Project 1 KRR

Ocnaru Mihai-Octavian - 407

December 8, 2025

1 Resolution

1.1 Knowledge base (natural language)

KB:

1. Every student who is enrolled in a course must complete assignments for that course.
2. Anyone who completes assignments needs access to reference books.
3. Reference books can only be borrowed by students with a valid library card.
4. All students enrolled in the university have a valid library card.
5. John is a student enrolled in the Artificial Intelligence course.

Query: Does John needs access to a reference book?

1.2 First order logic representation

1.2.1 Vocabulary definition

Predicates:

- $Student(x)$: x is a student.
- $ReferenceBook(b)$: b is a reference book.
- $Course(c)$: c is a course.
- $Enrolled(x, c)$: student x is enrolled in course c.
- $CompleteAssignments(x, c)$: person x must complete assignments for course c.
- $NeedsAccess(x, b)$: person x needs access to reference book b.
- $CanBorrow(x, b)$: person x can borrow reference book b.
- $HasLibraryCard(x)$: person x has a valid library card.

Constants:

- $John$.
- AI : artificial intelligence course.
- $ReferenceBooks$: as a category of reference books.

1.2.2 FOL translation

1. $\forall x \forall c(Student(x) \land Course(c) \land Enrolled(x,c) \implies CompleteAssignments(x,c))$
2. $\forall x \forall c(Student(x) \land Course(c) \land CompleteAssignments(x,c) \implies NeedsAccess(x, ReferenceBooks)$
3. $\forall x(CanBorrow(x, ReferenceBooks) \implies (HasLibraryCard(x) \land Student(x))$
4. $\forall x(Student(x) \implies HasLibraryCard(x))$
5. $Student(John) \land Enrolled(John, AI) \land Course(AI)$

Query: $NeedsAccess(John, ReferenceBooks)$

1.2.3 Converting KB from FOL to CNF

Statement 1:

$$\forall x\, \forall c\, (Student(x) \land Course(c) \land Enrolled(x,c) \implies CompleteAssignments(x,c))$$
$$\Rightarrow \forall x\, \forall c\, (\neg(Student(x) \land Course(c) \land Enrolled(x,c)) \lor CompleteAssignments(x,c))$$
$$\Rightarrow \forall x\, \forall c\, (\neg Student(x) \lor \neg Course(c) \lor \neg Enrolled(x,c) \lor CompleteAssignments(x,c))$$
$$\Rightarrow \neg Student(x) \lor \neg Course(c) \lor \neg Enrolled(x,c) \lor CompleteAssignments(x,c)$$

$$\{\neg Student(x), \neg Course(c), \neg Enrolled(x,c), CompleteAssignments(x,c)\} \quad (1)$$

Statement 2:

$$\{\neg Student(x), \neg Course(c), \neg CompleteAssignments(x,c), NeedsAccess(x, ReferenceBooks)\} \quad (2)$$

Statement 3:

$$\{\neg CanBorrow(x, ReferenceBooks), HasLibraryCard(x)\} \quad (3)$$

$$\{\neg CanBorrow(x, ReferenceBooks), Student(x)\} \quad (4)$$

Statement 4:

$$\{\neg Student(x), HasLibraryCard(x)\} \quad (5)$$

Statement 5:

$$\{Student(John)\}, \quad \{Enrolled(John, AI)\}, \quad \{Course(AI)\} \quad (6)$$

C1: $\{\neg Student(x), \neg Course(c), \neg Enrolled(x,c), CompleteAssignments(x,c)\}$
C2: $\{\neg Student(x), \neg Course(c), \neg CompleteAssignments(x,c), NeedsAccess(x, ReferenceBooks)\}$
C3: $\{\neg CanBorrow(x, ReferenceBooks), HasLibraryCard(x)\}$
C4: $\{\neg CanBorrow(x, ReferenceBooks), Student(x)\}$
C5: $\{\neg Student(x), HasLibraryCard(x)\}$
C6: $\{Student(John)\}$
C7: $\{Enrolled(John, AI)\}$
C8: $\{Course(AI)\}$
C9(Negated query): $\{\neg NeedsAccess(John, ReferenceBooks)\}$

1.2.4 Prove that question is logically entailed

C1:	$\{\neg Student(x), \neg Course(c), \neg Enrolled(x, c), CompleteAssignments(x, c)\}$
C6:	$\{Student(John)\}$
$\boldsymbol{\theta_1}$ **:**	$\{x/John\}$
C10:	$\{\neg Course(c), \neg Enrolled(John, c), CompleteAssignments(John, c)\}$
C8:	$\{Course(AI)\}$
$\boldsymbol{\theta_2}$ **:**	$\{c/AI\}$
C11:	$\{\neg Enrolled(John, AI), CompleteAssignments(John, AI)\}$
C7:	$\{Enrolled(John, AI)\}$
$\boldsymbol{\theta_3}$ **:**	$\{\}$
C12:	$\{CompleteAssignments(John, AI)\}$
C2:	$\{\neg Student(x), \neg Course(c), \neg CompleteAssignments(x, c), NeedsAccess(x, ReferenceBooks)\}$
C6:	$\{Student(John)\}$
$\boldsymbol{\theta_4}$ **:**	$\{x/John\}$
C13:	$\{\neg Course(c), \neg CompleteAssignments(John, c), NeedsAccess(John, ReferenceBooks)\}$
C8:	$\{Course(AI)\}$
$\boldsymbol{\theta_5}$ **:**	$\{c/AI\}$
C14:	$\{\neg CompleteAssignments(John, AI), NeedsAccess(John, ReferenceBooks)\}$
C12:	$\{CompleteAssignments(John, AI)\}$
$\boldsymbol{\theta_6}$ **:**	$\{\}$
C15:	$\{NeedsAccess(John, ReferenceBooks)\}$
C9:	$\{\neg NeedsAccess(John, ReferenceBooks)\}$
$\boldsymbol{\theta_7}$ **:**	$\{\}$
C16:	$\{\square\}$

$\implies$ The question is logically entailed from the KB.

The visual representation is available in figure 1.

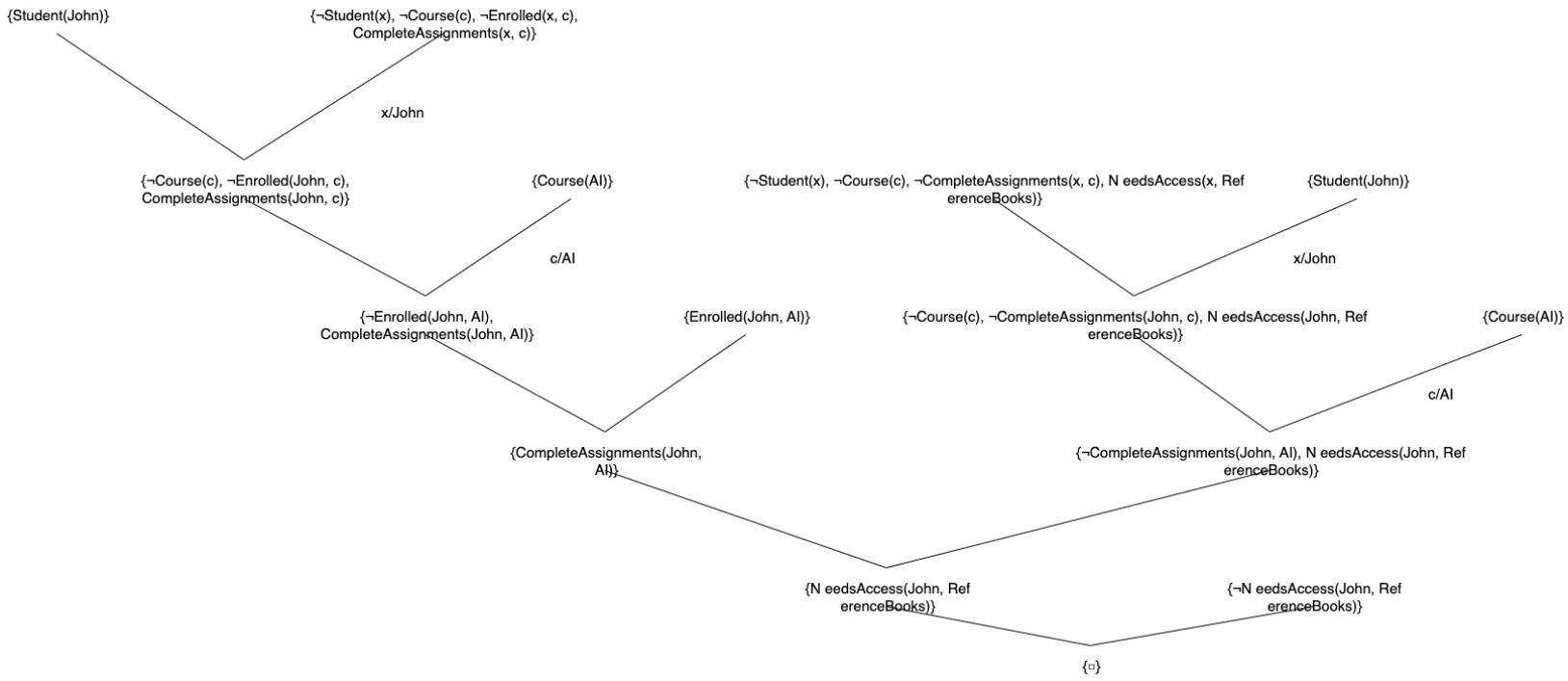

Figure 1: Resolution proof graph

1.3 Results of applying resolution implementation

The code implementation can be found in appendix A.

The knowledge base Prolog representation is made up of the statements below:

```
[neg(student(X)), neg(course(C)), neg(enrolled(X, C)), completeAssignments(X, C)].
[neg(student(X)), neg(course(C)), neg(completeAssignments(X, C)), needsAccess(X, referenceBooks)].
[neg(canBorrow(X, referenceBooks)), hasLibraryCard(X)].
[neg(canBorrow(X, referenceBooks)), student(X)].
[neg(student(X)), hasLibraryCard(X)].
[student(john)].
[enrolled(john, ai)].
[course(ai)].

% Question
[neg(needsAccess(john, referenceBooks))].
```

Running the query:

```
?- resolution([
[neg(student(X)), neg(course(C)), neg(enrolled(X, C)), completeAssignments(X, C)],
[neg(student(X)), neg(course(C)), neg(completeAssignments(X, C)), needsAccess(X, referenceBooks)],
[neg(canBorrow(X, referenceBooks)), hasLibraryCard(X)],
[neg(canBorrow(X, referenceBooks)), student(X)],
[neg(student(X)), hasLibraryCard(X)],
[student(john)],
[enrolled(john, ai)],
[course(ai)],
[neg(needsAccess(john, referenceBooks))]
], Result)
```

It will give us the result we expected:

```
Result = unsatisfiable
```

Applying it to the blocks problem will give:

```
?- resolution([
[on(a, b)],
[on(b, c)],
[green(a)],
[neg(green(c))],
[neg(green(X)), green(Y), neg(on(X, Y))]
], Result)

% Result = unsatisfiable
```

And for the plus knowledge base:

```
?- resolution([
[plus(zero,X,X)],
[neg(plus(X, Y, Z)), plus(succ(X), Y, succ(Z))],
[neg(plus(succ(succ(zero)),succ(succ(succ(zero))), _))]
], Result)

% Result = unsatisfiable
```

1.4 Applying resolution for various scenarios

1.4.1

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

% Result = unsatisfiable
```

1.4.2

```
[neg(b), a].
[neg(a), b, e].
[a, neg(e)].
[neg(a)].
[e].

% Result = unsatisfiable
```

1.4.3

```
[neg(a), b].
[c, f].
[neg(c)].
[neg(f), b].
[neg(c), b].

% Result = satisfiable
```

1.4.4

```
[a, b].
[neg(a), neg(b)].
[c].

% Result = satisfiable
```

2 SAT solver – The Davis Putnam procedure

I have implemented 2 strategies:

- *select_atom_shortest_clause*: which will pick the atoms from the shortest clause first
- *select_atom_most_balanced*: which will pick the atoms that have the smallest absolute difference between the number of occurrences of the positive atom and the negated version.

The code implementation can be found in appendix B.

2.1

```
Strategy: select_atom_most_balanced
----------------------------------------
Result: SATISFIABLE
Model: [toddler/true, child/true, female/true, male/true, boy/true, girl/true]
Steps: 7

Strategy: select_atom_shortest_clause
----------------------------------------
Result: SATISFIABLE
Model: [toddler/true, child/true, female/true, girl/true, male/true, boy/true]
Steps: 7
```

2.2

```
Strategy: select_atom_most_balanced
----------------------------------------
Result: UNSATISFIABLE
Steps: 1

Strategy: select_atom_shortest_clause
----------------------------------------
Result: UNSATISFIABLE
Steps: 1
```

2.3

```
Strategy: select_atom_most_balanced
----------------------------------------
Result: UNSATISFIABLE
Steps: 1

Strategy: select_atom_shortest_clause
----------------------------------------
Result: UNSATISFIABLE
Steps: 1
```

2.4

```
Strategy: select_atom_most_balanced
----------------------------------------
Result: UNSATISFIABLE
Steps: 1

Strategy: select_atom_shortest_clause
----------------------------------------
```

```
Result: UNSATISFIABLE
Steps: 1
```

2.5

```
Strategy: select_atom_most_balanced
----------------------------------------
Result: SATISFIABLE
Model: [e/true, b/true, f/true]
Steps: 4

Strategy: select_atom_shortest_clause
----------------------------------------
Result: SATISFIABLE
Model: [a/true, e/true, b/true, f/true]
Steps: 5
```

2.6

```
Strategy: select_atom_most_balanced
----------------------------------------
Result: UNSATISFIABLE
Steps: 1

Strategy: select_atom_shortest_clause
----------------------------------------
Result: UNSATISFIABLE
Steps: 1
```

A Resolution implementation

```
neg(neg(L), L) :- !.
neg(L, neg(L)).

is_tautology(C) :-
    member(L, C),
    neg(L, NL),
    member(L2, C),
    L2 == NL,
    !.

subsumes(General, Specific) :-
    copy_term(General, G),
    forall(
        member(L, G),
        member(L, Specific)
    ).

normalize(C, N) :- sort(C, N).

rename_var(C, R) :- copy_term(C, R).

resolve(C1, C2, Resolvent) :-
    rename_var(C1, R1),
    rename_var(C2, R2),
    select(L1, R1, Rest1),
    neg(L1, NL1),
    select(L2, R2, Rest2),
    unify_with_occurs_check(NL1, L2),
    append(Rest1, Rest2, Temp),
    normalize(Temp, Resolvent),
    \+ is_tautology(Resolvent).

all_resolvents(Clauses, Resolvents) :-
    findall(
        R,
        (
            member(C1, Clauses),
            member(C2, Clauses),
            C1 \== C2,
            resolve(C1, C2, R)
        ),
        All
    ),
    sort(All, Resolvents).

new_resolvents(Clauses, New) :-
    all_resolvents(Clauses, All),
    include(is_good_resolvent(Clauses), All, New).

is_good_resolvent(Clauses, R) :-
    \+ member(R, Clauses),
    \+ is_subsumed_by_any(R, Clauses).

is_subsumed_by_any(C, Clauses) :-
    member(C2, Clauses),
```

```
    C2 \== C,
    subsumes(C2, C),
    !.

saturate(Clauses, Result) :-
    (
        member([], Clauses) ->
        Result = unsatisfiable ;
        new_resolvents(Clauses, New),
        (
            New = [] ->
            Result = satisfiable ;
                member([], New) ->
                Result = unsatisfiable ;
                append(Clauses, New, Extended),
                list_to_set(Extended, Next),
                saturate(Next, Result)
        )
    ).

resolution(Clauses, Result) :-
    maplist(normalize, Clauses, Norm),
    exclude(is_tautology, Norm, NoTaut),
    list_to_set(NoTaut, Init),
    (
        member([], Init) ->
        Result = unsatisfiable ;
        saturate(Init, Result)
    ).
```

B Davis Putnam SAT implementation

```
neg(neg(L), L) :- !.
neg(L, neg(L)).

get_atom(neg(A), A) :- !.
get_atom(A, A).

bullet_op([], _, []).
bullet_op([Clause|Rest], Lit, Result) :-
    neg(Lit, NegLit),
    (
        member(Lit, Clause) ->
        bullet_op(Rest, Lit, Result) ;
        member(NegLit, Clause) ->
            select(NegLit, Clause, Reduced),
            bullet_op(Rest, Lit, RestResult),
            Result = [Reduced|RestResult] ;
            bullet_op(Rest, Lit, RestResult),
            Result = [Clause|RestResult]
    ).

collect_all_atoms([], []).
collect_all_atoms([Clause|Rest], Atoms) :-
    maplist(get_atom, Clause, ClauseAtoms),
    collect_all_atoms(Rest, RestAtoms),
    append(ClauseAtoms, RestAtoms, AllAtoms),
    list_to_set(AllAtoms, Atoms).

count_positive_negative([], _, 0, 0).
count_positive_negative([Clause|Rest], Atom, Pos, Neg) :-
    count_positive_negative(Rest, Atom, RestPos, RestNeg),
    (
        member(Atom, Clause) ->
            Pos is RestPos + 1,
            Neg = RestNeg ;
            member(neg(Atom), Clause) ->
                Pos = RestPos,
                Neg is RestNeg + 1 ;
                Pos = RestPos,
                Neg = RestNeg
    ).

balance_score(Atom, Clauses, Score) :-
    count_positive_negative(Clauses, Atom, Pos, Neg),
    Score is abs(Pos - Neg).

select_atom_most_balanced(Clauses, BestAtom) :-
    collect_all_atoms(Clauses, AllAtoms),
    AllAtoms = [FirstAtom|RestAtoms],
    balance_score(FirstAtom, Clauses, FirstScore),
    find_most_balanced(RestAtoms, Clauses, FirstAtom, FirstScore, BestAtom).

find_most_balanced([], _, BestAtom, _, BestAtom).
find_most_balanced([Atom|Rest], Clauses, CurrentBest, CurrentScore, BestAtom) :-
    balance_score(Atom, Clauses, Score),
    (
```

```
        Score < CurrentScore ->
            find_most_balanced(Rest, Clauses, Atom, Score, BestAtom) ;
            find_most_balanced(Rest, Clauses, CurrentBest, CurrentScore, BestAtom)
    ).

find_shortest_clause(Clauses, ShortestLength) :-
    maplist(length, Clauses, Lengths),
    min_list(Lengths, ShortestLength).

atom_in_shortest_clause(Atom, Clauses) :-
    find_shortest_clause(Clauses, MinLen),
    member(Clause, Clauses),
    length(Clause, MinLen),
    member(Lit, Clause),
    get_atom(Lit, Atom),
    !.

select_atom_shortest_clause(Clauses, Atom) :-
    atom_in_shortest_clause(Atom, Clauses).

dp_solve(_, [], yes([]), 1).
dp_solve(_, Clauses, no, 1) :-
    member([], Clauses),
    !.
dp_solve(Strategy, Clauses, Result, TotalSteps) :-
    call(Strategy, Clauses, Atom),
    (
        (
            bullet_op(Clauses, Atom, C1),
            dp_solve(Strategy, C1, SubResult, Steps1),
            SubResult = yes(Model)
        ) ->
        Result = yes([Atom/true|Model]),
        TotalSteps is Steps1 + 1
    ;
        (
            neg(Atom, NegAtom),
            bullet_op(Clauses, NegAtom, C2),
            dp_solve(Strategy, C2, SubResult, Steps2),
            SubResult = yes(Model)
        ) ->
        Result = yes([Atom/false|Model]),
        TotalSteps is Steps2 + 1
    ;
        Result = no,
        TotalSteps is 1
    ).

davis_putnam(Clauses, Strategy, Result, Steps) :-
    dp_solve(Strategy, Clauses, Result, Steps).
```