# LAB 2 : Rapport

## Beviskontroll-algoritmen

Vår beviskontroll bygger på att vi, för varje rad i beviset (varje rad är en lista), undersöker raden (rad numret, sats-uttrycket samt regeln som tillämpades) och kontrollera att raden är korrekt. Med "korrekt" menar vi att raden kan vara där den är, samt att inget inkonsekvent finns i den (exempelvis att det använts en regel och fått ut ett sats-uttryck som är helt felaktigt. Även att regeln som applicerades är på en rad som inte är giltig att använda regeln med). För att applicera ovanstående procedur, är inbyggda "member" fundamentet då den kontrollerar om inputen till varje regel finns redan definerad i en rad, för att vi ska kunna applicera regeln i en senare rad.

## Boxhantering

```
check_row(Prems, Goal, [[_, _,assumption]|T], Verifiedlines) :-
   valid_proof(Prems, Goal, T, [[_, _,assumption]|Verifiedlines]).
```

Boxar hanteras med hjälp av detta predikat som binder en rad med en assumption i sig till en lista av listor (dvs själva boxen). När detta är uppfyllt så anropas valid_proof för att gå genom alla rader i boxen och försöka verifiera dem med check_row. En box blir i princip ett bevis i sig själv (som är inlagt i det stora beviset). När det finns nästlade boxar så följer programmet samma princip

| Namn på predikat | Argument | När blir predikatet sant? |
|---|---|---|
| checkGoal | 2 st argument: Slutsatsen och hela beviset | Om sats-uttrycket i sista raden i beviset är identiskt till slutsatsen |
| check_row | 4 st argument: listan som innehåller premisserna, slutsatsen, en lista som representerar en rad i beviset, en lista som utökas för varje rad som verifieras i beviset | Det blir sant när varje rad i beviset kan verifieras med hjälp av raderna som är redan verifierade (dvs listorna i verifiedLines). |

| | | |
|---|---|---|
| | Varje regelapplicering har sitt eget "check_row" predikat som kollar ifall en viss regelapplicering stämmer eller inte. Detta brukar göras till exempel med hjälp av radnummer i bevis.<br><br>check_row(_,_,[Row, and(P,Q), andint(R1, R2)], Verifiedlines):-<br>Row > R1, Row > R2,<br>member([R1, P, _], Verifiedlines),<br>member([R2, Q, _], Verifiedlines).<br><br>Detta predikat kollar att introduktion av AND är korrekt genom att se raderna som innehåller P och Q står ovanför i beviset.<br><br>check_row(Prems, _, [_, P, premise], VerifiedLines) :- member(P,Prems).<br><br>Detta predikat ser till att en rad med premis i beviset innehåller samma sats-uttrycket som Prems(dvs den som vi läser in från txt fil) | |
| valid_proof | 4 st argument: listan som innehåller premisserna, slutsatsen, beviset och listan som utökas för varje rad som verifieras i beviset. | Det blir sant för varje bevis som är korrekt (men bevisets sats-uttryck i sista raden måste vara identiskt till slutsatsen). Det är ett rekursivt predikat och körs på varje rad i Proof (dvs listan med bevis som vi läser in från text fil) |
| verify | Namnet på filen som innehåller beviset som vi vill verifiera | Verify blir sant när premisser, slutsats och bevis är inlästa samt att satsuttryck i sista raden är samma som slutsats. Dessutom ska validproof också vara sant(kolla hur validproof fungerar nere i denna tabell) |

## Appendix

Här nedan följer hela programmet.

```
verify(InputFileName) :-
    see(InputFileName),
    read(Prems), read(Goal), read(Proof),
    seen,
    checkGoal(Goal, Proof),
    valid_proof(Prems, Goal, Proof, []),!.


valid_proof(_, _, [], _).

valid_proof(Prems, Goal, [H|T], Verifiedlines) :-
    check_row(Prems, Goal, H, Verifiedlines),
    valid_proof(Prems, Goal, T, [H|Verifiedlines]).

checkGoal(Goal, Proof):-
    last(Proof, LastRow),
    nth1(2, LastRow, Goal).

%%check premise
check_row(Prems, _, [_, P, premise], VerifiedLines) :- member(P,Prems).

%%andint
check_row(_,_,[Row, and(P,Q), andint(R1, R2)], Verifiedlines):-
    Row > R1, Row > R2,
    member([R1, P, _], Verifiedlines),
    member([R2, Q, _], Verifiedlines).

%%andel1
check_row(_,_,[_, P, andel1(Row)], Verifiedlines) :-
    member([Row, and(P,_), _], Verifiedlines).

%%andel2
check_row(_,_,[_, Q, andel2(Row)], Verifiedlines) :-
    member([Row, and(_,Q), _], Verifiedlines).

%%orint1
```

```
check_row(_,_,[_, or(P,Q), orint1(Row)], Verifiedlines) :-
    member([Row, P, _], Verifiedlines).

%%orint2
check_row(_,_,[_,or(P,Q), orint2(Row)], Verifiedlines) :-
    member([Row, Q, _], Verifiedlines).

%%impel
check_row(_,_,[Row,Q,impel(R1,R2)],Verifiedlines) :-
    member([R1, P, _], Verifiedlines), member([R2, imp(P,Q), _],
Verifiedlines), Row > R1, Row > R2.

%%negel
check_row(_,_,[Row, cont, negel(R1, R2)], Verifiedlines) :-
    member([R1, P, _], Verifiedlines), member([R2, neg(P), _],
Verifiedlines).

%%MT
check_row(_,_,[Row,neg(P),mt(R1,R2)], Verifiedlines) :-
    member([R1,imp(P,Q),_], Verifiedlines), member([R2,neg(Q),_],
Verifiedlines), Row > R1, Row > R2.

%%LEMDISC
check_row(_,_,[_, or(P, neg(P)), lem], Verifiedlines).


%%negnegint
check_row(_,_,[_, neg(neg(P)), negnegint(Row)], Verifiedlines) :-
    member([Row, P, _], Verifiedlines).

%%negnegel
check_row(_,_,[_, P, negnegel(Row)], Verifiedlines) :-
    member([Row, neg(neg(P)), _], Verifiedlines).

%%contel
check_row(_,_,[_, _, contel(Row)], Verifiedlines) :-
    member([Row, cont, _], Verifiedlines).

%%copy
check_row(_,_,[_, P, copy(Row)], Verifiedlines):-
    member([Row, P, _], Verifiedlines).

%%findbox
```

```
check_row(Prems, Goal, [[_, _,assumption]|T], Verifiedlines) :-
    valid_proof(Prems, Goal, T, [[_, _, assumption]|Verifiedlines]).

%%orel
check_row(_,_,[_, Ans, orel(X, Y, U, V, W)], Verifiedlines) :-
    member(List1, Verifiedlines),
    member(List2, Verifiedlines),
    member([X, or(P,Q), _], Verifiedlines),
    member([Y, P, assumption], List1),
    member([U, Ans, _], List1),
    member([V, Q, assumption], List2),
    member([W, Ans, _], List2).

%%impint
check_row(_,_,[_, imp(P,Q), impint(R1,R2)], Verifiedlines):-
    member(List, Verifiedlines),
    member([R1, P, assumption], List),
    member([R2, Q, _], List).

%%negint
check_row(_,_,[_, neg(P), negint(R1, R2)], Verifiedlines) :-
    member(List, Verifiedlines),
    member([R1, P, _], List),
    member([R2, cont, _], List).

%%PBC
check_row(_,_,[_, P, pbc(R1, R2)], Verifiedlines) :-
    member(List, Verifiedlines),
    member([R1, neg(P), assumption], List),
    member([R2, cont, _], List).
```

## Exempelbevisen:

Exempel på ett icke-trivial korrekt bevis, med box:

[or(and(p, q), r)].

or(p, or(q, r)).

```
[
   [1, or(and(p, q), r),    premise],
   [
     [2, and(p, q),         assumption],
     [
       [3, p,               assumption],
       [4, or(p, or(q, r)), orint1(3)]
     ],
     [
       [5, q,               assumption],
       [6, or(q, r),        orint1(5)],
       [7, or(p, or(q, r)),  orint2(6)]
     ],
     [8, or(p, or(q, r)),   orel(2,3,4,5,7)]
   ],
   [
     [9, r,                 assumption],
     [10, or(q, r),         orint2(9)],
     [11, or(p, or(q, r)),  orint2(10)]
   ],
   [12, or(p, or(q, r)),    orel(1,2,8,9,11)]
].
```

Exempel på ett icke-trivialt felaktigt bevis, med box:

[imp(q, r)].

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

```
[
   [1, imp(q, r),    premise],
   [
     [2, and(p, q),         assumption],
     [
       [3, p,               assumption],
       [4, and(p, r), orint1(3)]
     ],
     [
       [5, q,               assumption],
       [6, r,        impel(1,5)],
       [7, or(p, r),  orint2(6)]
     ]
   ],
   [8, imp(and(p, q), and(p, r)), impint(2,7)]
].
```