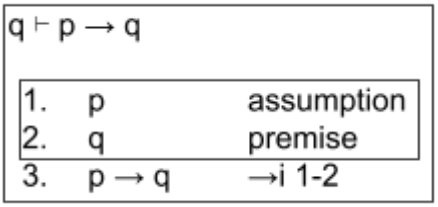
Labb 2, Beviskoll i prolog

DD1351, Logik för dataloger
Namn: Pim Erlandsson och Arvid Holm
Datum: 25/11-2021
CINTE

Introduktion

Labb 2 går ut på att kunna kontrollera bevis skrivet i naturlig deduktion. Indatan till programmet är en sekvent och ett bevis på ett speciellt format. Programmet ska alltså kunna verifiera om beviset är korrekt eller inte.

Exempel på indataformat.




```
[q].

imp(p,q).

[
 [
  [1,  p,     assumption],
  [2,  q,     premise]
 ],
 [3,   imp(p,q), impint(1,2)]
].
```

Beviskontrolls algoritmen

Vår algoritm använder sig framförallt av predikatet check_rule men med olika parametrar. check_rule kallas från valid_proof och har premissen, målet och beviset som parametrar tillsammans med en tom lista. Den tomma listan använder vi för att spara varje rad i beviset som vi redan har verifierat, den kommer kallas för “Prev” i resten av texten. De olika check_rule predikaten matchar med element i bevislistan för att sedan verifieras på lämpligt sätt beroende på vilken regel som applicerats i beviset. Om bevisraden är korrekt kallas check_rule igen, men denna gång skickar vi endast in svansen av bevislistan för gå till nästa rad i beviset. Dessutom sparar vi den verifierade raden i Prev för att senare kunna bekräfta regler som använder sig av olika tidigare rader.
När funktionen check_rule anropas studerar denna parametrarna som skickas med vilket inputfilen innehåller. Algoritmen inleder med att läsa in premisserna, sedan målet som ska bevisas och efter detta läser vi in själva beviset.

Boxhantering

För att hantera boxar söker vi efter en lista som är ett element i en lista, vilket är boxen som finns i en lista med bevis. Listan vi söker innehåller ett okänt radnummer, ett okänt “resultat” men regeln som applicerats måste vara “assumption” eftersom att varje box börjar med ett antagande. Uppfyller den det kravet ska vi bekräfta att boxen och reglerna som appliceras i den är korrekta. För att göra det sätter vi antagandet i Prev och sedan använder vi oss av check_rule med boxen som bevislista. Om alla regler är korrekta så sparar vi hela boxen i Prev.

```
check_rule(Prems, Goal, [[[Linum, Res, assumption]|Tbox]|Tproof], Prev) :-
  last([[Linum, Res, assumption]|Tbox], [_, BoxGoal, _]),
  check_rule(Prems, BoxGoal, Tbox, [[Linum,Res]|Prev]),
  check_rule(Prems, Goal, Tproof, [[[Linum, Res, assumption]|Tbox]|Prev]).
```

För att komma åt innehållet i boxarna från Prev använder vi oss av extBox (extractBox). Om en regel som använder en eller flera boxar ska kontrolleras ber vi om en box som börjar på ett speciellt radnummer från Prev. I extBox kollar vi om första elementet i Prev har rätt radnummer, “resultat” och att regeln är “assumption”. Om det inte stämmer går vi till nästa element i Prev och testar samma sak igen, tills det att vi hittat boxen. Vid det laget kan man kontrollera om de olika kraven på boxen uppfylls för att regeln ska vara korrekt.

```
extBox(Linum, [Box|_], Box) :-
  member([Linum,_,assumption], Box).

extBox(Linum, [_|Tbox], _) :-
  extBox(Linum, Tbox, _).
```

Boxhanteringen sker genom funktionen check_rule som rekursivt kallas och känner av vad för typ av problem som betraktas. När vi stöter på en box i beviset känner denna av detta genom att boxen startar med [[och avslutar boxen med]] samt att första raden av beviset innehåller ett antagande. När vi stöter på en box inleder vi med att ta ut resultatet på sista raden i boxen, detta

Predikat	Funktionalitet och parametrar
check_rule(Prems, Goal, Proof, Prev)	Detta predikat anropas med parametrarna premisser, mål, bevis och Prev. Beroende på parametrarnas värde kommer raden att testas på det sätt som krävs beroende av vilken av regel som ska betraktas. check_rule betraktar hela beviset, rad för rad tills det att alla bevis är verifierade. För att göra detta kollas regeln som används i beviset och ifall dessa är korrekt utförda. Detta sker rekursivt genom att denna till en början läser in första raden av beviset och kollar ifall denna stämmer, sedan kallas funktionen rekursivt med nästa rad och så vidare tills att vi gått igenom hela beviset. check_rule är sann då vårt basfall är uppfyllt samt då resultatet på sista raden stämmer överens med målet.
extBox	Detta predikat används för att extrahera boxar ur Prev och lägga dessa i en enskild lista för att lättare kunna arbeta med dem. Detta predikat är sann då en box har funnits i beviset och falsk annars.
valid_proof(Prems, Goal, Proof)	valid_proof anropas med premisser, mål och bevis och anropar sedan check_rule med dessa samt med den tomma listan Prev.
verify(InputFileName)	Tar emot namnet på filen med indata och läser in premiss, mål och bevis, kallar sedan på valid_proof.

Appendix A:

Programmet: "beviskoll.pl"

```
% beviskoll.pl
verify(InputFileName) :- see(InputFileName),
                         read(Prems), read(Goal), read(Proof),
                         seen,
                         valid_proof(Prems, Goal, Proof).

valid_proof(Prems, Goal, Proof) :-
  check_rule(Prems, Goal, Proof, []).

% När bevislistan är tom, jämför sista resultatet som testades med målet.
check_rule(Prems, Goal, [], [[Linum,Res]|Prev]) :-
  Goal == Res.

% Om regeln som använts är "premise" ska resultatet finnas i listan med
premisser.
% Hitta ett element i Proof som innehåller premise.
check_rule(Prems, Goal, [[Linum, Res, premise]|Tproof], Prev) :-
  % Finns resultatet i premisslistan.
  member(Res, Prems),
  % Gå till nästa rad i beviset genom att skicka med svansen från bevislistan
  % och lägg till den bekräftade raden i Prev
  check_rule(Prems, Goal, Tproof, [[Linum,Res]|Prev]).
```

```
% andint
check_rule(Prems, Goal, [[Linum, Res, andint(X,Y)]|Tproof], Prev) :-
  % Hitta det första elementet på rad X och det andra elementet på rad Y
  member([X,Z1], Prev),
  member([Y,Z2], Prev),
  % Kolla om elementen vi hittat på rad X och Y är samma som finns på den
  % aktuella raden.
  and(Z1,Z2) == Res,
  check_rule(Prems, Goal, Tproof, [[Linum,Res]|Prev]).

% andel
check_rule(Prems, Goal, [[Linum, Res, andel1(X)]|Tproof], Prev) :-
  % Hitta en and() på rad X där första elementet är Res
  member([X, and(Res, Z)], Prev),
  check_rule(Prems, Goal, Tproof, [[Linum,Res]|Prev]).

check_rule(Prems, Goal, [[Linum, Res, andel2(X)]|Tproof], Prev) :-
  % Hitta en and() på rad X där andra elementet är Res
  member([X, and(Z, Res)], Prev),
  check_rule(Prems, Goal, Tproof, [[Linum,Res]|Prev]).

% orint
check_rule(Prems, Goal, [[Linum, or(Z,W), orint1(X)]|Tproof], Prev) :-
  % Om det finns ett element Z på rad X stämmer det.
  member([X,Z],Prev),
  check_rule(Prems, Goal, Tproof, [[Linum,or(Z,W)]|Prev]).

check_rule(Prems, Goal, [[Linum, or(Z,W), orint2(X)]|Tproof], Prev) :-
  % Om det finns ett element W på rad X stämmer det.
  member([X,W],Prev),
  check_rule(Prems, Goal, Tproof, [[Linum,or(Z,W)]|Prev]).

% impel
check_rule(Prems, Goal, [[Linum, Res, impel(X,Y)]|Tproof], Prev) :-
  % Hitta elementet Z på rad X
  member([X,Z], Prev),
  % På rad Y ska det finns en imp() där Z implicerar resultatet
  member([Y,imp(Z, Res)], Prev),
  check_rule(Prems, Goal, Tproof, [[Linum,Res]|Prev]).

% negel
check_rule(Prems, Goal, [[Linum, Res, negel(X,Y)]|Tproof], Prev) :-
  % Hitta elementet Z på rad X
  member([X,Z], Prev),
  % Finns neg av Z på rad Y
  member([Y,neg(Z)], Prev),
  check_rule(Prems, Goal, Tproof, [[Linum,Res]|Prev]).

% contel (contradiction)
check_rule(Prems, Goal, [[Linum, Res, contel(X)]|Tproof], Prev) :-
  % Finns det en motsägelse på rad X
```

```
  member([X,cont], Prev),
  check_rule(Prems, Goal, Tproof, [[Linum,Res]|Prev]).

% negnegint
check_rule(Prems, Goal, [[Linum,neg(neg(Z)),negnegint(X)]|Tproof], Prev) :-
  % Finns det ett Z på rad X
  member([X,Z], Prev),
  check_rule(Prems, Goal, Tproof, [[Linum, neg(neg(Z))]|Prev]).

% negnegel
check_rule(Prems, Goal, [[Linum, Res, negnegel(X)]|Tproof], Prev) :-
  % Finns det negneg av Res på rad X
  member([X,neg(neg(Res))], Prev),
  check_rule(Prems, Goal, Tproof, [[Linum,Res]|Prev]).

% mt (modus tonem)
check_rule(Prems, Goal, [[Linum, neg(Z), mt(X,Y)]|Tproof], Prev) :-
  % Hitta Z -> W på rad X
  member([X,imp(Z, W)], Prev),
  % Finns neg W på rad Y
  member([Y,neg(W)], Prev),
  check_rule(Prems, Goal, Tproof, [[Linum,neg(Z)]|Prev]).

% lem
check_rule(Prems, Goal, [[Linum, or(Z,neg(Z)), lem]|Tproof], Prev) :-
  check_rule(Prems, Goal, Tproof, [[Linum, or(Z,neg(Z))]|Prev]).

% copy
check_rule(Prems, Goal, [[Linum, Res, copy(X)]|Tproof], Prev) :-
  % Finns Res på rad X
  member([X, Res], Prev),
  check_rule(Prems, Goal, Tproof, [[Linum,Res]|Prev]).

% ----------------- box handler ---------------------
% Hitta boxen genom att leta efter första raden, ett antagande.
check_rule(Prems, Goal, [[[Linum, Res, assumption]|Tbox]|Tproof], Prev) :-
  % Eftersom att vårat basfall alltid kollar om målet är uppfyllt hittar vi
det sista
  % elementet i boxen och använder det som mål.
  last([[Linum, Res, assumption]|Tbox], [_, BoxGoal, _]),
  % Vi skickar en boxen som ett nytt bevis i för att kontrollera alla rader,
  % första raden stoppas i Prev listan.
  check_rule(Prems, BoxGoal, Tbox, [[Linum,Res]|Prev]),
  % Sätter hela boxen i Prev listan som ett element
  check_rule(Prems, Goal, Tproof, [[[Linum, Res, assumption]|Tbox]|Prev]).
```

```
% ----------------- box rules ----------------------
% impint
check_rule(Prems, Goal, [[Linum, imp(Z,W), impint(X,Y)]|Tproof], Prev) :-
  % Hämta boxen som börjar på rad X från Prev
  extBox(X, Prev, Box),
  % Finns Z på första raden och är det ett antagande
  member([X, Z, assumption], Box),
  % Finns W på sista raden, rad Y
  member([Y, W, _],Box),
  check_rule(Prems, Goal, Tproof, [[Linum, imp(Z,W)]|Prev]).

% negint
check_rule(Prems, Goal, [[Linum, neg(Z), negint(X,Y)]|Tproof], Prev) :-
  % Hämta boxen som börjar på X från Prev
  extBox(X, Prev, Box),
  % Börjar boxen med Z på rad X
  member([X, Z, _], Box),
  % Slutar boxen med en motsägelse på rad Y
  member([Y, cont, _], Box),
  check_rule(Prems, Goal, Tproof, [[Linum, neg(Z)]|Prev]).

% pbc
check_rule(Prems, Goal, [[Linum, Res, pbc(X,Y)]|Tproof], Prev) :-
  % Hämta boxen som börjar på X från Prev
  extBox(X, Prev, Box),
  % Börjar boxen med neg Res på rad X, och är det ett antagande
  member([X, neg(Res), assumption], Box),
  % Slutar boxen på rad Y  med en motsägelse
  member([Y, cont, _], Box),
  check_rule(Prems, Goal, Tproof, [[Linum, Res]|Prev]).

% orel
% X1 raden för or
% X2-X3 Box1
% X4-X5 Box2
check_rule(Prems, Goal, [[Linum, Res, orel(X1,X2,X3,X4,X5)]|Tproof], Prev) :-
  % Finns det en or på rad X och vad innehåller den
  member([X1, or(Z,W)], Prev),
  % Hämta den första boxen
  extBox(X2, Prev, Box1),
  % Börjar första boxen med Z på rad X2
  member([X2, Z, assumption], Box1),
  % Slutar första boxen med Res  på rad X3
  member([X3, Res, _], Box1),
  % Samma för andra boxen
  extBox(X4, Prev, Box2),
  member([X4, W, assumption], Box2),
  member([X5, Res, _], Box2),
  check_rule(Prems, Goal, Tproof, [[Linum, Res]|Prev]).
```

```
% -------------------- box helper -------------------
% Söker efter en specifik box i Prev med hjälp av radnummer
% Anta att boxen är huvudet i Prev
extBox(Linum, [Box|_], Box) :-
  % Har huvudet i Prev ett element med rätt radnummer och ett antagande
  member([Linum,_,assumption], Box).

% Om elementet i Prev inte var boxen som söktes försöker vi igen
extBox(Linum, [_|Tbox], _) :-
  % Ta bor huvudet, gå till nästa element
  extBox(Linum, Tbox, _).
```