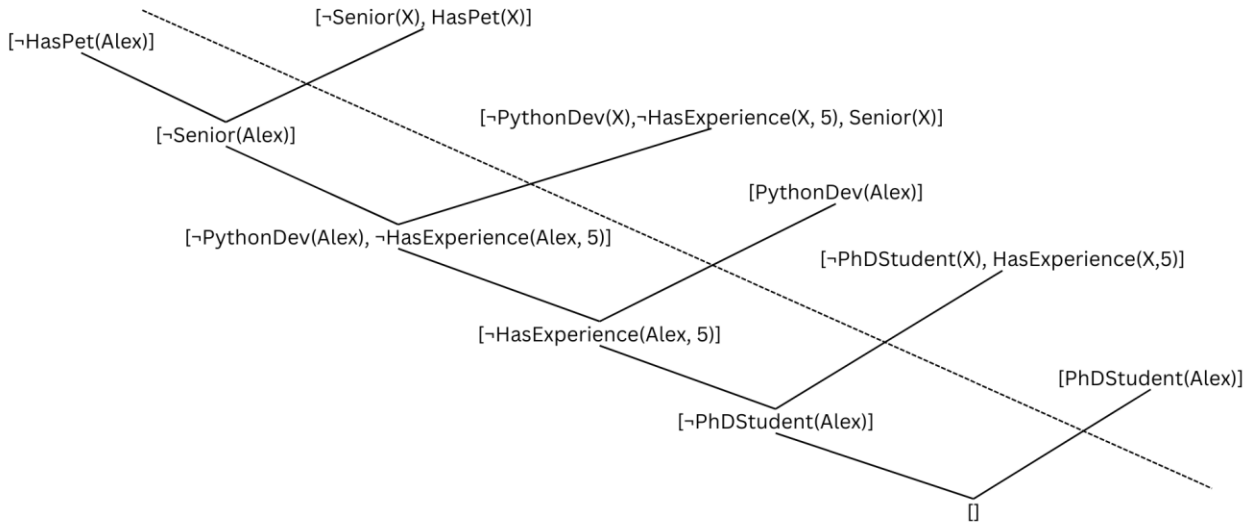
Question: Does Alex have a pet?

FOL:

- PhDStudent(Alex)
- ∀X[PythonDev(X) ^ HasExperience(x, 5) ⊃ Senior(X)]
- ∀X[Senior(X) ⊃ HasPet(X)]
- ∀X[PythonDev(X) ^ PhDStudent(X)⊃ HasExperience(x, 5)]
- PythonDev(Alex)
- HasPet(Alex)

CNF:

- PhDStudent(Alex)
- ¬ PythonDev(X) V ¬ HasExperience(x, 5) V Senior(X)
- ¬ Senior(X) V HasPet(X)
- ¬ PythonDev(X) V ¬ PhDStudent(X) V HasExperience(x, 5)
- PythonDev(Alex)
- ¬ HasPet(Alex)



Part II: SAT solver – The Davis Putnam Procedure

The Davis-Putnam algorithm has the purpose of checking the satisfiability of propositional logic formulas. For a knowledge base KB, the algorithm works as follows:

- Displays “YES” if KB is satisfiable. For this case, display the truth values associated with the literals.
- Displays “NOT” if KB is unsatisfiable.

For this project, two methods of choosing the “optimal” literal were used, namely selecting the most frequent literal in the KB, and selecting the first literal present in the shortest clause.

The implementation will be discussed during the oral examination.

The implementations

1. Resolution

```
input:- open('/...PATH.../reasoning_input.txt', read, Stream),
iter_inputs(Stream),
close(Stream).

iter_inputs(Stream):- read_line_to_codes(Stream, KBcodes), KBcodes \= at_end_of_stream(Stream),
read_term_from_codes(KBcodes, KB, []), main(KB), iter_inputs(Stream).
iter_inputs(_Stream):- !.

neg(n(X), X):- !.
neg(A, n(A)).

% optimizations

elim_element(_,[],[]):- !.
elim_element(E,[E|T],N):- elim_element(E,T,N).
elim_element(E,[H|T],[H|N]):- E\=H, elim_element(E,T,N).

concatenate([], L, L):- !.
concatenate([E|L1], L2, [E|L3]):- concatenate(L1, L2, L3).

find_pure_neg(_E, []):- !, fail.
find_pure_neg(E, [H|T]):- not(member(E, H)), find_pure_neg(E, T).
find_pure_neg(E, [H|_T]):- member(E, H), !.
```

```
iter_neg([], _T).

iter_neg([E1|L], T):- neg(E1, E2), copy_term(E2, X), find_pure_neg(X, T), iter_neg(L, T).


remove_pure(_R, [], []):- !.

remove_pure(R, [L|RestKB], [L|NKB]):- concatenate(R, RestKB, T), iter_neg(L, T), remove_pure([L|R], RestKB, NKB).

remove_pure(R, [L|RestKB], NKB):- concatenate(R, RestKB, T), not(iter_neg(L, T)), remove_pure(R, RestKB, NKB).


is_subsumed(_L, []):- !, fail.

is_subsumed(L, [H|T]):- not(subset(H, L)), is_subsumed(L, T).

is_subsumed(L, [H|_T]):- subset(H, L), !.


remove_subsumed(_R, [], []):- !.

remove_subsumed(R, [L|RestKB], NKB):- concatenate(R, RestKB, T), is_subsumed(L, T), remove_subsumed(R, RestKB, NKB).

remove_subsumed(R, [L|RestKB], [L|NKB]):- concatenate(R, RestKB, T), not(is_subsumed(L, T)), remove_subsumed([L|R], RestKB, NKB).


remove_tautologies([], []):- !.

remove_tautologies([L|RestKB], [L|NKB]):- member(E1, L), neg(E1, E2), copy_term(E2, X), not(member(X, L)), remove_tautologies(RestKB, NKB).

remove_tautologies([L|RestKB], NKB):- member(E1, L), neg(E1, E2), copy_term(E2, X), member(X, L), remove_tautologies(RestKB, NKB).


% main implementation


sortKB([],[]).

sortKB([A|B], [A1|B1]):- sort(A, A1), sortKB(B, B1).


afis_list([]):- nl.
```

```
afis_list([A|B]):-write(A), tab(2), afis_list(B).

resolvent(L1, L2, R3):- member(E1, L1), neg(E1,E2), copy_term(E2, X), member(X, L2),
elim_element(E1, L1, R1), elim_element(E2, L2, R2),
concatenate(R1, R2, R3).

res(KB) :- member([], KB), write('UNSAT'), nl, nl.
res(KB) :-member(L1, KB), member(L2, KB), L1 \= L2,
resolvent(L1, L2, R), sort(R, R1), not(member(R1, KB)),
res([R1|KB]).
res(_KB):- write('SAT'), nl, nl.

main(C):- write('Initial KB: '), afis_list(C), remove_subsumed([], C, C1),
write('After subsumed: '), afis_list(C1), remove_tautologies(C1, C2),
write('After tautologies: '), afis_list(C2), remove_pure([], C2, C3),
write('After pure: '), afis_list(C3), sortKB(C3, KB),
write('After sort: '), afis_list(KB), res(KB).
```

And the input looks as follows:

```
‘’’
[[n(a),b],[c,d],[n(d),b],[n(b)],[n(c),b],[e],[a,b,n(f),f], [h, a]].
[[a, b], [n(a), n(b)], [c]].
[[n(a),b],[c,f],[n(c)],[n(f),b],[n(c),b]].
[[n(b),a],[n(a),b,e],[a,n(e)],[n(a)],[e]].
[[on(a, b)], [green(a)], [on(b, c)], [n(green(a))], [n(on(X,Y)), n(green(X)), green(Y)]].
[[phDStudent(alex)], [n(pythonDev(X)), n(hasExperience(X, 5)), senior(X)], [n(senior(X)), hasPet(X)], [n(pythonDev(X)), n(phDStudent(X)), hasExperience(X, 5)], [pythonDev(alex)], [n(hasPet(alex))]].
’’’
```

2. Davis-Putnam

```
input:- open('/...PATH.../sat_input.txt', read, Stream),

iter_inputs(Stream),

close(Stream).


iter_inputs(Stream):- read_line_to_codes(Stream, KBcodes), KBcodes \= at_end_of_stream(Stream),

read_term_from_codes(KBcodes, KB, []), read_line_to_codes(Stream, Litcodes),

read_term_from_codes(Litcodes, Lit, []), main(KB, Lit), main_short(KB), main_freq(KB),
iter_inputs(Stream).

iter_inputs(_Stream):- !.


% ----------------


select_most_frequent(KB, M):- flatten(KB, FlatKB), sort(FlatKB, SortedKB), create_count_list(SortedKB,
FlatKB, CountList),

max_member(_Count/M, CountList).

create_count_list([], _FlatKB, []).

create_count_list([M|SortedKB], FlatKB, [Count/M|Rest]):- count_occurrences_in_clause(M, FlatKB,
Count),

create_count_list(SortedKB, FlatKB, Rest).


count_occurrences_in_clause(_, [], 0).

count_occurrences_in_clause(M, [M|T], Count) :-

count_occurrences_in_clause(M, T, RestCount),

Count is RestCount + 1.
```

```
count_occurrences_in_clause(M, [H|T], Count) :-
M \= H,
count_occurrences_in_clause(_M, T, Count).

% -------------

select_from_shortest_clause([], []).

select_from_shortest_clause([C|RestKB], P) :- \+ member([], [C|RestKB]),
determine_shortest_clause(RestKB, C, [P|_CBest]).

determine_shortest_clause([], CB, CB) :- !.

determine_shortest_clause([C|RestKB], CSC, CBest) :-
length(CSC, LenCSC),
length(C, LenC),
LenC < LenCSC, determine_shortest_clause(RestKB, C, CBest);
determine_shortest_clause(RestKB, CSC, CBest).

% -------------

neg(n(X), X):- !.
neg(A, n(A)).

remove_element(_, [], []).
remove_element(E, [E|T], N):-
remove_element(E, T, N).
remove_element(E, [H|T], [H|N]):-
E\=H,
```

```
remove_element(E, T, N).

% ----------

dot([], _M, _Mc, []).
dot([C|KB1], M, Mc, [C|KB2]):- \+ member(M, C), \+ member(Mc, C),
dot(KB1, M, Mc, KB2).
dot([C|KB1], M, Mc, [Cr|KB2]):- \+ member(M, C), member(Mc, C),
remove_element(Mc, C, Cr), dot(KB1, M, Mc, KB2).
dot([C|KB1], M, Mc, KB2):- member(M, C), dot(KB1, M, Mc, KB2).

redo_dot(KB1, M/false, Mc, KB2, KB3):- member([], KB2), dot(KB1, Mc, M, KB3), !.
redo_dot(_, _/true, _, KB3, KB3).

dp([],[]):- write('YES - empty KB'), nl.
dp(KB, []):- \+ member([], KB), write('YES'), nl.
dp(KB, []):- member([],KB), write('NOT'), nl.
dp(KB, [_/_|_]):- member([],KB), write('NOT - early'), nl, !.
dp(KB1,[M/A|S]):- neg(M, Mc), dot(KB1, M, Mc, KB2),
redo_dot(KB1, M/A, Mc, KB2, KB3), dp(KB3, S), write(M/A), nl.

afis_list([]):- nl.
afis_list([A|B]):-write(A), tab(2), afis_list(B).

main(KB, Lit):- nl, afis_list(KB), nl, write('Given params: '), dp(KB, Lit).
main_short(KB):- nl, write('Literal from shortest clause: '), select_from_shortest_clause(KB, Lit), dp(KB,
[Lit/_A]).
main_freq(KB):- nl, write('Most frequent literal: '), select_most_frequent(KB, Lit), dp(KB, [Lit/_A]).
```

The input looks like this:

‘’’

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

[a/A, b/B, d/D, e/E, f/F].

[[toddler],[n(toddler),child],[n(child),n(male),boy],[n(infant),child],[n(child),n(female),girl], [female], [girl]].

[toddler/T, child/C, male/M, girl/G, boy/B, female/F].

[[toddler],[n(toddler),child],[n(child),n(male),boy],[n(infant),child],[n(child),n(female),girl], [female], [n(girl)]].

[toddler/T, child/C, male/M, girl/G, boy/B, female/F].

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

[a/A, b/B, c/C, d/D, e/E, f/F].

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

[a/A, b/B, e/E].

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

[a/A, b/B, d/D, e/E, f/F].

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

[a/A, b/B].

’’’