Documentation for Project 1
Knowledge Representation and Reasoning

Sîrbu Oana Adriana

Artificial Intelligence / 407

1. Resolution

The first task of this project involves employing my custom implementation of the Resolution algorithm. This part of the project aims to utilize the algorithm on my original knowledge base (KB) as well as on various sets of propositional clauses. I will begin by presenting my KB expressed in natural language, followed by the transformation into First Order Logic (FOL), and ultimately, the conversion into Conjunctive Normal Form (CNF).

1.1 The Knowledge Base in natural language

1. Anyone who goes to Lapland believes in magic.
2. Anyone who believes in magic receives presents.
3. Andrew is a member of GiveHope Community.
4. Every member of GiveHope Comumnity goes to Lapland.

   Question: Andrew receives presents.

1.2 FOL representation

1. $\forall X$ (goesToLapland(X) $\rightarrow$ believesInMagic(X))
2. $\forall X$ (believesInMagic(X) $\rightarrow$ receivesPresents(X))
3. memberOfCommunity(andrew, giveHope)
4. $\forall X$ memberOfCommunity(X, giveHope) $\rightarrow$ goesToLapland(X)

   Question: receivesPresents(andrew).

1.3 CNF

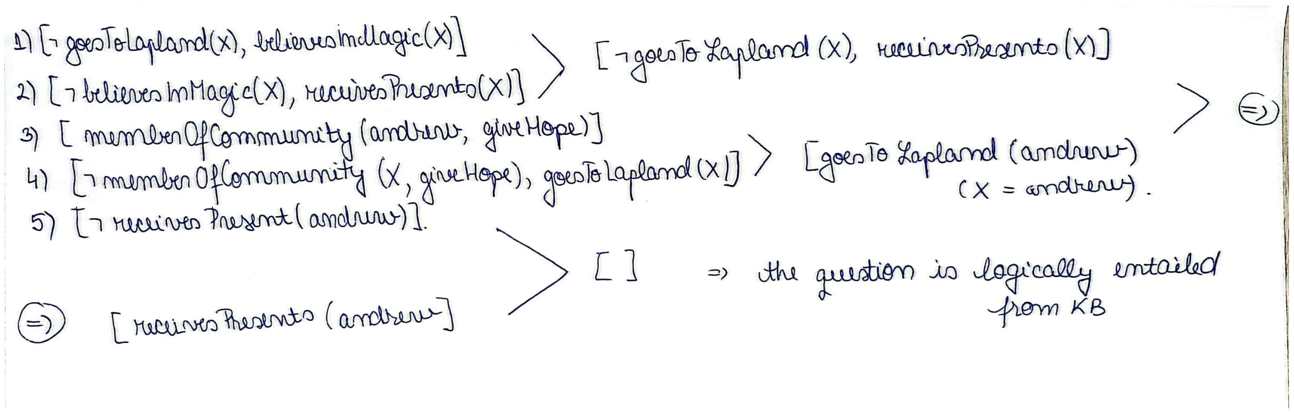
1. ¬goesToLapland(X) ∨ believesInMagic(X)
2. ¬believesInMagic(X) ∨ receivesPresents(X)
3. memberOfCommunity(andrew, giveHope)
4. ¬memberOfCommunity(X, giveHope) ∨ goesToLapland(X))

   Question: ¬receivesPresents(andrew)

1.4 List of clauses for the KB

- n(goesToLapland(X)), believesInMagic(X)
- n(believesInMagic(X)), receivesPresents(X)
- memberOfCommunity(andrew, giveHope)
- n(memberOfCommunity(X, giveHope)), goesToLapland(X)
- n(receivesPresents(andrew))

1.5 The manual proof that the Question is logically entailed from KB, by applying Resolution



1) [¬ goesToLapland(X), believesInMagic(X)]
2) [¬ believesInMagic(X), receivesPresents(X)]

> [¬ goesTo Lapland (X), receivesPresents (X)]

3) [memberOfCommunity (andrew, giveHope)]
4) [¬ memberOfCommunity (X, giveHope), goesToLapland (X)]

> [goesTo Lapland (andrew)
> (X = andrew).

> ⇒

5) [¬ receives Present (andrew)].

⇒ [receivesPresents (andrew]

> [ ]   ⇒ the question is logically entailed from KB

CamScanner

1.6 The automatic proof that the Question is logically entailed from KB, by implementing Resolution

The actual code will be appended at the end of the document, as specified in the project's requirements, and will be discussed during the oral examination. It is noteworthy that all three possible optimizations were implemented:

1. Removal of pure clauses (clauses containing some literal ρ such that $\neg\rho$ does not appear anywhere in the knowledge base).

2. Elimination of tautologies (clauses containing both ρ and $\neg\rho$).

3. Removal of subsumed clauses (clauses for which another clause already exists with a subset of its literals).

These optimizations are applied before calling the main Resolution function and contribute to reducing the complexity of the algorithm.

The knowledge base (KB) is read from a file, and the Resolution result for each clause written in that file is displayed below the corresponding clause.

For my KB, the output is UNSAT.

1.7 Resolution used for the propositional case

In this subsection, I present the sets of propositional clauses that were used to test my Resolution algorithm, along with the program's output indicating whether the result is SAT (Satisfiable) or UNSAT (Unsatisfiable).

Listing 1: Sets of propositional clauses

```
1. [[n(a),b],[c,d],[n(d),b],[n(b)],[n(c),b],[e],[a,b,n(f),f]] UNSAT
2. [[n(b),a],[n(a),b,e],[a, n(e)],[n(a)],[e]] UNSAT
3. [[n(a),b],[c,f],[n(c)],[n(f),b],[n(c),b]] SAT
4. [[a,b],[n(a),n(b)],[c]] SAT
```

2. SAT solver / Davis-Putnam Implementation

For the implementation of the Davis-Putnam algorithm (DP), I employed two strategies for selecting an atom p:

1. Choose p based on its frequency of appearance in KB.

2. Select p as the most balanced atom in KB.

Interestingly, in the majority of cases, the performance of my DP algorithm remains comparable, regardless of the chosen strategy. The most important difference between the two strategies becomes evident in a specific example. In this case, after a single run, the second strategy proves effective in finding a satisfactory solution, while the first strategy fails to compute it.

Upon debugging, I observed that if the atom p is the most frequently occurring one in the KB, the output for that particular clause would be 'no no no yes' after one run.

Listing 2: Sets of clauses for DP

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

DP1 output: no
n(child)/true n(toddler)/true
DP2 output: yes
child/true  female/true  boy/true  toddler/true  girl/true

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

DP1 output: no
n(child)/true n(toddler)/true
DP2 output: no
child/true  female/true  girl/true

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

DP1 output: no
b/true|_698
DP2 output: no
a/true  c/true  e/true  n(d)/true  b/true

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

DP1 output: no
n(a)/true n(e)/true
DP2 output: no
a/true

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

DP1 output: yes
n(b)/true n(a)/true
DP2 output: yes
e/true  b/true  f/true

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

DP1 output: no
n(b)/true n(a)/true
```

```
DP2 output: no
a/true  b/true
```

3. Source code

I wrote two different codes, for both Resolution and Davis-Putnam implementations.

3.1 Resolution implementation

Listing 3: Code for Resolution

```
negation(n(Y), Y):- !.
negation(Y, n(Y)).

concat_lists([], L2, L2).
concat_lists([X|L1], L2, [X|L3]):- concat_lists(L1, L2, L3).

elim_elem(_X, [], []).
elim_elem(X, [X|L], L):- !.
elim_elem(X, [Y|L], [Y|FinalList]):- elim_elem(X, L, FinalList).

check_res(A, L):- not(member(A, L)), not(is_taut(A)), not(is_subsumed(A, L)).

is_not_in_KB(_P, []).
is_not_in_KB(P, [L|KB]):- copy_term(P, P2), not(member(P2, L)), is_not_in_KB(P2, KB).

is_pure(L, KB):- member(P,L), negation(P, NP), copy_term(NP, NP2), is_not_in_KB(NP2, KB).

elim_pure_clauses([], [],_).
elim_pure_clauses([L|KB], KB1,P):- append(KB,P,KB0),is_pure(L, KB0),!,
    elim_pure_clauses(KB, KB1,P).
elim_pure_clauses([L|KB], [L|KB1],P):- elim_pure_clauses(KB, KB1,[L|P]).

elim_pure_clauses(A,B):-elim_pure_clauses(A,B,[]).

is_taut(L):- member(P, L), negation(P, NP), copy_term(NP, NP2), member(NP2, L),
    ground(P), ground(NP2).

elim_taut([], []).
elim_taut([L|KB], KB2):- is_taut(L), !, elim_taut(KB, KB2).
elim_taut([L|KB], [L|KB2]):- elim_taut(KB, KB2).

is_subsumed(L1, L2):- subset(L1, L2), L1 \== [].
```

```
elim_subsumed([], [], _).
elim_subsumed([L|KB], KB2, P):- append(KB, P, KB0), member(L2, KB0), copy_term(L2, SL),
    is_subsumed(SL, L), !, elim_subsumed(KB, KB2, P).
elim_subsumed([L|KB], [L|KB2], P):- elim_subsumed(KB, KB2, [L|P]).

elim_subsumed(KB, KB2):- elim_subsumed(KB, KB2, []).

final_res(BKB):- elim_pure_clauses(BKB, KB1), elim_taut(KB1, KB2),
    elim_subsumed(KB2, KB3), res(KB3).

res(KB):- member([], KB), !, write('UNSAT'), nl.
res(KB):-
    member(L1, KB), member(L2, KB), L1 \== L2,
    member(E1, L1), negation(E1, E2), copy_term(E2, X), member(X, L2),
    elim_elem(E1, L1, R1), elim_elem(X, L2, R2),
    concat_lists(R1, R2, R3), sort(R3, R4), check_res(R4, KB), res([R4|KB]).
res(_):- write('SAT'), nl.

afis_lista([]):- nl.
afis_lista([X|A]):- write(X), tab(2), afis_lista(A).
%res1(KB,KB4):- elim_pure_clauses(KB, KB2), elim_taut(KB2, KB3), elim_subsumed(KB3, KB4).

read_kb(Stream):- at_end_of_stream(Stream).

read_kb(Stream):- not(at_end_of_stream(Stream)), read_line_to_codes(Stream, KB),
    read_term_from_codes(KB, L, []), afis_lista(L),
    final_res(L),
    read_kb(Stream).

main:- open('/home/oana/Documents/master-I/KRR/reasoning.txt', read, Stream),
    read_kb(Stream),
    close(Stream).
```

3.2 Davis-Putnam implementation

Listing 4: Davis-Putnam code

```
negation(n(Y), Y):- !.
negation(Y, n(Y)).

no_occur(_X, [], 0).
no_occur(X, [X|T], Nr):- no_occur(X, T, Nr2), Nr is Nr2 + 1.
no_occur(X, [Y|T], Nr):- X \== Y, no_occur(X, T, Nr).

lit_with_most_occur(L, Lit):- flatten(L, OneL),
     setof(Nr-Lit, (member(Lit, OneL),no_occur(Lit, OneL, Nr)), LitList),
     max_member(_MaxNr-Lit, LitList).

find_balance(Lit, L, Balance):- negation(Lit, NLit),
     no_occur(Lit, L, PosNoLit),
     no_occur(NLit, L, NegNoLit),
     Balance is abs(PosNoLit - NegNoLit).

lit_most_balanced(KB, ChosenLit):-
     flatten(KB, OneL),
     setof(Balance-Lit, (member(Lit, OneL), find_balance(Lit, OneL, Balance)), Balances),
     keysort(Balances, SortedBalances),
     SortedBalances = [_MinBalance-ChosenLit|_].

%check(OneL, LitList):- setof(Nr-Lit, (member(Lit, OneL),no_occur(Lit, OneL, Nr)),
     LitList).

is_empty(L):- length(L, 0).


doesnt_contain_p_and_np(P, L):- not(member(P, L)), negation(P, NP), not(member(NP, L)).
first_dot(L, P, R):- include(doesnt_contain_p_and_np(P), L, R).



without_p(P, L):- not(member(P, L)).
with_np(NP, L):- member(NP, L).

elim_np_from_clause(C, NP2, Res1):- delete(C, NP2, Res1).

elim_np_from_clauses([], _P, []).
elim_np_from_clauses([C|L], P, [Res1|Res2]):- negation(P, NP),
     elim_np_from_clause(C, NP, Res1), elim_np_from_clauses(L, P, Res2).
%elim_np_from_clauses([C|L], P, [C|Res2]):- elim_np_from_clauses(L, P, Res2).

second_dot(L, P, R1):- negation(P, NP),
     include(without_p(P), L, R), include(with_np(NP), R, R2),
     elim_np_from_clauses(R2, P, R1).
```

```
dot_op(C, P, FinalResult):- first_dot(C, P, R),second_dot(C, P, R1),
    append(R, R1, FinalResult).


dp1([],[]):- write('yes'), nl.
dp1(C,_):- member([], C), !, fail.
dp1(C,[P/true|S]):- lit_with_most_occur(C, P), dot_op(C, P, L1), dp1(L1,S),!.
dp1(C,[P/false|S]):- lit_with_most_occur(C, P), negation(P, NP), dot_op(C, NP, L2),
    dp1(L2, S).

dp2([],[]):- write('yes'), nl.
dp2(C,_):- member([], C), write('no'), nl.
dp2(C,[P/true|S]):- lit_most_balanced(C, P), dot_op(C, P, L1), dp2(L1,S),!.
dp2(C,[P/false|S]):- lit_most_balanced(C, P), negation(P, NP), dot_op(C, NP, L2),
    dp2(L2, S).

afis_lista([]):- nl.
afis_lista([X|A]):- write(X), tab(2), afis_lista(A).

read_kb(Stream):- at_end_of_stream(Stream).

read_kb(Stream):- not(at_end_of_stream(Stream)),
    read_line_to_codes(Stream, KB),
    read_term_from_codes(KB, L, []), afis_lista(L),
    dp1(L, S1), afis_lista(S1),
    dp2(L, S2), afis_lista(S2), nl,
    read_kb(Stream).

main:- open('/home/oana/Documents/master-I/KRR/dp.txt', read, Stream),
    read_kb(Stream),
    close(Stream).
```

4. References

Both the Resolution and Davis-Putnam algorithms were implemented by adhering to the versions presented in the course materials, therefore based on *Ronald J. Brachman, Hector J. Levesque. Knowledge Representation and Reasoning, Morgan Kaufmann, 2004.*

For the development of helper functions in both programs, I referenced the following resources:

1. https://www.swi-prolog.org/pldoc/doc_for?object=root, accessed on December 4, 2023.
2. https://cs.union.edu/~striegnk/learn-prolog-now/html/node106.html, accessed on December 4, 2023.