Prolog Labb 2
Jacob Hedén Malm
199804051499
CINTE åk2

Programmets Utformelse

Programmets startpunkt är i verify predikatet. Verify accepterar en indatafil bestående av tre prolog-termer. Den bevarar sedan den här indatan i tre variabler, Prems, Goal, och Proof. Dessa tre variabler skickas därefter vidare till nästa predikat, valid_proof. Valid_proof gör endast två saker, skickar Goal och Proof till predikatet check_goal, och skickar Proof tillsammans med en tom lista till check_proof. Check_goal skickar Proof vidare till predikatet findLastProof, som itererar igenom listan fram till sista raden av bevis. Den unifierar därefter det andra argumentet, X, med mittentermen i sista raden. Sedan kollar check_goal om X är samma som Goal, vilket det borde vara om beviset härleddes till målet.

Check_proof accepterar som sagt en tom lista som den unifierar med CheckedProof, och listan med bevis raderna. Startvärdet på CheckedProof är null på grund av att när vi börjar så har vi ej jobbat oss igenom några rader bevis. Vad check_proof gör är att den skickar första elementet i Proof listan tillsammans med CheckedProof till predikatet check_line. Om check_line inte orsakar en fail, lägger vi till elementet i listan CheckedProof mha addList, sedan åberopar vi check_proof igen, med nya CheckedProof som innehåller elementet vi just försäkrat oss om är korrekt, och Proof minus det första elementet. Basfallet är en tom lista, vilket signalerar att vi jobbat oss igenom hela Proof listan.

Check_line predikatet accepterar ett element från Proof listan, samt en lista av alla element redan verifierade. Det finns en version av check_line för alla olika regler, samt en version för boxhantering. Beroende på vilken regel som elementet skickas med matchas elementet till en av alla möjliga check_line varianter. Till exempel, om elementet [1, imp(p, q), premise], skickas till check_line, kommer det matchas till versionen av predikatet som hanterar premisser. Vilket är i form av check_line([_, P, premise], CheckedProof). Imp(p, q) kommer unifieras med P, och den överensstämmande sista delen av elementet orsakar att rätt version av check_line väljs. Vad versionen av check_line gör är beroende på hur den motsvarande regeln är utformad. Till exempel ser versionen som tar hand om konjunktion eliminering ut som följande:

```
check_line([_, P, andel1(Line)], CheckedProof):-
member([Line, and(P, _), _], CheckedProof), !.
```

Vi kollar om raden som refereras innehåller P ^ godtyckligt element. Om detta finns är and elimineringen godkänd och predikatet är sant. På följande sätt är resten av reglerna utformade.

BOXHANTERING

Om elementet som skickas till check_line är en box, dvs det ser ut som [Startline, Assumption, assumption]|Restofbox], skapar vi en temporär kopia av CheckedProof. Sedan skickar vi vidare boxen till check_box. Check_box skickar varje rad inuti boxen till check_line till check_line på samma sätt som check_proof med den temporära versionen av CheckedProof. Vi lägger till varje rad vi kollar av i samma lista. När vi härleder en rad från en box kollar vi av om det finns en box som vars första och sista rad matchar den logiska regelns krav. Vi gör detta genom predikatet findBox. Här tittar vi efter en box i CheckedProof vars första och sista rad är av formen som behövs på liknande sätt som i check_line. Vi hittar första raden av boxen när vi identifierar boxen, och sista raden genom getLast predikatet. Boxen finns endast i CheckedProof om den passerat genom check_box, vilket garanterar att den interna logiken är korrekt. Om allt detta är korrekt, är beviset valid.

Problem:
Eftersom vi endast kollar efter första och sista raden i en box, kan en korrekt box med endast två linjer refereras till utanför boxen, vilket inte alltid är korrekt. Detta kan leda till falska true avläsningar.

APPENDIXEN

PREDIKAT	TRUE	FALSE	PURPOSE
verify	När argumentet är en fil i rätt format		Läsa indatafilen och förvara informationen i tre variabler, Prems, Proof, och Goal
addList	När argumenten är en lista, ett element, och en ny lista		Skapar en ny lista i tredje argumentet av form [Element|Lista].
updateList	När argumentet är en lista och ett element		Lägger till elementet i början på listan utan att skapa en ny lista
copyList	När argumenten är två listor		Kopierar den ena listan till den andra element för element
valid_proof	När check_goal och check_proof returnerar true	När check_goal eller check_proof inte returnerar true	Det är hit programmet hoppar efter verify
check_goal	När Goal är samma som atomen i sista raden av Proof		Kolla om vi ens behöver kolla validiteten av Proof.
findLastProof	När argumenten är en Lista och det sista elementet i listan		Hitta sista raden i listan.
check_proof	När argumenten är två listor		Gå igenom Proof linje för linje (eller element för element).
check_line	När linjen från Proof är korrekt härledd från tidigare rader.		Kolla validiteten av Proof.
check_box	När argumenten är tre listor		Kolla validiteten inuti en lista.
findBox	När det finns en rad i TemporaryCheckedProofs där den första		Ett predikat som kollar om det finns en box som kan härleda

	raden är en assumption, och sista raden är identisk mot sista raden av proofboxen.		elementet som den blir kallad av.
getLast	När det andra argumentet är en lista, och sista argumentet är ospecificerat.		Hittar sista raden av en box och förvarar det i tredje argumentet.

PROGRAMKOD

```
/**
The verify predicate takes an input file and
stores the premises, the goal we wish to prove, and
the proof lines into three variables. It then sends these 3
variables to the startpoint of the proof checking, valid_proof.
*/
verify(InputFileName) :- see(InputFileName),
read(Prems), read(Goal), read(Proof),
seen, valid_proof(Prems, Goal, Proof).


/**
Predicate for updating list of proofs checked
*/
addList(List1, Element, [Element|List1]).
```

```
updateList([Element|List1], Element).
updateList(List1, Element):-
updateList([Element|List1], Element).


/**
A predicate which copies list 1 to list 2
*/
copyList([],[]).
copyList([H|T1],[H|T2]) :- copyList(T1,T2).

/**
starts by checking if the last line of the
proof is the same as the goal. Essentially it checks that
the proof somehow ends up with the desired result. Also checks that the proof sequence is
correct.
*/
valid_proof(Prems, Goal, Proof):-
check_goal(Goal, Proof), check_proof(Proof, []).

/**
Check that the goal is the same as the last line of proofs.
*/
check_goal(G, Proof):-
findLastProof(Proof, X), G = X, !.

/**
Iterate through proof until last line is reached, return the proof sequence in X.
*/
findLastProof([[_, Last, _]|[]], Last).
findLastProof([First|Tail], X):-
findLastProof(Tail, X).

/**
Iterate through each line of proofs until we reach end, checking each line as we pass through it
and
adding each checked line of proof to a list we will use to check our following lines against.
*/
check_proof([], _).
check_proof([H|T], CheckedProof):-
check_line(H, CheckedProof), addList(CheckedProof, H, NewCheckedProof), check_proof(T,
NewCheckedProof).
```

```
/**
CHECKING THE VALIDITY OF EACH LINE (EXCLUDING BOXES)
*/


/**
If the line of proof check_line is called with is a premise, we check that it is a member of the list
of premises.
*/
check_line([_, P, premise],_):-
member(P, Prems), !.


/**
If the line of proof check_line is called with is derived using and elimination, we check that the
derived element
exists anded with some other element on the given line in our already checked lines.
*/
check_line([_, P, andel1(Line)], CheckedProof):-
member([Line, and(P, _), _], CheckedProof), !.

check_line([_, P, andel2(Line)], CheckedProof):-
member([Line, and(_, P), _], CheckedProof), !.

check_line([_, P, copy(Line)], CheckedProof):-
member([Line, P, _], CheckedProof), !.

check_line([_, and(X, Y), andint(Line1, Line2)], CheckedProof):-
member([Line1, X, _], CheckedProof), member([Line2, Y, _], CheckedProof), write("andint
performed").

check_line([_, or(X, _), orint1(Line)], CheckedProof):-
member([Line, X, _], CheckedProof), !.

check_line([_, or(_, Y), orint2(Line)], CheckedProof):-
member([Line, Y, _], CheckedProof), !.

check_line([_, P, impel(Line1, Line2)], CheckedProof):-
member([Line1, P1, _], CheckedProof),!, member([Line2, imp(P1, P), _], CheckedProof),!.
```

```
check_line([_, neg(neg(P)), negnegint(Line)], CheckedProof):-
member([Line, P, _], CheckedProof), !.

check_line([_, P, negnegel(Line)], CheckedProof):-
member([Line, neg(neg(P)), _], CheckedProof),!.

check_line([_, neg(P), mt(Line1, Line2)], CheckedProof):-
member([Line1, imp(P, Q), _], CheckedProof), !, member([Line2, neg(Q), _], CheckedProof), !.

check_line([_, or(P, neg(P)), lem], CheckedProof):-
true, !.

check_line([[Startline, Assumption, assumption]|Restofbox], CheckedProof):-
copyList(CheckedProof, TemporaryCheckedProof), updateList(CheckedProof, [Startline,
Assumption, assumption]), check_box([[Startline, Assumption, assumption]|Restofbox],
TemporaryCheckedProof, CheckedProof).

check_line([_, P, contel(Line)], CheckedProof):-
member([Line, cont, _], CheckedProof), !.

check_line([_, cont, negel(Line1, Line2)], CheckedProof):-
member([Line1, P, _], CheckedProof), member([Line2, neg(P), _], CheckedProof), !.

/**
BOX TYPE SHIT BELOW
*/

check_line([_, imp(P, Q), impint(Line1, Line2)], CheckedProof):-
findBox(Line1, Line2, CheckedProof, [Line1, P, assumption], [Line2, Q, _]).


check_line([_, Assumption, assumption], CheckedProof):-
true.

check_line([_, P, pbc(Line1, Line2)], CheckedProof):-
findBox(Line1, Line2, CheckedProof, [Line1, neg(P), assumption], [Line2, cont, _]).

check_line([_, X, orel(Line1, Line2, Line3, Line4, Line5)], CheckedProof):-
member([Line1, or(P, Q), _], CheckedProof), findBox(Line2, Line3, CheckedProof, [Line2, P,
assumption], [Line3, X, _]),
findBox(Line4, Line5, CheckedProof, [Line4, Q, assumption], [Line5, X, _]).

check_line([_, P, negint(Line1, Line2)], CheckedProof):-
```

```
findBox(Line1, Line2, CheckedProof, [Line1, neg(P), assumption], [Line2, cont, _]).


/**
check the validity line by line inside the box by sending each line to check_line. After we are
done, the whole box is added to checked proof.
*/
check_box([Head|[]], TemporaryCheckedProof, [H|CheckedProof]).
check_box([H|T], TemporaryCheckedProof, CheckedProof):-
check_line(H, TemporaryCheckedProof), addList(TemporaryCheckedProof, H,
NewCheckedProof), check_box(T, NewCheckedProof, CheckedProof).

findBox(Line1, Line2, [[Line1, Assumption, assumption]|Tail], [Line1, Assumption, assumption],
Lastline).
findBox(Line1, Line2, [H|T], Firstline, Lastline):-
getLast(Line2, Tail, Lastline),!, findBox(Line1, Line2, [[Line1, Assumption, assumption]|Tail],
[Line1, Assumption, assumption], Lastline).

getLast(Line2, [Head|[]], Head).
getLast(Line2, [H|T], Lastline):-
getLast(Line2, T, Lastline).
```