# DD1351 Logik för dataloger
# Laboration 2: Beviskontroll med Prolog

Simon Moradbakti & Nipun Thianwan

Hösten 2024

## Introduktion

I denna laboration ska man skapa och implementera en algoritm som kontrollerar om ett bevis, skrivet i naturlig deduktion, är korrekt. Detta ska göras genom att skriva ett program i Prolog. Indata till programmet är en sekvent och ett bevis, och programmet ska svara "yes" om det givna beviset är korrekt och visar att den angivna sekventen gäller, och "no" annars. Som man vet består naturlig deduktion av ett antal regler som beskriver när och hur nya formler kan härledas. Ett språk som passar väl för att kontrollera att sådana regler följs är Prolog, vilket vi kommer att använda i denna laboration. Labben kommer att anses vara klar när filen "run all tests.pl" ger en 'passed' notation bredvid på samtliga bevis, som visar att beviset är hanterat korrekt.

## Algoritm för Beviskontroll

En algoritm för beviskontroll är mycket viktig för den här uppgiften och kan beskrivas genom funktionerna i det Prolog programmet som gör upp algoritmen för beviskontroll. Bevis som programmet kan kontrollera är ihopsatta som Prolog listor, där listorna består av en huvudlista som innehåller en del underlistor som representerar rader i beviset med naturlig deduktion. Algoritmen utför beviskontrollen i flera steg. Algoritmen initieras av predikatet `verify`, den läser indatafilen och separerar innehållet i tre huvudkomponenter: premisser (`Presmises`), målet (`Goal`) och själva beviset (`Proof`). Därefter anropas `proof_verification`, där den inleder den egentliga kontrollen av beviset. Predikatet `proof_verification` består av tre påståenden (klauser) som avgör om beviset är korrekt eller inte. Om det första påståendet uppfylls, där bevisets mål och slutsats är identiska, så kommer svaret "Yes" returneras till terminalen. Om någon av de två andra påståenderna uppfylls, antingen på grund av att målet och slutsatsen inte matchar eller att någon rad i beviset är ogiltig, returneras svaret "No" istället. Om det första påståendet och målet är samma, så kommer den via `goal_reached`, att sedan anropa `proof_constructed` för att verifiera beviset, där varje rad måste kod kontrolleras.

## Kontroll av beviset

Predikatet `proof_constructed` kontrollerar varje rad i beviset och består i sin tur av tre huvudsakliga påståenden. Det första påståendet är ett baspåstående som lyckas och avslutar rekursionen om man får en tom lista, vilket endast kan hända när alla rader i beviset har kontrollerats. De två återstående påståenderna tillämpas rekursivt på varje rad i beviset. Det andra påståendet används specifikt för att hantera boxar. Det tredje påståendet kontrollerar varje rad genom att anropa predikatet `rule_is_valid`, som innehåller regler för att verifiera att tillhörande regel tillämpats korrekt för varje rad. Om raden är korrekt, läggs den till i en lista för andra kontrollerade rader. Den listan fylls först med premisser, vilket säkerställs genom att jämföra den raden med en såkallad premisslista, och används sedan för att uttnyttja härledningsregler, som till exempel och-elimination samt implikation-introduktion, detta görs för att successivt bygga upp listan av kontrollerade rader.

## Beviskontroll inom boxar

Kontrollen av boxar i beviset måste hanteras lite speciellt då reglerna för boxar måste följas på ett speciellt sätt. Ett exempel är att ingen rad får hänvisa till en rad i en tidigare stängd box, och vissa bevisregler kräver att ett antagande görs i boxens början och att en slutsats nås först i slutet av boxen.
Boxhanteringen initieras genom att det andra påståendet i `proof_constructed` predikatet, aktiveras när en rad som består av ett antagande också är följt av en lista. Detta är det som markerar starten på en box, och boxen utvärderas rekursivt genom predikatet `box_verified`, som bearbetar varje rad inom boxen tills slutet av boxen nås.
Predikatet `box_verified` innehåller också tre påståenden. Första påståendet kommer att avsluta rekursionen när den får en tom lista, vilket endast sker när boxens slut har nåtts. Det andra påståendet aktiveras om en ny, inre box måste skapas inom den första boxen, och den nya boxen utvärderas rekursivt från början till slut, innan kontrollen av den yttre boxen kan fortsätta. Det tredje påståenden läser en rad och kontrollerar om den är korrekt genom `rule_is_valid`. När alla rader i en box har kontrollerats, lagras boxen som en lista tillsammans med eventuella andra övriga kontrollerade rader i beviset. Då boxen lagras som en separat lista är det inte tillåtet att hänvisa till rader inom boxen utanför sammanhanget.

## Regler för boxreferenser

Vissa utav reglerna för naturlig deduktion kräver även specifika hänvisningar till början och slutet av en box. Predikatet `find_box` används för att verifiera dem hänvisningarna, och söker igenom listan över kontrollerade bevis efter en lista med det angivna radnumret och uttrycket genom predikatet `is_box`. Om den efterfrågade boxen hittas kontrolleras dess startrad och slutrad för att säkerställa att dessa följer de angivna linjenumren.

## Predikat och dess sanningstabell

| **Predikat** | **Sant När** | **Falskt När** |
|---|---|---|
| `verify/1` | Filen läses korrekt, innehåller premisser, mål och bevis. | Filen kan inte läsas eller innehåller ogiltiga data. |
| `validate_proof/1` | Beviset verifieras med korrekta premisser, mål och bevisformat. | Verifieringen misslyckas på grund av oläsbar fil eller felaktigt dataformat. |
| `proof_verification/3` | Målet uppnås och alla regler i beviset tillämpas korrekt. | Målet matchar inte slutsatsen eller någon regel tillämpas felaktigt. |
| `goal_reached/2` | Målet matchar den sista raden i beviset. | Målet matchar inte den sista raden i beviset. |
| `proof_constructed/3` | Varje rad i beviset är giltig enligt regler och antaganden. | En rad i beviset är ogiltig eller en regel tillämpas felaktigt. |
| `box_verified/3` | Alla rader i rutan är korrekta och följer antaganden och slutsatser. | En rad i rutan är ogiltig, eller rutan hänvisar till en stängd rad. |
| `is_box/5` | Rutan innehåller de specificerade start- och slutraderna. | Rutan saknar de specificerade raderna eller har felaktiga uttryck. |
| `find_box/3` | Hittar en specifik ruta i listan över verifierade bevis. | Rutan finns inte i listan över verifierade bevis. |
| `rule_is_valid/3` | Raden uppfyller alla krav för den specificerade regeln (t.ex. `premiss`, `andel1`). | Regelvillkor är inte uppfyllda eller referenser saknas i verifierade bevis. |

Table 1: Prolog-predikat i programmet och deras villkor för sanning och falskhet

## Appendix A - Prolog kod

```
% Alias for compatibility with the test suite
verify(File) :- validate_proof(File).

%  Reading input data from file
validate_proof(InputFile) :-
    see(InputFile),
    read(Premises),
    read(Goal),
    read(Proof),
    seen,
    proof_verification(Premises, Goal, Proof).

%  Verifying if the proof is valid
% Case where the proof is valid
proof_verification(Premises, Goal, Proof) :-
    goal_reached(Goal, Proof),
    proof_constructed(Premises, Proof, []),
    write("Yes").
% Case where the proof is invalid
proof_verification(_, Goal, Proof) :-
    \+ goal_reached(Goal, Proof), !,
    write("No"),
    fail.
proof_verification(Premises, _, Proof) :-
    \+ proof_constructed(Premises, Proof, []),
    write("No"),
    fail.

%  Goal verification
% Base case: checks if the goal matches the last statement
goal_reached(Goal, [[_, LastExpr, _]|[]]) :-
    Goal = LastExpr.
% Recursive case to find the goal in the proof
goal_reached(Goal, [_|Rest]) :-
    goal_reached(Goal, Rest).

%  Proof construction verification

% Base case: if the proof is empty, it s valid
proof_constructed(_, [], _).
% Handling assumptions in a box
proof_constructed(Premises, [[[LineNo, Expr, 'assumption']|
    BoxTail]|Rest], VerifiedProofs) :-
    box_verified(Premises, BoxTail, [[LineNo, Expr, '
        assumption']|VerifiedProofs]),
    proof_constructed(Premises, Rest, [[[LineNo, Expr, '
        assumption']|BoxTail]|VerifiedProofs]).
```

```
% Regular line in the proof
proof_constructed(Premises, [Line|Rest], VerifiedProofs) :-
    rule_is_valid(Premises, Line, VerifiedProofs),
    proof_constructed(Premises, Rest, [Line|VerifiedProofs])
        .

%  Box verification

% Base case: if the box is empty, it s valid
box_verified(_, [], _).
% Nested box structure within a box
box_verified(Premises, [[[LineNo, Expr, 'assumption']|
    BoxTail]|Rest], VerifiedProofs) :-
    box_verified(Premises, BoxTail, [[LineNo, Expr, '
        assumption']|VerifiedProofs]),
    box_verified(Premises, Rest, [[[LineNo, Expr, '
        assumption']|BoxTail]|VerifiedProofs]).
% Regular line in a box
box_verified(Premises, [Line|Rest], VerifiedProofs) :-
    rule_is_valid(Premises, Line, VerifiedProofs),
    box_verified(Premises, Rest, [Line|VerifiedProofs]).

% Checks if a box with specified line numbers exists
is_box(LineNo, LineNo2, VerifiedProofs, Expr, Expr2) :-
    find_box(LineNo, VerifiedProofs, Box),
    member([LineNo, Expr, _], Box),
    member([LineNo2, Expr2, _], Box).

% Finds the specific box containing the given line number
find_box(LineNo, [Box|_], Box) :-
    member([LineNo, _, _], Box).
find_box(LineNo, [_|Rest], Box) :-
    find_box(LineNo, Rest, Box).

%  Rule validation
% premise rule
rule_is_valid(Premises, [_, Expr, 'premise'], _) :-
    member(Expr, Premises).
% copy(x) rule
rule_is_valid(_, [_, Expr, copy(LineNo)], VerifiedProofs) :-
    member([LineNo, Expr, _], VerifiedProofs).
% andint(x, y) rule
rule_is_valid(_, [_, and(X, Y), andint(LineNo1, LineNo2)],
    VerifiedProofs) :-
    member([LineNo1, X, _], VerifiedProofs),
    member([LineNo2, Y, _], VerifiedProofs).
% andel1(x) rule
rule_is_valid(_, [_, Expr, andel1(LineNo)], VerifiedProofs)
    :-
    member([LineNo, and(Expr, _), _], VerifiedProofs).
```

```
% andel2(x) rule
rule_is_valid(_, [_, Expr, andel2(LineNo)], VerifiedProofs)
    :-
    member([LineNo, and(_, Expr), _], VerifiedProofs).
% orint1(x) rule
rule_is_valid(_, [_, or(X, _), orint1(LineNo)],
    VerifiedProofs) :-
    member([LineNo, X, _], VerifiedProofs).
% orint2(x) rule
rule_is_valid(_, [_, or(_, Y), orint2(LineNo)],
    VerifiedProofs) :-
    member([LineNo, Y, _], VerifiedProofs).
% orel(x, y, z, u, v) rule
rule_is_valid(_, [_, Expr, orel(Line1, Line2, Line3, Line4,
    Line5)], VerifiedProofs) :-
    member([Line1, or(Expr2, Expr3), _], VerifiedProofs),
    is_box(Line2, Line3, VerifiedProofs, Expr2, Expr),
    is_box(Line4, Line5, VerifiedProofs, Expr3, Expr).
% impint(x, y) rule
rule_is_valid(_, [_, imp(Expr, Expr2), impint(Line1, Line2)
    ], VerifiedProofs) :-
    is_box(Line1, Line2, VerifiedProofs, Expr, Expr2),
    find_box(Line1, VerifiedProofs, Box),
    last(Box, [Line2, Expr2, _]).
% impel(x, y) rule
rule_is_valid(_, [_, Expr, impel(LineNo1, LineNo2)],
    VerifiedProofs) :-
    member([LineNo1, Expr2, _], VerifiedProofs),
    member([LineNo2, imp(Expr2, Expr), _], VerifiedProofs).
% negint(x, y) rule
rule_is_valid(_, [_, neg(Expr), negint(LineNo1, LineNo2)],
    VerifiedProofs) :-
    is_box(LineNo1, LineNo2, VerifiedProofs, Expr, 'cont').
% negel(x, y) rule
rule_is_valid(_, [_, 'cont', negel(LineNo1, LineNo2)],
    VerifiedProofs) :-
    member([LineNo1, Expr, _], VerifiedProofs),
    member([LineNo2, neg(Expr), _], VerifiedProofs).
% contel(x) rule
rule_is_valid(_, [_, _, contel(LineNo)], VerifiedProofs) :-
    member([LineNo, 'cont', _], VerifiedProofs).
% negnegint(x) rule
rule_is_valid(_, [_, neg(neg(Expr)), negnegint(LineNo)],
    VerifiedProofs) :-
    member([LineNo, Expr, _], VerifiedProofs).
% negnegel(x) rule
rule_is_valid(_, [_, Expr, negnegel(LineNo)], VerifiedProofs
    ) :-
    member([LineNo, neg(neg(Expr)), _], VerifiedProofs).
% mt(x, y) rule
```

```
rule_is_valid(_, [_, neg(Expr), mt(Line1, Line2)],
    VerifiedProofs) :-
    member([Line1, imp(Expr, Expr2), _], VerifiedProofs),
    member([Line2, neg(Expr2), _], VerifiedProofs).
% pbc(x, y) rule
rule_is_valid(_, [_, Expr, pbc(LineNo1, LineNo2)],
    VerifiedProofs) :-
    is_box(LineNo1, LineNo2, VerifiedProofs, neg(Expr), '
        cont').
% lem rule
rule_is_valid(_, [_, or(Expr, neg(Expr)), 'lem'], _).
```

# Appendix B - Prologbevis

## Invalid.txt

```
[imp(p, imp(q, r))].

imp(and(q, neg(r)), neg(p)).

[
    [1, imp(p, imp(q, r)), premise],
    [
        [2, and(q, neg(r)), assumption],
        [3, q, andel1(2)],
        [
            [4, p, assumption],
            [5, imp(q, r), impel(4, 1)],
            [6, r, impel(3, 5)],
            [7, neg(r), andel2(2)],
            [8, cont, negel(6, 7)]
        ],
        [9, neg(and(q, neg(r))), negint(4, 8)],
        [10, cont, negel(2, 9)],
        [11, neg(p), contel(10)]
    ],
    [12, imp(and(q, neg(r)), neg(p)), impint(2, 11)]
].
```

### Valid.txt

```
[imp(p, imp(q, r))].

imp(and(q, neg(r)), neg(p)).

[
    [1, imp(p, imp(q, r)), premise],
    [
        [2, and(q, neg(r)), assumption],
        [
            [3, p, assumption],
            [4, imp(q, r), impel(3, 1)],
            [5, q, andel1(2)],
            [6, r, impel(5, 4)],
            [7, neg(r), andel2(2)],
            [8, cont, negel(6, 7)]
        ],
        [9, neg(p), negint(3, 8)]
    ],
    [10, imp(and(q, neg(r)), neg(p)), impint(2, 9)]
].
```

### Körning av alla bevis från uppgiften

```
PS C:\Users\Simon Moradbakti\Pictures\Logik f r dataloger\
    IV1351\tests> swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.8)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free
    software.
Please run ?- license. for legal details.
For online help and background, visit https://www.swi-prolog
    .org
For built-in help, use ?- help(Topic). or ?- apropos(Word).
1 ?- ['run_all_tests'].
true.
2 ?- run_all_tests('../testkod.pl').
valid01.txtYes passed
valid02.txtYes passed
valid03.txtYes passed
valid04.txtYes passed
valid05.txtYes passed
valid06.txtYes passed
valid07.txtYes passed
valid08.txtYes passed
valid09.txtYes passed
valid10.txtYes passed
```

```
valid11.txtYes passed
valid12.txtYes passed
valid13.txtYes passed
valid14.txtYes passed
valid15.txtYes passed
valid16.txtYes passed
valid17.txtYes passed
valid18.txtYes passed
valid19.txtYes passed
valid20.txtYes passed
valid21.txtYes passed
valid22.txtYes passed
valid23.txtYes passed
invalid01.txtNo passed
invalid02.txtNo passed
invalid03.txtNo passed
invalid04.txtNo passed
invalid05.txtNo passed
invalid06.txtNo passed
invalid07.txtNo passed
invalid08.txtNo passed
invalid09.txtNo passed
invalid10.txtNo passed
invalid11.txtNo passed
invalid12.txtNo passed
invalid13.txtNo passed
invalid14.txtNo passed
invalid15.txtNo passed
invalid16.txtNo passed
invalid17.txtNo passed
invalid18.txtNo passed
invalid19.txtNo passed
invalid20.txtNo passed
invalid21.txtNo passed
invalid22.txtNo passed
invalid23.txtNo passed
invalid24.txtNo passed
invalid25.txtNo passed
invalid26.txtNo passed
invalid27.txtNo passed
invalid28.txtNo passed
invalid29.txtNo passed
invalid30.txtNo passed
invalid31.txtNo passed
invalid32.txtNo passed
invalid33.txtNo passed
invalid34.txtNo passed
invalid35.txtNo passed
invalid36.txtNo passed
```