Laboration 2 - Beviskontroll - Emil Ståhl
Logik för dataloger DD1351

1. Introduktion

Denna rapport behandlar implementering och testning av ett kontrollprogram för naturlig deduktion skrivet i prolog. Indata till programmet är en sekvent och ett bevis, programmet returnerar sedan antingen ”yes” eller ”no” beroende på om beviset är korrekt eller ej. Programmet har stöd för samtliga regler som normalt används i naturlig deduktion, en komplett lista över dessa regler återfinns i appendix.

2. Generell beskrivning av algoritm

Den givna algoritmen är implementerad för att gå igenom beviset uppifrån och ned. Algoritmen startar i predikatet *verify(InputFileName)* som tar en fil med det bevis som skall prövas och sparar premisser, mål och bevis som sparas i *Prems, Goal* och *Proof*. Efter detta anropas *valid_proof* med dessa tre variabler.

Det *valid_proof* börjar med är att kontrollera om sista raden i beviset är detsamma som målet som söks. Detta görs i predikatet *check_goal* och *findLastProof* genom att rekursivt jobba sig fram till sista raden. Om målet och sista raden ej matchar så terminerar programmet.

Efter detta så körs *check_proof* som itererar igenom beviset, kontrollerar varje rad och lägger till varje kontrollerad rad till *addList* som senare används för att kontrollera kontrollerade rader emot. Detta krävs då algoritmen behandlar beviset uppifrån och ned eftersom programmet på något sätt måste veta vad som förekommit på tidigare rader.

Nu är det dags att kontrollera raderna med *check_line*. Om predikatet anropas med en premiss så kontrolleras att denna premiss finns med i listan av premisser som gavs som input, detta görs med det inbyggda predikatet *member.*

På liknande sätt finns ett predikat *check_line* för varje regel i naturlig deduktion samt en för boxhantering. Prolog mönstermatchar för att använda matchande regel som står på nuvarande rad och kontrollera att regeln använts korrekt. Som exempel kan tas regeln ∧e (and el): För att kontrollera denna regel så söker algoritmen om det givna

elementet förekommer tidigare i beviset bundet genom konjunktion till ett annat godtyckligt element. Övriga regler är implementerade på ett liknande sätt.

2.1 Boxhantering

Om argumentet är en lista av listor innebär detta att det är en box på formen [Startline, Assumption, assumption] | Restofbox]. För att hålla reda på raderna inuti boxen skapas en temporär kopia *temporaryCheckedProof*. Efter detta körs *checkBox* som tar varje rad inuti boxen och skickar till *checkLine* tillsammans med *temporaryCheckedProof* som är samma strategi som *check_proof* gör med de vanliga raderna som beskrivs ovan. Varje kontrollerad rad i boxen läggs dessutom till i samma lista.

För att härleda en rad från en box kontrolleras det om det finns en box där första och sista elementet stämmer överens med den nuvarande logiska regelns krav. Detta görs med hjälp av *findBox*. Algoritmen letar efter en box i *CheckedProof* vars första och sista rad är på formen som behövs på liknande sätt som i *check_line*. Första raden hittas enkelt direkt när boxen identifieras medan sista raden hittas med hjälp av predikatet *getLast*. Det positiva är att den sökta boxen finns endast i *CheckedProof* om och endast om den passerat genom *check_box*. Detta garanterar att boxens interna logik är korrekt enligt algoritmens regler.

Om det som beskrivs i avsnitt 2. och 2.1 är korrekt för ett bevis som indata implicerar det att beviset är korrekt.

2.2 Algoritmens begränsningar

Algoritmen har i sin nuvarande form en del begränsningar som kan leda till vissa felaktiga slutsatser. Detta på grund av att algoritmen endast söker efter den första och sista raden i en box vilket leder till att algoritmen endast kan hantera referenser till en korrekt box med endast två linjer utanför boxen, vilket inte alltid är det korrekta. När algoritmens testkörs med scriptet *run_all_tests* redovisas att algoritmen klarar att kontrollera samtliga valida bevis med godkänt resultat. Dock fås felaktigt resultat vid test av invalid.txt version 1, 3, 8, 9, 18, 20, 22, 27 och 28. Ett förekommande mönster är att bevis som använder boxar med endast en rad i resulterar i problem.

Appendix

Predikat	SANT	FALSKT	Syfte
`verify/1`	Argumentet är en fil given i korrekt format		Läser indata och sparar filens data i tre variabler.
`addList/3`	Argumenten är en lista, ett element och en ny lista		Adderar det givna elementet till den nya listan [Element | Lista]
`updateList/2`	Argumentet är ett element och en lista		Adderar elementet till listan.
`copyList/2`	Argumenten är två listor		Kopierar den första listans element till den andra listan.
`check_goal/2`	Argumentet Goal matchar med atomen på sista raden i beviset (Proof).		Kontrollerar om beviset har önskvärt mål som slutsats. Om inte behöver inte beviset kontrolleras.
`findLastProof/2`	Argumentet är en lista och det sista elementet i listan.		Returnera sista raden i beviset.
`check_proof/2`	Argumenten är två listor		Iterera beviset rad för rad.
`valid_proof/3`	Sann om check_goal och check_proof returnerar true	Falsk om check_goal och check_proof returnerar false	Körs direkt efter predikatet verify.
`check_line/2`	Linjen från Proof är korrekt härledd från rader ovan.		Kontrollera validiteten för beviset.
`check_box/3`	Tre listor som argument		Kontrollera validiteten för en lista / box.
`find_box/5`	När det finns en rad i TemporaryCheckedP roofs där den första och sista raden är identisk mot sista raden av boxen.		Kontrollerar om det finns en box som kan härleda elementet som den blir kallad av.
`getLast/3`	Andra argumentet är en lista som ska itereras och sista elementet är en variabel för det sista elementet i listan.		Går igenom listan och sparar det sista elementet i listan som representerar boxen.

Programkod

```
verify(InputFileName) :- see(InputFileName),
read(Prems), read(Goal), read(Proof),
seen, valid_proof(Prems, Goal, Proof).
addList(List1, Element, [Element|List1]).


updateList([Element|List1], Element).
updateList(List1, Element):- !,
updateList([Element|List1], Element),!.
copyList([],[]).
copyList([H|T1],[H|T2]) :- copyList(T1,T2).
valid_proof(Prems, Goal, Proof):-
check_goal(Goal, Proof), check_proof(Proof, []).
check_goal(G, Proof):-
findLastProof(Proof, X), G = X, !.
findLastProof([[_, Last, _]|[]], Last).
findLastProof([First|Tail], X):-
findLastProof(Tail, X).
check_proof([], _).
check_proof([H|T], CheckedProof):-
check_line(H, CheckedProof), addList(CheckedProof, H, NewCheckedProof),
check_proof(T, NewCheckedProof).
check_line([_, P, premise],_):-!,
member(P, Prems), !.
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), !.
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_box([], List, List2):- true.
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).
```

Exempelbevis:

Korrekt

```
[neg(neg(imp(p, neg(p))))].

neg(p).

[
[1, neg(neg(imp(p,neg(p)))), premise ],
[2, imp(p, neg(p)), negnegel(1)],
[
[3, p, assumption ],
[4, neg(p), impel(3,2) ],
[5, cont, negel(3,4) ]
],
[6, neg(p), negint(3,5)]
].
```

Inkorrekt

```
[and(imp(p, r), imp(q, r))].

imp(r, and(p,q)).

[
[1, and(imp(p, r), imp(q, r)), premise ],
[2, imp(p, r), andel(1) ],
[
[3, and(p, q), assumption ],
[4, p,  andel(3) ],
[5, r, impel(4,2) ]
],
[6, imp(r, and(p, q), impint(3,5) ]
].
```

Results of script `run_all_tests`:

```
valid01.txt passed
valid02.txt passed
valid03.txt passed
valid04.txt passed
valid05.txt passed
valid06.txt passed
valid07.txt passed
valid08.txt passed
valid09.txt passed
valid10.txt passed
valid11.txt passed
valid12.txt passed
valid13.txt passed
valid14.txt passed
```

```
valid15.txt passed
valid16.txt[97,110,100,105,110,116,32,112,101,114,102,111,114,109,101,100] passed
valid17.txt passed
valid18.txt passed
valid19.txt[97,110,100,105,110,116,32,112,101,114,102,111,114,109,101,100] passed
valid20.txt[97,110,100,105,110,116,32,112,101,114,102,111,114,109,101,100] passed
valid21.txt passed
invalid01.txt failed. The proof is invalid but your program accepted it!
invalid02.txt passed
invalid03.txt failed. The proof is invalid but your program accepted it!
invalid04.txt passed
invalid05.txt passed
invalid06.txt passed
invalid07.txt passed
invalid08.txt failed. The proof is invalid but your program accepted it!
invalid09.txt failed. The proof is invalid but your program accepted it!
invalid10.txt[97,110,100,105,110,116,32,112,101,114,102,111,114,109,101,100] passed
invalid11.txt[97,110,100,105,110,116,32,112,101,114,102,111,114,109,101,100] passed
invalid12.txt passed
invalid13.txt passed
invalid14.txt passed
invalid15.txt passed
invalid16.txt passed
invalid17.txt passed
invalid18.txt failed. The proof is invalid but your program accepted it!
invalid19.txt passed
invalid20.txt failed. The proof is invalid but your program accepted it!
invalid21.txt passed
invalid22.txt[97,110,100,105,110,116,32,112,101,114,102,111,114,109,101,100] failed. The
proof is invalid but your program accepted it!
invalid23.txt passed
invalid24.txt passed
invalid25.txt passed
invalid26.txt passed
invalid27.txt failed. The proof is invalid but your program accepted it!
invalid28.txt failed. The proof is invalid but your program accepted it!
Emils-MBP-15:Tests emilstahl$
```

The basic rules of natural deduction:

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