Daniel Landberg, CDATE17

DD1351 Logik för dataloger
Laboration 2: beviskoll

Daniel Landberg, CDATE17

Problemspecifikation

Syftet med laborationen var att konstruera ett beviskontroll program som kontrollerar om bevisen är korrekta för Predikatlogik givet regler som finns i Appendix A.
Indata är i form av rader med olika applicerade regler på det som ska bevisas. Programmet kommer svara True om filen innehåller ett korrekt bevis och False om beviset inte är korrekt.

Uppbyggnad

Programmet är uppbyggt kring 4 st predikat;
- `verify(InputFileName)`
- `validate(Prems, Goal, Proof)`
- `check_goal(Proof, Goal)`
- `check_proof(Prems, [H|T], Prooved)`

Den första (verify) är den som anropas och tar och läser in en fil till programmet (det som ska kontrolleras). Validate tar sedan allt som verify läste in och skickar till check_proof. Den kommer sedan att skickas vidare rad för rad tills den träffar rätt regel. När den träffat rätt regel så kommer den att lägga till den lästa regeln i prooved och försätta att leta efter en ny regel. När alla regler lästs så kommer den träffa sitt bas fall.
Efter basfallet körts för Check_proof kommer Validate fortsätta att köra och då köra check_goal. Den kommer kolla att sista objektet på listan Proof matchar formen för argument 2.

Problem och reflektioner

Det finns antagligen ett antal olika tillvägagångssätt för att verifiera att beviset är korrekt. Vi valde att bygga koden runt reglerna och att den testar alla varje gång.
Något vi märkte i slutet var att istället för att låta Check_proof träffa varje regel så kunde vi ha kodat så att den endast blev träffad när den regeln skulle appliceras. Men det kom vi på efter och därför ändrade vi inget. Det som vi tyckte va svårast var att lösa hur vi skulle behandla antaganden (boxar) och hur vi skulle implementera det i koden.

Daniel Landberg, CDATE17

Lösningens begränsningar

Vid körning av egna testfall så fick vi väntat resultat. Vid körning av alla tester failar test invalid19 och valid10/11. Anledningen till att test invalid19 ger false är att vi inte håller koll på om copy försöker kopiera något som står i en box. Valid10/11 har vi implementerat vår assumtion så att vi bara kan hantera boxar som börjar med ett assumtion.

Predikat

`verify(InputFileName):-`	Det predikat som anropas. Plockar först ut Prems, Goal, Proof från filen det fått som argument. Anropar validate.
`validate(Prems, Goal, Proof):-`	Kollar om vår indata är valid genom att kalla check_proof & check_goal
`check_Proof(_, [ ], _):-`	Basfall som kollar om vi har slut på argument i listan.
`check_Proof(Prems, [H|T], Prooved):-`	Kollar så att våra steg i beviset är giltiga. Kommer vara den som tar varje regel och testar den. Anroppar rekursivt
`check_goal(Proof, Goal):-`	Kolla att sista objektet på listan Proof matchar formen för argument 2
`check_proof(Prems, T, [H	Prooved]:-`	Anropa check_proof rekursivt, med Tail:n för bevislistan och lägg till Head:n i listan med Prooved
`check_proof(Prems, Box, [Assume|Prooved]):-`	Anroppar rekursivt med det som vi har i vår ”box” och med Assume inlagt i Prooved
`check_proof(Prems, T, NewProoved):-`	Forsätter med samma rekusiva anropp efter vi lämnat boxen.
`check_proof(Prems, T, Prooved):-`	Kallelse från Law of excluded middle

The basic rules of natural deduction:

	introduction	*elimination*
$\wedge$	$\dfrac{\phi \quad \psi}{\phi \wedge \psi}\ \wedge\mathrm{i}$	$\dfrac{\phi \wedge \psi}{\phi}\ \wedge\mathrm{e}_1 \qquad \dfrac{\phi \wedge \psi}{\psi}\ \wedge\mathrm{e}_2$
$\vee$	$\dfrac{\phi}{\phi \vee \psi}\ \vee\mathrm{i}_1 \qquad \dfrac{\psi}{\phi \vee \psi}\ \vee\mathrm{i}_2$	$\dfrac{\phi \vee \psi \quad \boxed{\begin{matrix}\phi \\ \vdots \\ \chi\end{matrix}} \quad \boxed{\begin{matrix}\psi \\ \vdots \\ \chi\end{matrix}}}{\chi}\ \vee\mathrm{e}$
$\rightarrow$	$\dfrac{\boxed{\begin{matrix}\phi \\ \vdots \\ \psi\end{matrix}}}{\phi \rightarrow \psi}\ \rightarrow\mathrm{i}$	$\dfrac{\phi \quad \phi \rightarrow \psi}{\psi}\ \rightarrow\mathrm{e}$
$\neg$	$\dfrac{\boxed{\begin{matrix}\phi \\ \vdots \\ \bot\end{matrix}}}{\neg\phi}\ \neg\mathrm{i}$	$\dfrac{\phi \quad \neg\phi}{\bot}\ \neg\mathrm{e}$
$\bot$	(no introduction rule for $\bot$)	$\dfrac{\bot}{\phi}\ \bot\mathrm{e}$
$\neg\neg$		$\dfrac{\neg\neg\phi}{\phi}\ \neg\neg\mathrm{e}$

Some useful derived rules:

$$\dfrac{\phi \rightarrow \psi \quad \neg\psi}{\neg\phi}\ \mathrm{MT} \qquad \dfrac{\phi}{\neg\neg\phi}\ \neg\neg\mathrm{i}$$

$$\dfrac{\boxed{\begin{matrix}\neg\phi \\ \vdots \\ \bot\end{matrix}}}{\phi}\ \mathrm{PBC} \qquad \dfrac{}{\phi \vee \neg\phi}\ \mathrm{LEM}$$

Figure 1.2. Natural deduction rules for propositional logic.

Bevissystem

Daniel Landberg, CDATE17

Källkod

```
verify(InputFileName) :-
		see(InputFileName),
		read(Prems), read(Goal), read(Proof),
		seen,
		validate(Prems, Goal, Proof). %Validera beviset

validate(Prems, Goal , Proof) :-
		check_proof(Prems, Proof, []),
		check_goal(Proof, Goal).

check_goal(Proof, Goal) :-
		last(Proof, [_, Goal, _]).%kolla att sista objektet på listan Proof matchar formen
för argument 2

check_proof(_, [], _). %Basfallet för beviskontrollering. Om vi får in en tom lista för Proof
terminerar programmet

%Kolla att premisserna är giltiga. ok
check_proof(Prems, [H | T], Prooved) :-
		H = [_, Prem, premise],
		member(Prem, Prems),
		check_proof(Prems, T, [H | Prooved]).
		%Anropa check_proof rekursivt,  med Tail:n för bevislistan och lägg till Head:n i
listan med Prooved

%AND introduction
check_proof(Prems, [H|T],  Prooved) :-
		H = [R, and(P, Q), andint(L1,L2)],
		%;Matcha denna rad i beviset (H) med and(P,Q) i mitten och andint(L1,L2) till
höger
		% Beviset på denna rad säger att vi använder och-introduktion på rad L1 och
L2 (båda variabler)
		% Och därför kan vi skriva P och Q

		R>L1, R>L2, %Beviset i H ligger på rad med index R. L1 och L2 äor de
raderna vi använder och-introduktion på. Dessa måste vara mindre än R för att beviset ska
vara gitligt
		member([L1, P, _], Prooved), %Kolla att vi har bevisat att vi har P på rad L1
		member([L2, Q, _], Prooved), %Kolla att vi har bevisat att vi har Q på rad L2
		check_proof(Prems, T, [H | Prooved]).%När vi är klara med kontroll av AndInt
lägger vi till beviset H i listan med alla kontrollerade bevis. Notera att positionen för H i
Prooved inte spelar någon roll
```

Daniel Landberg, CDATE17

```
%AND Elimination 1 kollat
check_proof(Prems, [H|T], Prooved) :-
		H = [R, P , andel1(L)],
		R>L,
		member([L, and(P, _), _], Prooved), %Kolla att [L, P och _, _] finns med i
bevisade isf lägger vi till H till Prooved.
		check_proof(Prems, T, [H | Prooved]).

%AND Elimination 2 kollat
check_proof(Prems, [H|T], Prooved) :- %exakt som ovan fast med and(_,P)
		H = [R, P , andel2(L)],
		R>L,
		member([L, and(_, P), _], Prooved),
		check_proof(Prems, T, [H | Prooved]).

%OR introduction 1
check_proof(Prems, [H|T], Prooved) :-
		H = [R, or(P,_), orint1(L)],
		R>L,
		member([L, P, _], Prooved),
		check_proof(Prems, T,[H | Prooved]).

%OR introduction 2
check_proof(Prems, [H|T], Prooved) :-
		H = [R, or(_,P), orint2(L)],
		R>L,
		member([L, P, _], Prooved),
		check_proof(Prems, T, [H | Prooved]).

%OR elemination %test19
check_proof(Prems, [H|T], Prooved) :-
		H = [R, X, orel(L1,L2,L3,L4,L5)],
		R>L1,R>L2,R>L3,R>L4,R>L5,
		member([L1, or(P,Q),_], Prooved), %kollar att premisen ligger i Prooved
		member([L2, P, assumption], Prooved), %kollar att vårt första antagande ligger
i Prooved
		member([L3, X ,_], Prooved), %kollar att det vi söker (vill ha) ligger i Prooved
		member([L4, Q, assumption], Prooved), %kollar att vårt andra antagande ligger
i Prooved
		member([L5, X,_], Prooved),%kollar att det vi söker (vill ha) ligger i Prooved
		check_proof(Prems, T, [H | Prooved]). %lääger in i Prooved

%IIMPLICATION introduction
check_proof(Prems, [H|T], Prooved) :-
		H = [R, imp(P,Q), impint(L1,L2)],
		R>L1, R>L2,
		member([L1,P,assumption], Prooved),
```

```
                member([L2, Q, _], Prooved),
                check_proof(Prems, T, [H|Prooved]).

%IIMPLICATION elimination
check_proof(Prems, [H| T], Prooved) :-
                H = [R, Q, impel(L1, L2)],
                R>L1, R>L2,
                member([L1, P ,_], Prooved), %Kolla att HL i imp() är True
                member([L2, imp(P,Q) ,_], Prooved),%Kolla att P implicerar Q på någon rad i
beviset
                check_proof(Prems, T,[H | Prooved]).

%NEGATION introduction
check_proof(Prems, [H| T], Prooved) :-
                H = [R, neg(P), negint(L1, L2)], %rad, neg p, vilka rader antangandet sker
                R>L1, R>L2,
                member([L1, P , assumption], Prooved), %kollar ås första raden är assumtion.
                member([L2, cont, _], Prooved), %kollar att sista raden är motsägelse.
                check_proof(Prems, T,[H | Prooved]). %skickar vidare.

%NEGATION elemination
check_proof(Prems, [H|T], Prooved) :-
                H = [R, cont, negel(L1,L2)],
                R>L1, R>L2,
                member([L1, P, _], Prooved),
                member([L2, neg(P),_], Prooved),
                check_proof(Prems, T,[H | Prooved]).

%DOUBLE_NEGATION introduction
check_proof(Prems, [H| T], Prooved) :-
                H = [R, neg(neg(P)), negnegint(L)],
                R>L,
                member([L, P,_], Prooved),
                check_proof(Prems, T, [H|Prooved]).

%DOUBLE_NEGATION elemination
check_proof(Prems, [H| T], Prooved) :-
                H = [R, P, negnegel(L)],
                R>L,
                member([L, neg(neg(P)),_], Prooved),
                check_proof(Prems, T,[H | Prooved]).

%CONTRADICTION elimination
check_proof(Prems, [H|T], Prooved) :-
                H = [R, _, contel(L)],
                R>L,
                member([L,cont,_], Prooved),
                check_proof(Prems, T,[H | Prooved]).
```

Daniel Landberg, CDATE17

```
%ASSUMPTION
check_proof(Prems , [H|T], Prooved) :-
		H = [Assume|Box], %assume = första reden i boxen, dvs raden som har
Assumption i sig. Box består av resterande rader i boxen
		%Vi kollar om H i sig består av flera listor. Vi tar Head:n av H till Assume och
Tail:n av H till Box
		Assume = [_, _, assumption],%Vi kontrollerar att Assume (Första raden i
boxen) matchar formen
		check_proof(Prems, Box, [Assume|Prooved]),
		last(Box,BoxGoal),%Ta ut sista elementet av boxen, och döp den till BoxGoal
		append([Assume|[BoxGoal]], Prooved, NewProoved),%Nu lägger vi till Assume
och och BoxGoal i listan Prooved och skapar en ny lista
		check_proof(Prems, T, NewProoved).

		%Check that all the proofs in the box are valid within



%COPY
check_proof(Prems, [H|T], Prooved) :-
		H = [R, P, copy(L)],
		R > L,
		member([L, P, _], Prooved),
		check_proof(Prems, T,[H | Prooved]).

%MODUS_TOLLENS
check_proof(Prems, [H| T], Prooved) :-
		H = [R, neg(P), mt(L1,L2)],
		R>L1, R>L2,
		member([L1, imp(P, Q), _], Prooved),
		member([L2, neg(Q), _], Prooved),
		check_proof(Prems, T,[H | Prooved]).

%PROOF_BY_CONTRADICTION
check_proof(Prems, [H|T], Prooved) :-
		H = [R, P, pbc(L1,L2)],
		R>L1, R>L2,
		member([L1, neg(P), assumption], Prooved),
		member([L2, cont, _] , Prooved),
		check_proof(Prems, T,[H | Prooved]).

%LAW_OF_EXCLUDED_MIDDLE
check_proof(Prems, [[_, or(P, neg(P)), lem]| T], Prooved) :-
		 check_proof(Prems, T, Prooved).
```
