Sam Khosravi
Robert Minasyan

Labbrapport Lab 2

Algoritmen

Algoritmen ska läsa igenom beviset och kontrollera att det stämmer. För att åstadkomma detta behöver algoritmen kontrollera att premisserna stämmer, målet som vi kommer till är samma som målet vi ska nå och att varje rad stämmer och är rätt enligt satslogik. Samtidigt behöver boxar hanteras och kontrolleras vilket beskrivs senare i rapporten.

```
valid_proof(Prems, Goal, Proof):-
   check_goal(Goal,Proof),
   rule_check1(Prems, Goal, Proof, []).

check_goal(Goal,Proof):-
   last(Proof,LastRow),
   nth1(2,LastRow,Goal).
```

I valid_proof som vi kan se ovan, skickar vi in premisser, målet och beviset. Därefter kallar vi på check_goal, som kommer hitta sista raden genom prologs egna “last” funktion, och hittar andra kolumnen på den raden vilket vi sätter till att vara vårt mål.

Sedan kallar valid_proof på rule_check1 och skickar med premisser, mål, proof och en tom lista.

```
rule_check1(Prems,Goal,[],_ ).
rule_check1(Prems, Goal, [H|TailofProof], Prev):-
   %write('1'),
   rule_check(Prems, Goal, H, Prev),
   %write('2'),
   rule_check1(Prems, Goal, TailofProof, [H|Prev]), !.


rule_check1(Prems, Goal, [[[Line, Result, assumption]|TailofBoxProof]|TailofProof],
Prev):-
%write('innit_ASSUMPTION'),
   rule_check1(Prems, Goal, TailofBoxProof, [[Line,Result,assumption]|Prev]),
%write('cet check'),
   rule_check1(Prems, Goal, TailofProof,
[[[Line,Result,assumption]|TailofBoxProof]|Prev]).
```

rule_check1 har ett base case, där vi slutar ifall vår proof lista blir tom.

Annars delar den upp vårt bevis i en Head och Tail och vi har även en lista vi kallar Prev. Sedan kallar vi på rule_check och skickar premisser, mål, huvudet på listan och Prev. Med detta så kommer vi kolla upp alla regler nedanför en rad i taget och sedan under där vi kallar på rule_check1, men denna gång skickar vi in Tail som listan, och lägger till H till Prev listan varje gång vi kommer in i loopen.

Sedan, beroende på om vi har en assumption eller inte kommer det vara olika vilken rule_check1 vi går till. Den nedre (den över innit_ASSUMPTION), kommer att sätta vänstra delen av vårt bevis till det som finns i vår assumption låda, och TailofProof kommer vara det som fanns innan vi kom till assumption lådan.

Sedan kommer vi till den under innit_ASSUMPTION, och där så kallar vi på rule_check1 igen, och om det är en ny assumption så kommer vi in i denna funktion igen, annars går vi till den rule_check1 innan, och skickar in Tail av assumption lådan och behandlar funktionen på samma sätt som tidigare gjorts.

Därefter på sista metoden i funktionen så kallar vi på rule_check1 igen, men denna gång skickar vi in det som är kvar av tailen som vårt proof, och allt annat i beviset skickar vi in som vår previous lista.

Boxhantering

```
%box
inside_box(Line, [Box|_], Box ) :-
   member([Line, _, assumption], Box).

inside_box(Line, [_|Prev], _):-
   inside_box(Line, Prev, _).
```

ex:

```
%impint
rule_check(Prems, Goal, [_, imp(X,Y), impint(S,K)], Prev):-
   %write(' IMP_INT'),
   inside_box(S, Prev, Box),
   member([S, X, _], Box),
   member([K, Y, _], Box),!.
```

Första funktionen tar in “Line”, vilket är vårt radnummer, och sedan Head på en lista och sätter även denna Head till en vanlig lista, som vi inte vet om än. Efteråt kollar vi om den raden vi är på är en assumption, som även finns i denna Box. Om inte så går den vidare till funktionen under, och där så kollar den Tailen av den inskickade listan, som även kan ha egna listor i sig, och skickar sedan Tailen tillbaka till funktionen över. När vi hittat rätt i member, så kommer prolog att backtracka och sätta rätt värde till Box.

Predikattabell

Predikat	**Sann/falsk**
impel(X,Y)	låt resultatet vara K, sann om S är på rad X och imp(S, K) är på rad Y
impint(S,K)	Låt resultatet vara imp(X,Y), sann om vi öppnar en box och på rad S är det X och vi kommer fram på rad K till Y där vi stänger boxen
andel1(X)	Sann om X är and(X,_) alltså är X första parametern
andel2(X)	Sann om X är and(_,X) alltså är X andra parametern
andint(A,B)	Låt resultatet vara and(X,Y), sann om X är på rad A och Y är på rad B
negel(X,Y)	Låt resultatet på rad X vara Z, sann om på rad Y är negationen av Z (neg(Z))
negint(A,B)	Låt resultatet vara neg(X), sann om vi öppnar en box och X är en assumption på rad A och det är en contradiction (cont) på rad B där vi stänger boxen
negnegel(X)	Låt resultatet vara Y, sann om negnegel(Y) är på rad X
negnegint(X)	Låt resultatet vara negneg(Y), sann om Y är på rad X
orint1(A)	Låt resultatet vara or(X,Y), sann om X är på rad A
orint2(A)	Låt resultatet vara or(X,Y), sann om Y är på rad A
mt(A,B)	Låt resultatet vara neg(X), sann om imp(X,Y) är på rad A och neg(Y) är på rad B
contel(Z)	Sann om det är en contradiction (cont) på rad Z

copy(Y)	Låt resultatet vara X, sann om X är på rad Y
lem	Sann då den använder principen or(X, neg(X)
orel(Q,W,E,R,T)	Låt resultatet vara X, sann om på raden Q det är or(A,B), vi öppnar en box där premissen är A på rad W och vi kommer fram till X på rad E och stänger boxen, sedan öppnas en ny box där B är premissen på rad R och vi kommer fram till X på rad T och stänger boxen
pbc(X,Y)	Låt resultatet vara Z, sann om det är en assumption neg(Z) på rad X och vi kommer fram till en contradiction på rad Y och stänger boxen

Appendix

Programkod

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

valid_proof(Prems, Goal, Proof):-
   check_goal(Goal,Proof),
   rule_check1(Prems, Goal, Proof, []).

check_goal(Goal,Proof):-
   last(Proof,LastRow),
   nth1(2,LastRow,Goal).

rule_check1(Prems,Goal,[],_ ).
rule_check1(Prems, Goal, [H|TailofProof], Prev):-
   %write('1'),
   rule_check(Prems, Goal, H, Prev),
   %write('2'),
   rule_check1(Prems, Goal, TailofProof, [H|Prev]), !.
```

```
rule_check1(Prems, Goal, [[[Line, Result, assumption]|TailofBoxProof]|TailofProof],
Prev):-
%write('innit_ASSUMPTION'),
   rule_check1(Prems, Goal, TailofBoxProof, [[Line,Result,assumption]|Prev]),
%write('cet check'),
   rule_check1(Prems, Goal, TailofProof,
[[[Line,Result,assumption]|TailofBoxProof]|Prev]).
%box
inside_box(Line, [Box|_], Box ) :-
   member([Line, _, assumption], Box).

inside_box(Line, [_|Prev], _):-
   inside_box(Line, Prev, _).

%prem
rule_check(Prems, Goal, [Line,Result, premise], Prev):-
   member(Result, Prems),!.
   %write(' PREMISRULE '), !.

   %impelim
rule_check(Prems,Goal, [Line, Result, impel(X,Y)], Prev):-
%write(' IMPLICATION_ELMINATION ')
member([X, A, _], Prev),
member([Y, imp(A, B), _], Prev),!.


   %andel1
rule_check(Prems, Goal, [Line, Result, andel1(X)], Prev ):-
%write(' andel1 '),
member([X, and(Result, _), _], Prev),!.

   %andel2
rule_check(Prems, Goal, [Line, Result, andel2(X)], Prev ):-
%write(' andel2 '),
member([X, and(_, Result), _], Prev),!.

%negel
rule_check(Prems, Goal, [Line, Result, negel(X,Y)], Prev) :-
%write(' NEG_EL '),
member([X, Z, _], Prev),
member([Y, neg(Z), _], Prev),!.

%contradiction_el

rule_check(Prems, Goal, [Line,Result, contel(Z)], Prev):-
```

```
   %write(' CONTRADICTION_ELIMINATION '),
   member([Z, cont, _], Prev),!.

%negnegel

rule_check(Prems, Goal, [_,Y,negnegel(X)], Prev):-
   %write(' NEGNEG_ELIMINATION '),
   member([X, neg(neg(Y)),_], Prev),!.

%negnegint

rule_check(Prems, Goal, [_,neg(neg(Y)), negnegint(X)], Prev) :-
   %write(' NEGNEG_INT'),
   member([X,Y,_], Prev),!.

%LEM

rule_check(Prems, Goal, [_,or(X, neg(X)), lem], Prev).
   %write(' lEM ').

%impint
rule_check(Prems, Goal, [_, imp(X,Y), impint(S,K)], Prev):-
   %write(' IMP_INT'),
   inside_box(S, Prev, Box),
   member([S, X, _], Box),
   member([K, Y, _], Box),!.

%copy
rule_check(Prems, Goal, [_, X, copy(Y)],Prev):-
   %write(' COPY_RULE '),
   member([Y, X, _], Prev),!.

%impel
rule_check(Prems, Goal, [_, K, impel(X,Y)], Prev):-
%write(' IMPLICATION_ELIMINATION '),
member([X, S, _], Prev),
member([Y, imp(S,K), _], Prev),!.

%orel

rule_check(Prems, Goal, [_, X, orel(Q,W,E,R,T)], Prev):-

member([Q, or(A,B), _], Prev),
inside_box(W, Prev, Box),
member([W, A, assumption], Box),
```

```
member([E, X, _], Box),
inside_box(R, Prev, BoxNext),
member([R, B, assumption], BoxNext),
%write(' OR_ELIMINATION '),
member([T, X, _], BoxNext),!.

%pbc

rule_check(Prems, Goal, [_,Z,pbc(X,Y)], Prev):-
   inside_box(X, Prev, Box),
   member([X, neg(Z), assumption], Box),
   %write(' PBC_RULE '),
   member([Y, cont, _], Box),!.

%negint

rule_check(Prems, Goal, [_, neg(X), negint(A,B)], Prev):-
   inside_box(A, Prev, Box),
   member([A, X, assumption], Box),
   %write(' NEG_INTRODUCTION '),
   member([B,cont, _], Box),!.

%MT_Rule

rule_check(Prems, Goal, [_, neg(X),mt(A,B)], Prev):-
   member([A,imp(X,Y),_], Prev),
   member([B, neg(Y), _], Prev),!.
   %write(' MT_RULE').

%and_introduction

rule_check(Prems, Goal, [_, and(X,Y), andint(A,B)], Prev):-
   member([A,X,_], Prev),
   member([B, Y, _], Prev),!.
   %write('AND_INTRODUCTION').

%OR_INT1
rule_check(Prems, Goal, [_, or(X,Y), orint1(A)], Prev) :-
   member([A, X, _], Prev),!.

%OR_INT2
rule_check(Prems, Goal, [_, or(X,Y), orint2(A)], Prev) :-
   member([A, Y, _], Prev),!.
```

Några exempelbevis

Ex 1 (valid04):

```
[or(p,q),imp(p,r),imp(q,r)].
r.
[
  [1, or(p,q),     premise],
  [2, imp(p,r),    premise],
  [3, imp(q,r),    premise],
  [
        [4, p,              assumption],
        [5, r,     impel(4,2)]
  ],
  [
        [6, q,              assumption],
        [7, r,     impel(6,3)]
  ],
  [8, r,            orel(1,4,5,6,7)]
].
```

Ex 2 (valid13):

```
[neg(neg(and(p,q)))].

q.

[
  [1, neg(neg(and(p,q))),   premise   ],
  [2, and(p,q),             negnegel(1)],
  [3, q,                    andel2(2) ]
].
```

Ex 3 (invalid 04):

```
[p].

p.

[
  [
        [1, p, assumption]
  ]
```

].