Laborationsrapport: Lab 2 Logik för Dataloger (DD1351)

Beviskontrollalgoritmen

Algoritmen för bevis kontroll består i sin helhet av fyra delar, inläsning, mål-kontrollering, regel-matchning och box-hantering.

Inläsning

För att läsa in beviset och variablerna används predikatet verify som vi tilldelats i laborationsinstruktionerna. Denna predikat läser in bevisets predikat (Prems), mål (Goal) och bevis (Proof) från en utomstående fil och kallar sedan predikaten kontrollera_mal och kontrollera_bevis.

Målkontroll

För att kontrollera att beviset leder till det målet som angivits används predikaten kontrollera_mal. Detta predikat använder hitta_sista och hitta_varde för att hitta innehållet på sista raden av beviset och jämför sedan det med Goal, det mål som lästs in tidigare i verify.

Naturliga Deduktions Regler

Delen av koden som ger utslag ifall någon av de naturliga deduktions reglerna används på ett felaktigt sätt kallas kontrollera_bevis. Kontrollera_bevis för med sig fyra variabler, Prems, Bevis, Scope och en icke-namngedd lista. I Prems finns de premisser som lästs in från verify, i Bevis finns hela beviset som lästs in i verify, i Scope lagras en lista av de rader som redan kollats och i den icke-namngivna listan sparas den delen av beviset som inte kontrollerats ännu. Kontrollera_bevis funkar på så sätt att det finns ett alternativ av predikatet för varje regel. De olika “versionerna” bemärks med att första raden i den icke-namngivna listan implementerar olika regler. Då en rad i beviset läses in i kontrollera_bevis matchas sedan den till en version av kontrollera_bevis med en lista (den icke-namngivna listan) som innehåller just den regeln som raden hävdar att den utför. Predikatet som tilldelas raden gör sedan en serie jämförelser (med hjälp av member) mellan radens innehåll och de rader som de refererar till (med undantag från assumption och premise som inte refererar till några tidigare kontrollerade rader). De tidigare kontrollerade raderna återfinns i Scope. Om alla villkor stämmer kallar kontrollera_bevis sig själv rekursivt. Men då den gör det ändras både Scope och den icke-namngivna listan. Raden som nu kontrollerats läggs in i Scope samtidigt som den tas ut ur den icke-namngivna listan, på så sätt kontrolleras nu en ny rad samtidigt som en godkänd rad sparas.

Box-hantering

Box-hanteringen sker till största del av aktiv_lada, hitta_sista, hitta_forsta och till viss del i kontrollera_bevis. Aktiv_lada går igenom beviset med hjälp av rekursion och söker efter en rad som matchar en lådas signatur. Hitta_sista och hitta_forsta är inte direkt kopplade till låd-predikatet men då de används för att hitta första respektive sista atomerna i en angiven lista så utnyttjas de ofta i samband med aktiv_lada. I kontrollera_bevis så har vi även ett predikat som tar hand om assumption. Detta görs genom att den delar upp beviset ytterligare i två listor. Den första av dem är resten av lådan, som kallas för Ladsvans, och den andra listan är resten av beviset efter lådan. Den tillkallar då först sig självt rekursivt och kör igenom resten av lådan och sen kommer den tillkalla sig självt rekursivt igen och köra igenom resten av beviset.

Tabell

Predikat	**När det är sant**	**När det är falskt**	**Användning**
verify	När filen som skickas in är i rätt filformat	Om filen inte är av rätt format	För att läsa indatan som sedan ska användas för att kontrollera beviset. Tillkallar även kontrollera_premis och kontrollera_mal.
hitta_sista	När den första parametern är en lista och den andra är ospecificerad	Om den första parametern inte är en lista	Den används för att hitta den sista raden i en låda.
hitta_varde	Den är sann när den första parametern är en lista och den andra är ospecificerad. I listan så måste det andra värdet vara specificerat.	Om den första parametern inte är en lista.	Den används för att hämta ett värdet från en lista (rad). Den kommer ligga på andra plats i raden.
hitta_forsta	När den första parametern är en lista och den andra är ospecificerad. Om den andra som skickas med är en lista så kommer den försöka att matcha den listan till den som hittades i lådan. Om det är en tom variabel så ges värdet av listan som hittas till den.	Om den första raden inte matchar den listan som skickas med så kommer den att vara falsk.	Den används för att hitta första raden antingen i hela beviset eller i en låda.
kontrollera_mal	När sista raden i beviset är samma som inlästa målet.	Om sista raden i beviset inte är detsamma som inlästa målet	Den kollar om bevisets slutsats stämmer överens med det som ska bevisas.
kontrollera_premis	När premisserna som är inlästa är detsamma som de som använts i beviset.	Om beviset använder sig av en premiss som inte finns.	Den kollar om beviset använder sig av korrekta predikat.
aktiv_lada	Om den låda som sökes hittas.	Om lådan som söks är tom eller om lådan som söks inte kan hittas i beviset.	Den letar efter en bestämd låda i beviset.
kontrollera_bevis	Då de naturliga deduktions reglerna följs.	Om någon regel används på fel sätt i beviset eller om beviset appliceras på fel sätt.	Den kollar om beviset applicerar naturlig deduktions reglerna på ett korrekt sätt.

Appendix

Programkoden

```
verify(InputFileName):-
        see(InputFileName),
        read(Prems), read(Mal), read(Bevis),
        seen,
        %writeln(Prems), writeln(Mal), writeln(Bevis), writeln(' '),
        kontrollera_mal(Bevis,Mal),
                        %kollar om bevisets sista rad är lika med målet som lästs in
        !, kontrollera_bevis(Prems, Bevis, Bevis, []), !.

hitta_sista([Huvud | []], Huvud).          %Basklausul för hitta_sista kommer att
användas när Huvud är sista elementet.
hitta_sista([_ | Svans], Huvud):-          %används när det finns element kvar i listan.
Ignorer då vad det finns för värde innan och går sen vidare rekursivt för att hitta sista
elementet.
        hitta_sista(Svans, Huvud).

hitta_varde([_, Huvud, _], Huvud).         %Används för att matcha ett värde och hämta
det

hitta_forsta([Huvud | _], Huvud).          %Hittar första saken i listan och sedan ger den
variabeln som skickas med det värdet eller kollar om den matchar det som blir medskickat i
den andra variabeln.


kontrollera_mal(Bevis, Mal):-
        hitta_sista(Bevis, Rad),                %writeln(Rad),%
        hitta_varde(Rad, Varde),                %writeln(Varde), writeln(Mal),
        Mal = Varde, !.                                 %jämför målet som lästs
med sista det som hämtats ur sista raden i beviset

kontrollera_premis(_, []):- !, fail.                    %basfall så att om
premissen som skickats med är tom failar operationen
kontrollera_premis(Varde, [Varde | _]).                         %
kontrollera_premis(Varde, [_ | Svans]) :-               %om en premis
%write('Varde: '), writeln(Varde), write('Svans: '), writeln(Svans),
!, kontrollera_premis(Varde, Svans).
```

```
aktiv_lada(_, [], _) :-

%Basfall, om lådan är tom vid något tillfälle så kommer funktionen att faila.
!, fail.

aktiv_lada([Nummer, Varde, _], [[[Nummer, Varde, _] | Ladsvans] | _], [[Nummer, Varde, _]
| Ladsvans]).          %Letar efter en del av listan som matchar en låda. Den tredje variablen
som skickas med kommer få det värdet.

aktiv_lada([Nummer, Varde, _], [_ | Svans], Lada):-
                                                    %Rekursivt åkallar sig
själv tills den matchar signaturen av en lada i den listan som skickas med.
!, aktiv_lada([Nummer, Varde, _], Svans, Lada).


kontrollera_bevis(_, _, [], _) :- !.
                                                    %Basklausul.
tillkallas när Svans är tom och vi är färdiga med beviset.

%kontrollerar att premiss är rätt
kontrollera_bevis(Prems, Bevis, [[Rad, Varde, premise] | Svans], Scope):-
        kontrollera_premis(Varde, Prems),
                                                    %kallar
kontrollera_premis
        %writeln('Premise'),
        %write('Varde: '), writeln(Varde), write('Prems: '), writeln(Prems),
        !, kontrollera_bevis(Prems, Bevis, Svans, [[Rad, Varde, premise] | Scope]).
                                                    %kallar kontrollera bevis rekursivt


%Kollar ett antagande
kontrollera_bevis(Prems, Bevis, [[[Rad, Varde, assumption] | Ladsvans] | Svans], Scope):-

        %writeln('assumption'),
        kontrollera_bevis(Prems, Bevis, Ladsvans, [[Rad, Varde, assumption] | Scope]),
                                                    %kallar kontrollera bevis rekursivt för att kolla
lådans innehåll
        kontrollera_bevis(Prems, Bevis, Svans, [[[Rad, Varde, assumption] | Ladsvans] |
Scope]).                                %fortsätter med resten av beviset

%Kopierar
kontrollera_bevis(Prems, Bevis, [[Rad, Varde, copy(X)] | Svans], Scope):-
```

```
        member([X, Gammalr, _], Scope),

%identifierar den rad som ska kopieras
        %write('copy'),

        Gammalr = Varde, !,

        %jämför innehållet i raden som kopierat och raden som utfört kopierandet
        !, kontrollera_bevis(Prems, Bevis, Svans, [[Rad, Varde, copy(X)] | Scope]).
                                                %kallar kontrollera bevis rekursivt

%introducerar ett and element
kontrollera_bevis(Prems, Bevis, [[Rad, and(A,B), andint(X,Y)]| Svans], Scope):-
        %writeln('andint'),
        member([X, A, _], Scope),

%kollar att första termen som and ska införas mellan finns på utsedda raden
        member([Y, B, _], Scope),

%kollar att andra termen som and ska införas mellan finns på utsedda raden
        !, kontrollera_bevis(Prems, Bevis, Svans, [[Rad, and(A,B), andint(X,Y)] | Scope]).
                                        %kallar kontrollera bevis rekursivt

%and el 1
kontrollera_bevis(Prems, Bevis, [[Rad, Varde, andel1(X)] | Svans], Scope):-
        %writeln('andel1'),
        member([X, and(Varde, _a), _], Scope),
                                                                %Letar
efter raden X och kontrollerar att den innehåller ett and vars första plats ockuperas
        !, kontrollera_bevis(Prems, Bevis, Svans, [[Rad, Varde, andel1(X)] | Scope]).
                                        %av Varde och att det finns ett värde på den
andra platsen men det spelar ingen roll vad det värdet är.


%and el 1
kontrollera_bevis(Prems, Bevis, [[Rad, Varde, andel2(X)] | Svans], Scope):-
        %writeln('andel2'),
        member([X, and(_a, Varde), _], Scope),
                                                                %Letar
efter raden X och kontrollerar att den innehåller ett and vars andra plats ockuperas
        !, kontrollera_bevis(Prems, Bevis, Svans, [[Rad, Varde, andel2(X)] | Scope]).
                                        %kallar kontrollera bevis rekursivt

%or int
```

```
kontrollera_bevis(Prems, Bevis, [[Rad, or(A,B), orint1(X)] | Svans], Scope):-
        %writeln('orint1'),
        member([X, A, _], Scope),

%Kollar att första termen i radens orsats finns på raden som refereras till
        !, kontrollera_bevis(Prems, Bevis, Svans, [[Rad, or(A,B), orint1(X)] | Scope]).
                                                %kallar kontrollera bevis rekursivt

%or int 2
kontrollera_bevis(Prems, Bevis, [[Rad, or(A,B), orint2(X)] | Svans], Scope):-
        %writeln('orint2'),
        member([X, B, _], Scope),

%Kollar att andra termen i radens orsats finns på raden som refereras till
        !, kontrollera_bevis(Prems, Bevis, Svans, [[Rad, or(A,B), orint2(X)] | Scope]).
                                                %kallar kontrollera bevis rekursivt

%or el
kontrollera_bevis(Prems, Bevis, [[Rad, Varde, orel(X,Y,U,V,W)] | Svans], Scope):-
   %writeln('orel'),
   member([X, or(A,B), _], Scope),
                                                                        %namnger
termerna som oras till A och B samt kollar att det finns en or att eliminera
   aktiv_lada([Y, A, _], Scope, Lada1),
                                                                %letar efter en
låda som börjar med A på rad Y
   aktiv_lada([V, B, _], Scope, Lada2),
                                                                %letar efter en
låda som börjar med B på rad V
   hitta_forsta(Lada1, [Y, A, _]),
                                                                        %kollar att
lådan  börjar med A på rätt rad
   hitta_forsta(Lada2, [V, B, _]),
                                                                        %kollar att
lådan  börjar med B på rätt rad
   hitta_sista(Lada1, [U, Varde, _]),
                                                                %kollar om sista
raden båda lådorna är dessamma
   hitta_sista(Lada2, [W, Varde, _]),

   !, kontrollera_bevis(Prems, Bevis, Svans, [[Rad, Varde, orel(X,Y,U,V,W)] | Scope]).
                                        %kallar kontrollera bevis rekursivt

%impint
```

```
kontrollera_bevis(Prems, Bevis, [[Rad, Varde, impint(X,Y)] | Svans], Scope):-
   %writeln('impint'),
   aktiv_lada([X, A, _], Scope, Box),
                                                                %letar efter en
låda som börjar på rad X
   hitta_forsta(Box, [X, A, _]),
                                                                    %letar
efter första raden i lådan
   hitta_sista(Box, [Y, B, _]),
                                                                %letar efter sista
raden i lådan
   !, kontrollera_bevis(Prems, Bevis, Svans, [[Rad, Varde, impint(X,Y)] | Scope]).
                                        %kallar kontrollera bevis rekursivt

%impel
kontrollera_bevis(Prems, Bevis, [[Rad, Varde, impel(X,Y)] | Svans], Scope):-

        %writeln('impel'),
        member([X, X2, _], Scope),

%namnger innehållet på första raden
        member([Y, imp(X2, Varde), _], Scope), !,
                                                                %kollar att rad Y
innehåller en implikation mellan innehållet i rad X
        !, kontrollera_bevis(Prems, Bevis, Svans, [[Rad, Varde, impel(X,Y)] | Scope]).
                                            %kallar kontrollera bevis rekursivt

%negint
kontrollera_bevis(Prems, Bevis, [[Rad, neg(A), negint(X,Y)]| Svans], Scope):-
        %writeln('negint'),
        aktiv_lada([X, A, _], Scope, Lada),
                                                                %Letar efter en
låda som matchar signaturen i Scope. Lada kommer att få det som värde.
        hitta_forsta(Lada, [X, A, _]),
                                                                    %kollar att
första raden i lådan är rad X som innehåller A
        hitta_sista(Lada, [Y, cont, _]),
                                                                    %kollar att
sista raden i lådan är Y
        !, kontrollera_bevis(Prems, Bevis, Svans, [[Rad, neg(A), negint(X,Y)] | Scope]).
                                        %kallar kontrollera bevis rekursivt

%negel
```

```
kontrollera_bevis(Prems, Bevis, [[Rad, cont, negel(X,Y)] | Svans], Scope):-

        %writeln('negel'),
        member([X, X2, _],Scope),

%kollar att raden X fins och namnger innehållet
        member([Y, neg(X2), _], Scope),

%kollar att rad Y finns och innehåller en negation av innehållet i X
        !, kontrollera_bevis(Prems, Bevis, Svans, [[Rad, cont, negel(X,Y)] | Scope]).
                                                %kallar kontrollera bevis rekursivt

%contel
kontrollera_bevis(Prems, Bevis, [[Rad, Varde, contel(X)]| Svans], Scope):-

        %writeln('contel'),
        member([X, cont, _], Scope),
                                                                        %kollar att
raden som refereras till innehåller cont
        !, kontrollera_bevis(Prems, Bevis, Svans, [[Rad, Varde, contel(X)] | Scope]).
                                                %kallar kontrollera bevis rekursivt

%negnegint
kontrollera_bevis(Prems, Bevis, [[Rad, Varde, negnegint(X)] | Svans], Scope):-
        %writeln('negnegint'),
        member([X, Copy, _], Scope),

%kollar att rad X finns och namnger innehållet
        neg(neg(Copy)) = Varde,

        %kollar att en dubbelnegation av rad X inehåll är desamma som radens innehåll
        !, kontrollera_bevis(Prems, Bevis, Svans, [[Rad, Varde, negnegint(X)] | Scope]).
                                        %kallar kontrollera bevis rekursivt

%negnegel
kontrollera_bevis(Prems, Bevis, [[Rad, Varde, negnegel(X)] | Svans], Scope):-
        %writeln('negnegel'),
        member([X, neg(neg(Varde)), _], Scope),
                                                                        %kollar att
X inehåller en dubbel negation med värdet i
        !, kontrollera_bevis(Prems, Bevis, Svans, [[Rad, Varde, negnegel(X)] | Scope]).
                                                %kallar kontrollera bevis rekursivt

%mt
```

```
kontrollera_bevis(Prems, Bevis, [[Rad, neg(Varde), mt(X,Y)] | Svans], Scope):-
   %writeln('mt'),
   member([X, imp(Varde,B), _], Scope),
                                                                %kollar att X
inehåller en implikation melln radens värde och något som namnges som B
   member([Y, neg(B), _], Scope),
                                                                    %kollar
attrad Y innehåller en negation av B
   kontrollera_bevis(Prems, Bevis, Svans, [[Rad, neg(Varde), mt(X,Y)] | Scope]).
                                        %kallar kontrollera bevis rekursivt

%pbc
kontrollera_bevis(Prems, Bevis, [[Rad, Varde, pbc(X,Y)] | Svans], Scope):-
   %writeln('pbc'),
   aktiv_lada([X, neg(Varde), _], Scope, Lada),
                                                        %letar efter en låda som
börjar på rad X och som innehåller en negation av värdet i raden
   hitta_forsta(Lada, [X, neg(Varde), _]),
                                                                %Kollar att första
raden i lådan stämmer med X och negation
   hitta_sista(Lada, [Y, cont, _]),
                                                                %kollar att sista
raden innehåller en cont
   !, kontrollera_bevis(Prems, Bevis, Svans, [[Rad, Varde, pbc(X,Y)] | Scope]).
                                        %kallar kontrollera bevis rekursivt

%lem
kontrollera_bevis(Prems, Bevis, [[Rad, or(A,B), lem] | Svans], Scope):-
        %writeln('lem'),
        A = neg(B) ; B = neg(A), !,

%kollar att A är en negation av B och B är en negation av A
        !, kontrollera_bevis(Prems, Bevis, Svans, [[Rad, or(A,B), lem] | Scope]).
                                            %kallar kontrollera bevis rekursivt
```

Exempelbevisen

Sant	Falskt