Logik för dataloger (DD1351 HT22)
Laboration 2: Beviskontroll

Hibatallah Belhajali

December 2022

1 Inledning:

Syftet med detta labb är att skapa ett program som ger olika bevis skrivet med naturlig deduktion bör verifiera om detta bevis är giltigt. Beviset kommer förses i sin helhet med "sekventen" och alla linjer som kompromissar beviset. För att kontrollera om beviset verkligen är giltigt måste man kontrollera att alla regler som finns inom ramen för naturligt deduktion följs. Om reglerna har följts genom hela beviset och det finns inga avvikelser från detta då kan beviset klassas som korrekt. Men om beviset gör en ogiltigt antagande eller använder en regel på fel sätt eller ange något som är inte logiskt korrekt beviset bör anses vara felaktigt. Algoritmen och programmet kommer att skrivas med hjälp av programmeringsspråket Prolog.

2 Metod för programbildning:

Det första steget är att kontrollera mål för "sekvent" och jämför det med sista raden, detta görs eftersom om beviset slutar inte med målet kan man direkt säga att beviset är felaktigt och man kan avsluta det där. Detta är viktigt att ha för att unvika slösa tiden på att söka igenom hela beviset utan anledning eftersom du redan vet att slutresultatet av beviset inte är vad det ska vara. Efter detta man kan göra själva bevis- och regelkontrollen
Beviset representeras som en lista så man kommer att korsa listlinjen för linje och validera att linjen är acceptabel. När vi har markerat en rad kommer vi att lägga den i en annan lista som kommer att innehålla den del av beviset vi redan har kontrollerat och ansett som legitim. Ett kommer att ha gått igenom korrekturet när provlistan är tom och den kan sedan avsluta beviskontrollen. Men programmet måste känna till regler den ska följa för att kunna avgöra vad som är rätt och vad som är fel.
Detta innebär att vi måste definiera olika regler som används när vi gör en naturlig deduktions bevis. Genom att använda predikat kommer man att representera de saker som är tillåtna när vi använder en viss regel och om denna inte följs kan den avsluta beviskontrollen.

3 Hantering av Boxar:

I naturligt deduktion när vi använder vissa regler behöver vi skapa en ruta(Box), för till exempel när vi har ett antagande måste vi öppna en box och sedan stänga den när vi har slutat antagandet. Dessutom måste man använda boxar till exempel för reglerna OR eliminering, imp introduktion och PBC. Eftersom korrektur kan ha lådor måste programmet på något sätt ta hand om dessa och det finns olika sätt att göra detta. Ett sätt som genomfördes var att leta efter underlistor i själva beviset.Sedan representationen av bevis skapas på ett sätt att när en box startas skapades en ny lista inuti det allmänna beviset. Så för att veta när en ruta nås måste vi kontrollera om det finns en underlista eller inte och när vi har den kan vi kontrollera den sista och första raderna i rutan och när vi har en regel som behöver en box använder vi den och kontrollerar att regeln följs.

4 Implementation:

För att implementera beviskontrollen måste man följa metoden för programbildning för att sedan översätta detta till faktisk Prolog-kod för att skapa program. Detta kommer att vara predikat. Först behöver vi predikatet som kommer att kontrollera "sekvent" och den sista raden i koden för att säkerställa att det första steget är uppfyllt och om inte kan beviskontrollen avslutas vilket görs med en målkontroll predikat. Nästa steg är att göra regelkontrollen som nämns i metoden och det betyder att vi måste definiera varje regel. Detta gjordes med alla olika bevisvalidering för att definiera reglerna och hur de ska vara användbara. De specifika predikaten kommer att visas i en tabell nedan. Då hade vi också att ta hand om boxar och detta gjordes med att kontrollera att om en box hittas då kommer vi att ha en underlista i beviset.

5 Predikat tabell:

goal-check	Kontrollera om målet är lika med den sista raden i beviset då är det korrekt
rule-check	Gå genom hela beviset och kontrollera om lagar är uppfylld på rätt sätt för att returnera True
valid-proof	Om rule-check och goal-check är korrekta då returneras True
box-check	korrekt om det finns en subblista i beviset
rule-validity	Kontrollera olika "Rules" enligt nedan
Premis	Kontrollera om premis vi har i beviset är korrekt
Antagande	True när vi har en antagande (Assumption)
And int	När Två seperata variabel finns med i beviset och både är "true"
And el1	Kontrollera om vi har en and formel i beviset och ta bort den första variabel från det
And el2	Kontrollera om vi har en and formel i beviset och ta bort den andra variabel från det
Or int1	Addera OR en frivilligt variabel om den första variabel finns med i beviset och det är "true"
Or int2	Addera OR en frivilligt variabel om den andra variabel finns med i beviset och det är "true"
Or el	Om vi har en OR formula i beviset, Assumera den första variabel och få en resultat sen assumera den andra variabel, om vi får samma resultat då kan man ta bort OR
imp el	Om vi har en impikation i beviset och vi vet att den första variabel är "true" den blir också True
imp int	Om vi öppnar en box med antagande den första variabel och sen få den andra som resultat så vi kan lägga till implikations tecken
neg el	om vi har en variabel och sin negation både i beviset
neg int	Kan eliminera negation om vi har en contradiction
neg neg el	Är korrekt om vi har negation av negation i beviset
Contradiction el	om vi har en contradiction i beviset vi kan introduce vilken variabel som helst
MT	Om vi har en implikation och negation av den första variabel då kan vi nå negation av den första variabel också
PBC	Om vi öppnar en box men en negerad variabel och sen få e contradiction då kan det ge oss en variabel som är inte negerad
Lem	den är korrekt om vi når en variabel or sin negation

6 Appendix:

exempel av bevis:

1	$(p \vee q) \vee r$	premise
2	$(p \vee q)$	assumption
3	p	assumption
4	$p \vee (q \vee r)$	$\vee \mathrm{i}_1$ 3
5	q	assumption
6	$q \vee r$	$\vee \mathrm{i}_1$ 5
7	$p \vee (q \vee r)$	$\vee \mathrm{i}_2$ 6
8	$p \vee (q \vee r)$	$\vee$e 2, 3−4, 5−7
9	r	assumption
10	$q \vee r$	$\vee \mathrm{i}_2$ 9
11	$p \vee (q \vee r)$	$\vee \mathrm{i}_2$ 10
12	$p \vee (q \vee r)$	$\vee$e 1, 2−8, 9−11

Korrekt bevis

1	$(p \vee q) \vee r$	premise
2	$(p \vee q)$	assumption
3	p	assumption
4	$p \vee (q \vee r)$	$\vee \mathrm{i}_1$ 3
5	q	assumption
6	$q \vee r$	$\vee \mathrm{i}_1$ 5
7	$p \vee (q \vee r)$	$\vee \mathrm{i}_2$ 6
8	r	assumption
9	$q \vee r$	$\vee \mathrm{i}_2$ 9
10	$p \vee (q \vee r)$	$\vee \mathrm{i}_2$ 10

Fel bevis
Source tabell: Book"LOGIC IN COMPUTER SCIENCE" Michael Huth, ISBN 0 521 54310X S.19

Kod av Bevis kontroller:

```
:- discontiguous proof_validation/3.

verify(InputFileName) :- see(InputFileName),
                         read(Prems), read(Goal), read(Proof),
                         seen,
                         valid_proof(Prems, Goal, Proof).

% Kontrollera om målet och sista raden är samma, om målet och sista raden är
%samma men dess antagande då failas den
goal_check(Proof,Goal):-last(Proof,[_,Goal,assumption]),!,fail.
goal_check(Proof,Goal):-last(Proof,[_,Goal,_]).


% kontrollera om beviset är giltigt, om vi har tom lista så är vi klara
rule_check(_,[],_).

rule_check(Prems,[H|T],checked_list):-
rule_validity(Prems,H,checked_list),
append(checked_list,[H],new_list),
rule_check(Prems,T,new_list).

%kontrollera om Goal samt rule både är rätt
valid_proof(Prems,Goal,Proof) :- goal_check(Proof,Goal),
rule_check(Prems,Proof,[]), !.

% premise
rule_validity(Prems, [_,Atom,premise], _):-member(Atom,Prems).

% antagande
rule_validity(Prems,[[_,_,assumption]|T],checked_list):-
append(checked_list,[[_,_,assumption]],new_list),rule_check(Prems,T,new_list).

%kontrollera att vi har en underlista i beviset och första raden och
%sista raden i Box

box_check(First, Last, checked_list):- member([First|T],checked_list),last(T,Last).

box_check(A,A,checked_list):-member([A],checked_list).

%and int (andint(X,Y))
rule_validity(_,[_,and(A,H),andint(X,Y)], checked_list):-
member([X,A,_],checked_list), member([Y,H,_],checked_list).
```

```
%and el1 (andel1(X))
rule_validity(_,[_,A,andel1(X)],checked_list):-
member([X,and(A,_),_],checked_list).

%and el2 (andel2(X))
rule_validity(_,[_,H,andel2(X)],checked_list):-
member([X,and(_,H),_],checked_list).

%or int1 (orint1(X))
rule_validity(_,[_,or(A,_),orint1(X)], checked_list):-
member([X,A,_],checked_list).

%or int2 (orint2(X))
rule_validity(_,[_,or(_,H), orint2(X)], checked_list):-
member([X,H,_],checked_list).

%or el (orel(X,Y,U,V,W))
rule_validity(_,[_,A,orel(X,Y,U,V,W)],checked_list):-
member([X, or(B,H),_],checked_list),
box_check([Y,B,assumption],[U,A,_],checked_list),
box_check([V,H,assumption],[W,A,_],checked_list).

%implikations el (impel(X,Y))
rule_validity(_,[_,B,impel(X,Y)],checked_list):-
member([X,A,_],checked_list),
member([Y, imp(A,B),_],checked_list).

%implikations int (impint(X,Y))
rule_validity(_,[_,imp(A,H),impint(X,Y)],checked_list):-
box_check([X,A,assumption],[Y,H,_],checked_list).

%negations el (negel(X,Y))
rule_validity(_,[_,cont,negel(X,Y)],checked_list):-
member([X,A,_],checked_list),member([Y,neg(A),_],checked_list).

% negations int (negint(X,Y))
rule_validity(_,[_,neg(A),negint(X,Y)],checked_list):-
box_check([X,A,assumption],[Y,cont,_],checked_list).

% negationsnegations int (negnegint(X))
rule_validity(_,[_,neg(neg(Atom)),negnegint(X)],checked_list):-
member([X,Atom,_],checked_list).

% negations negations el (negnegel(X))
rule_validity(_,[_,Atom,negnegel(X)],checked_list):-
member([X,neg(neg(Atom)),_],checked_list).
```

```
% MT (mt(X,Y))
rule_validity(_,[_,_,mt(X,Y)],checked_list):-
member([X,imp(_,B),_],checked_list),member([Y,neg(B),_],checked_list).

%PBC (pbc(X,Y))
rule_validity(_,[_,A,pbc(X,Y)],checked_list):-
box_check([X, neg(A),assumption],[Y,cont,_],checked_list).

% LEM (lem)
rule_validity(_,[_,or(A,neg(A)),lem],_).

% Kopiera (copy(X))
rule_validity(_,[_,A,copy(X)],checked_list):-member([X,A,_],checked_list).

%  contraduction (contel(X))
rule_validity(_,[_,_,contel(X)],checked_list):-
member([X,cont,_],checked_list).
```