Laboration 1
Beviskontroll med Prolog

Kungliga Tekniska Högskolan
Logik för dataloger
DD1350, HT2015

Mikael Forsberg <miforsb@kth.se>, Robin Gunning <rgunning@kth.se>

3 december 2015

Innehåll

1 Inledning

Denna laboration gick ut på att implementera ett program för automatisk kontroll av satslogiska bevis med naturlig deduktion i programspråket Prolog.

2 Metod

I detta avsnitt presenteras den algoritm som utformades för att utföra beviskontroller, och som sedan implementerades i Prolog.

2.1 Algoritm - informell beskrivning

Vår algoritm arbetar sig igenom beviset uppifrån och ned. Varje rad innehåller en regelmotivation. Algoritmen validerar för varje rad att den hänvisade regeln tillämpats korrekt på de rader som eventuellt hänvisats till. Boxar hanteras med rekursion, genom att mer eller mindre se på innehållet som ett eget bevis, som dock kan hänvisa till gällande formler utanför boxen (sådana formler måste finnas ovanför boxen).

2.2 Algoritm - formell beskrivning

Indata: en lista med premisser P, en slutsats S och ett bevis B där B utgörs av en lista där varje element $E_i, i = 0, 1, 2, ..., n$ antingen är en trippel på formen (*radnummer*, *påstående*, *motivering*) eller en lista innehållande fler element (en box).

```
<Triple> ::= tuple(lineno, statement, motivation)
<P> ::= [] | cons(statement, <P>)
<S> ::= statement
<B> ::= cons(<Triple>, <B>) | cons(<B>, <B>) | cons(<Triple>, [])
```

I grammatiken ovan skall `cons` tolkas som att vara definierad så att det går att lägga en lista inuti en annan lista, dvs. `cons(<List>, <List>)`.

Definitioner:

- Tp(n) : den trippel $X \in B$ för vilken Ln(X) = n
- Ln(X) : radnumret i en trippel X
- St(X) : påståendet i en trippel X
- St(n) : påståendet St(Tp(n))
- Mt(X) : motiveringen i en trippel X
- Rs(X) : de radnummer på samma boxnivå som hänvisas till i Mt(X)
- Rd(X) : de radnummer på djupare boxnivå som hänvisas till i Mt(X)

För samtliga definitioner ovan gäller att beviset B finns tillgängligt implicit. Symbolen n skall tolkas som ett heltal. Den överlagrade funktionen St(...) skall tolkas verka på tripplar för argument med stor bokstav, och heltal för argument med liten bokstav.

Att validera ett bevis B givet en lista med premisser P samt en slutsats S:

- Markera samtliga radnummer som stängda
- Markera samtliga radnummer i tripplar på första nivån i B som öppna.
- Validera det första elementet $E_0 \in B$
- Kontrollera att det sista elementet $E_n \in B$ är en trippel och att St(E_n) = S.

Att validera ett element X givet ett bevis B samt en lista med premisser P:

OM X är en trippel

OM $\text{Ln}(X)$ är stängt
misslyckas

OM något radnummer $\in \text{Rs}(X)$ är stängt
misslyckas

OM något radnummer $\in \text{Rs}(X)$ är större eller lika med $\text{Ln}(X)$
misslyckas

OM något radnummer $\in \text{Rd}(X)$ är öppet
misslyckas

VÄLJ $\text{Mt}(X)$ ur:

premiss:
OM $\text{St}(X) \notin P$
misslyckas

copy(k):
OM $\text{St}(X) \neq \text{St}(k)$
misslyckas

negationseliminering(k,m):
OM $\text{St}(k) \neq \neg\text{St}(m)$ **ELLER** $\text{St}(X) \neq \bot$
misslyckas

falsumeliminering(k):
OM $\text{St}(k) \neq \bot$
misslyckas

icke-ickeintroduktion(k):
OM $\text{St}(X) \neq \neg\neg\text{St}(k)$
misslyckas

LEM:
OM INTE $\text{St}(X)$ är på formen $p \vee \neg p$
misslyckas

MT(k,m):
OM $\text{St}(k) \neq (p \rightarrow q)$ **ELLER** $\text{St}(m) \neq \neg q$ **ELLER** $\text{St}(X) \neq \neg q$
misslyckas

implikationseliminering(k,m):
OM $\text{St}(m) \neq (\text{St}(k) \rightarrow \text{St}(X))$
misslyckas

och-eliminering-1(k):
OM INTE $\text{St}(k)$ är på formen $\text{St}(X) \wedge q$
misslyckas

och-eliminering-2(k):
OM INTE $\text{St}(k)$ är på formen $p \wedge \text{St}(X)$
misslyckas

och-introduktion(k,m):
OM $\text{St}(X) \neq \text{St}(k) \wedge \text{St}(m)$
misslyckas

icke-icke-eliminering(k):
OM $\text{St}(k) \neq \neg\neg\text{St}(X)$
misslyckas

antagande:
OM INTE $\text{Ln}(X)$ öppnar en box med djup > 0
misslyckas

eller-eliminering(k,m,p,q,r):

OM INTE raderna m och p öppnar respektive stänger en och samma box
misslyckas

OM INTE raderna q och r öppnar respektive stänger en och samma box
misslyckas

OM $\mathrm{St}(k) \neq (\mathrm{St}(m) \vee \mathrm{St}(q))$
misslyckas

OM $\mathrm{St}(p) \neq \mathrm{St}(r)$ **ELLER** $\mathrm{St}(X) \neq \mathrm{St}(p)$
misslyckas

pbc(k,m):

OM INTE raderna k och m öppnar respektive stänger senast stängda[1] box
misslyckas

OM $\mathrm{St}(k) \neq \neg\mathrm{St}(X)$ **ELLER** $\mathrm{St}(m) \neq \bot$
misslyckas

implikationsintroduktion(k,m):

OM INTE raderna k och m öppnar respektive stänger senast stängda[2] box
misslyckas

OM $\mathrm{St}(X) \neq (\mathrm{St}(k) \rightarrow \mathrm{St}(m))$
misslyckas

negationsintroduktion(k,m):

OM INTE raderna k och m öppnar respektive stänger senast stängda[3] box
misslyckas

OM $\mathrm{St}(m) \neq \bot$ **ELLER** $\mathrm{St}(X) \neq \neg\mathrm{St}(k)$
misslyckas

ANNARS // X är inte en trippel

OM INTE X är en lista där element 0 är en trippel motiverad med ”antagande”
misslyckas

Markera samtliga rader på djup 0 i X som öppna

Validera X

Markera samtliga rader på djup 0 i X som stängda

OM $\mathrm{Tp}(\mathrm{Ln}(X)+1)$ inte är tomma listan

Validera $\mathrm{Tp}(\mathrm{Ln}(X)+1)$

ANNARS

färdigt, korrekt!

2.3 Algoritmens boxhantering

Som ses i algoritmens formella beskrivning ovan hanteras boxar genom att programmet kan märka vissa radnummer som öppna eller stängda. När en box påträffas i ett beviskräd (eller ett boxträd) markeras raderna på den första nivån i trädet som öppna och valideras enligt precis samma algoritm som elementen i huvudbeviset. När boxen stängts stängs även raderna. Vid kontroll av bevisregler görs olika kontroller för hänvisningar till boxar, vilket bland annat omfattar att kontrollera så samtliga rader i hänvisade boxar är stängda.

[1] Den box som, sett från det radnummer man står på, senast stängdes.
[2] Se fotnot 1
[3] Se fotnot 1

2.4 Predikatdokumentation

Predikat	Beskrivning
`andel1/2`	`andel1(?X, ?Y)` Verifierar den satslogiska bevisregeln $\wedge e_1$. Sant om X = and(Y, _).
`andel2/2`	`andel2(?X, ?Y)` Verifierar den satslogiska bevisregeln $\wedge e_2$. Sant om X = and(_, Y).
`andint/3`	`andint(?X, ?Y, ?Z)` Verifierar den satslogiska bevisregeln $\wedge i$. Sant om Z = and(X, Y).
`contel/2`	`contel(?X, ?Y)` Verifierar den satslogiska bevisregeln $\bot e$. Sant om X = $\bot$.
`copy/2`	`copy(?X, ?Y)` Verifierar den satslogiska bevisregeln `copy`. Sant om X = Y.
`impel/3`	`impel(?X, ?Y, ?Z)` Verifierar den satslogiska bevisregeln $\rightarrow e$. Sant om Y = imp(X, Z).
`impint/3`	`impint(?X, ?Y, ?Z)` Verifierar den satslogiska bevisregeln $\rightarrow i$. Sant om Z = imp(X, Y).
`lem/1`	`lem(?X)` Verifierar den satslogiska bevisregeln LEM. Sant om X = or(Y, $\neg$ Y) för något Y.
`mt/3`	`mt(?X, ?Y, ?Z)` Verifierar den satslogiska bevisregeln MT (modus tollens). Sant om X = imp(A, B), Y = neg(A) och Z = neg(B).
`negel/3`	`negel(?X, ?Y, ?Z)` Verifierar den satslogiska bevisregeln $\neg e$. Sant om Y = neg(X) och Z = $\bot$.
`negint/3`	`negint(?X, ?Y, ?Z)` Verifierar den satslogiska bevisregeln $\neg i$. Sant om Z = neg(X) och Y = $\bot$.
`negnegel/2`	`negnegel(?X, ?Y)` Verifierar den satslogiska bevisregeln $\neg\neg e$. Sant om X = neg(neg(Y)).
`negnegint/2`	`negnegint(?X, ?Y)` Verifierar den satslogiska bevisregeln $\neg\neg i$. Sant om Y = neg(neg(X)).
`orel/6`	`orel(?X, ?Y, ?Z, ?P, ?Q, ?R)` Verifierar den satslogiska bevisregeln $\vee e$. Sant om X = or(Y, P) och Z = Q = R.

Predikat	Beskrivning
pbc/3	`pbc(?X, ?Y, ?Z)` Verifierar den satslogiska bevisregeln PBC. Sant om X = neg(Z) och Y = $\perp$.
open/1	`open(?X)` Sant om open(X) gäller. Används dynamiskt med `assert`/`retract`.
all_open/2	`all_open(?X, ?Y)` Sant om open(X) och open(Y) gäller.
all_open/3	`all_open(?X, ?Y, ?Z)` Sant om open(X), open(Y) och open(Z) gäller.
none_open/2	`none_open(?X, ?Y)` Sant om varken open(X) eller open(Y) gäller.
none_open/3	`none_open(?X, ?Y, ?Z)` Sant om varken open(X), open(Y) eller open(Z) gäller.
box_mark_open/1	`box_mark_open(+Box)` Markerar samtliga radnummer i alla tripplar på lägsta nivån i beviset/boxen Box som öppna (`open`) genom anrop till `assert`. Alltid sant.
box_mark_closed/1	`box_mark_closed(+Box)` Markerar samtliga radnummer i alla tripplar på lägsta nivån i beviset/boxen Box som stängda genom anrop till `retract`. Alltid sant.
maxbox/2	`maxbox(+List, -Result)` Plockar fram det högsta ”box-id” i en lista beräknad av `lines_collect_info/4`. Sant då `Result` är detta högsta ”box-id”.
lines_collect_info/2	`lines_collect_info(+Proof, -Result)` Går igenom ett bevisträd och beräknar utifrån samtliga tripplar en flat lista på formen `[[radnummer, box-id, radnummer-i-box], ...]`. Sant då `Result` är denna lista.
lines_collect_info/5	`-` Hjälppredikat för `lines_collect_info/2`.
line_in_box/3	`line_in_box(+Proof, +LineNo, ?Box)` Söker reda på vilken box som radnummret `LineNo` tillhör. Sant då `Box` är detta ”box-id”.
linenumbers/2	`linenumbers(+Proof, -List)` Skapar en lista på samtliga radnummer som förekommer i `Proof`. Sant då `List` är denna lista.
increasing/1	`increasing(+List)` Sant om `List` innehåller tal ordnade i icke-strikt stigande ordning.
lines_find_info/3	`-` Hjälppredikat för `line_get_info/3`.
line_get_info/3	`line_get_info(+Proof, +LineNo, -Result)` Hämtar trippeln för ett givet radnummer ur ett bevisträd. Sant då `Result` är denna trippel.

Predikat	Beskrivning
`box_max_line/3`	`box_max_line(+Proof, +BoxId, -Result)` Finner det högsta radnummer (globalt i beviset) som tillhör den box som har det givna ”box-id” `BoxId`. Detta ”box-id” beräknas på samma sätt som hos `lines_collect_info/2`. Sant när `Result` är detta radnummer.
`box_max_line/4`	- Hjälppredikat för `box_max_line/3`.
`box_min_line/3`	`box_min_line(+Proof, +BoxId, -Result)` Finner det lägsta radnummer (globalt i beviset) som tillhör den box som har det givna ”box-id” `BoxId`. Detta ”box-id” beräknas på samma sätt som hos `lines_collect_info/2`. Sant när `Result` är detta radnummer.
`box_min_line/4`	- Hjälppredikat för `box_min_line/3`.
`line_opens_box/2`	`line_opens_box(+Proof, ?LineNo)` Sant om någon box öppnas på rad `LineNo` i beviset.
`line_closes_box/2`	`line_closes_box(+Proof, ?LineNo)` Sant om någon box stängs på rad `LineNo` i beviset.
`lines_delimit_box/3`	`lines_delimit_box(+Proof, ?Line1, ?Line2)` Sant om någon box öppnas på rad `Line1` och stängs på rad `Line2` i beviset. `Line1` och `Line2` måste tillhöra samma box och finnas på samma djup.
`first_opened_box_in_box/3`	`first_opened_box_in_box(+Proof, +OwnBox, ?BoxNo)` Finner ”box-id” på den första box som öppnas i den box vars ”box-id” är `OwnBox`. Sant när `BoxNo` är detta ”box-id”.
`most_recently_closed_box/3`	`most_recently_closed_box(+Proof, +LineNo, ?BoxNo)` Finner ”box-id” på den box som senast stängdes sett från ett givet radnummer `LineNo`. Sant när `BoxNo` är detta ”box-id”.
`proof_nth_line/3`	`proof_nth_line(+Proof, +N, ?Line)` Hämtar den trippel i `Proof` vars radnummer är `N`. Sant när `Line` är denna trippel.
`proof_nth_statement/3`	`proof_nth_statement(+Proof, +N, ?Statement)` Hämtar påståendet ur trippeln vars radnummer är `N`. Sant när `Statement` är detta påstående.
`verify/1`	`verify(+Filename)` Läser in premisser, slutsats och bevis från filen `Filename`. Om beviset är korrekt skrivs ”yes” ut på standard out, annars skrivs ”no” ut på standard out. Predikatet i sig är alltid sant.
`validate_proof/3`	`validate_proof(+Premises, +Conclusion, +Proof)` Validerar ett bevis enligt tidigare beskrivna algoritm. Sant om beviset är korrekt.
`validate/3`	`validate(+Proof, +Premises, +Element)` Validerar ett element enligt tidigare beskrivna algoritm. Sant om algoritmen inte lyckas hitta någon anledning att misslyckas.

3 Resultat

Programmet klarade samtliga testfall (valid01-valid21, invalid01-invalid28) som fanns tillgängliga via kurshemsidan, samt våra egna två testfall (se bilagor).

4 Bilagor

4.1 labb1.pl

```
1  :- initialization
2      consult('natrules.pl'),
3      consult('proofs.pl').
4
5  verify(Filename):- %obs anvand single quote for filename
6      see(Filename),
7      read(Premises),
8      read(Conclusion),
9      read(Proof),
10     seen,
11     box_mark_open(Proof),
12     (valid_proof(Premises, Conclusion, Proof) ->
13         write('yes'), nl, box_mark_closed(Proof);
14         write('no'), nl, box_mark_closed(Proof), fail).
15
16 valid_proof(Premises, Conclusion, Proof):-
17     validate(Proof, Premises, Proof),
18     last(Proof, [_, Conclusion, _]).
19
20 validate(Proof, Premises, [[LineNo, Statement, premise]|Tail]):-
21     !,
22     open(LineNo),
23     memberchk(Statement, Premises),
24     validate(Proof, Premises, Tail).
25
26 validate(Proof, Premises, [[LineNo, /*Statement*/_, assumption]|Tail]):-
27     !,
28     open(LineNo),
29     line_opens_box(Proof, LineNo),
30     validate(Proof, Premises, Tail).
31
32 validate(Proof, Premises, [[LineNo, Statement, copy(K)]|Tail]):-
33     !,
34     K < LineNo,
35     all_open(LineNo,K),
36     proof_nth_statement(Proof, K, StatementK),
37     copy(StatementK, Statement),
38     validate(Proof, Premises, Tail).
39
40 validate(Proof, Premises, [[LineNo, cont, negel(K,M)]|Tail]):-
41     !,
42     K < LineNo,
43     M < LineNo,
44     all_open(LineNo, K, M),
45     proof_nth_statement(Proof, K, StatementK),
46     proof_nth_statement(Proof, M, StatementM),
47     negel(StatementK, StatementM, cont),
48     validate(Proof, Premises, Tail).
49
50 validate(Proof, Premises, [[LineNo, Statement, orel(O, BS1, BE1, BS2, BE2)]|Tail]):-
51     !,
52     O < LineNo,
53     BS1 < LineNo,
54     BE1 < LineNo,
55     BS2 < LineNo,
56     BE2 < LineNo,
57     all_open(LineNo, O),
58     lines_delimit_box(Proof, BS1, BE1),
59     lines_delimit_box(Proof, BS2, BE2),
60     none_open(BS1, BE1),
61     none_open(BS2, BE2),
62     proof_nth_statement(Proof, O, StatementO),
```

```
63      proof_nth_statement(Proof, BS1, StatementBS1),
64      proof_nth_statement(Proof, BE1, StatementBE1),
65      proof_nth_statement(Proof, BS2, StatementBS2),
66      proof_nth_statement(Proof, BE2, StatementBE2),
67      orel(StatementO, StatementBS1, StatementBE1, StatementBS2, StatementBE2,
            Statement),
68      validate(Proof, Premises, Tail).
69
70  validate(Proof, Premises, [[LineNo, Statement, pbc(K,M)]|Tail]):-
71      !,
72      most_recently_closed_box(Proof, LineNo, BoxNo),
73      box_min_line(Proof, BoxNo, K),
74      box_max_line(Proof, BoxNo, M),
75      open(LineNo),
76      none_open(K, M),
77      lines_delimit_box(Proof, K, M),
78      proof_nth_statement(Proof, K, StatementK),
79      proof_nth_statement(Proof, M, StatementM),
80      pbc(StatementK, StatementM, Statement),
81      validate(Proof, Premises, Tail).
82
83  validate(Proof, Premises, [[LineNo, Statement, contel(K)]|Tail]):-
84      !,
85      K < LineNo,
86      all_open(LineNo, K),
87      proof_nth_statement(Proof, K, StatementK),
88      contel(StatementK, Statement),
89      validate(Proof, Premises, Tail).
90
91  validate(Proof, Premises, [[LineNo, Statement, negnegint(K)]|Tail]):-
92      !,
93      K < LineNo,
94      all_open(LineNo, K),
95      proof_nth_statement(Proof, K, StatementK),
96      negnegint(StatementK, Statement),
97      validate(Proof, Premises, Tail).
98
99  validate(Proof, Premises, [[LineNo, Statement, lem]|Tail]):-
100     !,
101     open(LineNo),
102     lem(Statement),
103     validate(Proof, Premises, Tail).
104
105 validate(Proof, Premises, [[LineNo, Statement, mt(K,M)]|Tail]):-
106     !,
107     K < LineNo,
108     M < LineNo,
109     all_open(LineNo, K, M),
110     proof_nth_statement(Proof, K, StatementK),
111     proof_nth_statement(Proof, M, StatementM),
112     mt(StatementK, StatementM, Statement),
113     validate(Proof, Premises, Tail).
114
115 validate(Proof, Premises, [[LineNo, Statement, impint(K,M)]|Tail]):-
116     !,
117     most_recently_closed_box(Proof, LineNo, BoxNo),
118     box_min_line(Proof, BoxNo, K),
119     box_max_line(Proof, BoxNo, M),
120     open(LineNo),
121     none_open(K, M),
122     proof_nth_statement(Proof, K, StatementK),
123     proof_nth_statement(Proof, M, StatementM),
124     /*same_box(K,M),*/
125     impint(StatementK, StatementM, Statement),
126     validate(Proof, Premises, Tail).
127
128 validate(Proof, Premises, [[LineNo, Statement, impel(K,M)]|Tail]):-
129     !,
130     K < LineNo,
131     M < LineNo,
132     open(LineNo),
133     proof_nth_statement(Proof, K, StatementK),
134     proof_nth_statement(Proof, M, StatementM),
```

```
135     impel(StatementK, StatementM, Statement),
136     validate(Proof, Premises, Tail).
137
138 validate(Proof, Premises, [[LineNo, Statement, andel1(K)]|Tail]):-
139     !,
140     K < LineNo,
141     open(LineNo),
142     proof_nth_statement(Proof, K, StatementK),
143     andel1(StatementK, Statement),
144     validate(Proof, Premises, Tail).
145
146 validate(Proof, Premises, [[LineNo, Statement, andel2(K)]|Tail]):-
147     !,
148     K < LineNo,
149     open(LineNo),
150     proof_nth_statement(Proof, K, StatementK),
151     andel2(StatementK, Statement),
152     validate(Proof, Premises, Tail).
153
154 validate(Proof, Premises, [[LineNo, Statement, andint(K,M)]|Tail]):-
155     !,
156     K < LineNo,
157     M < LineNo,
158     all_open(LineNo, K, M),
159     proof_nth_statement(Proof, K, StatementK),
160     proof_nth_statement(Proof, M, StatementM),
161     /*same_box(K,M),*/
162     andint(StatementK, StatementM, Statement),
163     validate(Proof, Premises, Tail).
164
165 validate(Proof, Premises, [[LineNo, Statement, negint(K,M)]|Tail]):-
166     !,
167     most_recently_closed_box(Proof, LineNo, BoxNo),
168     box_min_line(Proof, BoxNo, K),
169     box_max_line(Proof, BoxNo, M),
170     open(LineNo),
171     none_open(K, M),
172     lines_delimit_box(Proof, K, M),
173     proof_nth_statement(Proof, K, StatementK),
174     proof_nth_statement(Proof, M, cont),
175     negint(StatementK, cont, Statement),
176     validate(Proof, Premises, Tail).
177
178 validate(Proof, Premises, [[LineNo, Statement, negnegel(K)]|Tail]):-
179     !,
180     K < LineNo,
181     open(LineNo),
182     proof_nth_statement(Proof, K, StatementK),
183     negnegel(StatementK, Statement),
184     validate(Proof, Premises, Tail).
185
186 validate(Proof, Premises, [Head|Tail]):-
187     !,
188     [[_,_,assumption]|_] = Head,
189     box_mark_open(Head),
190     validate(Proof, Premises, Head),
191     box_mark_closed(Head),
192     validate(Proof, Premises, Tail).
193
194 validate(/*Proof*/_, /*Premises*/_, []).
195
196 % main:-verify('tests/testvalid.txt').
197 main(X):-
198     verify(X).
```

4.2 proofs.pl

```
1  :- dynamic
2      open/1.
3
4  % space-saving macros for open/1
5  all_open(A,B):-open(A),open(B).
6  all_open(A,B,C):-open(A),open(B),open(C).
7  none_open(A,B):-not(open(A)),not(open(B)).
8  none_open(A,B,C):-not(open(A)),not(open(B)),not(open(C)).
9
10 % box_mark_open/1 (+List)
11 % marks the line numbers of each non-box entry in List as open
12 box_mark_open([Head|Tail]):-
13     [LineNo, /*Statement*/_, /*Motivation*/_] = Head,
14     !,
15     assert(open(LineNo)),
16     box_mark_open(Tail).
17
18 % skip boxes
19 box_mark_open([_|Tail]):-
20     box_mark_open(Tail).
21
22 % end recursion
23 box_mark_open([]).
24
25 % box_mark_closed/1 (+List)
26 % marks the line numbers of each non-box entry in List as closed
27 box_mark_closed([Head|Tail]):-
28     [LineNo, /*Statement*/_, /*Motivation*/_] = Head,
29     !,
30     retract(open(LineNo)),
31     box_mark_closed(Tail).
32
33 % skip boxes
34 box_mark_closed([_|Tail]):-
35     box_mark_closed(Tail).
36
37 % end recursion
38 box_mark_closed([]).
39
40 % maxbox/2 (+List, -Result)
41 % helper for lines_collect_info/4
42 % finds the greatest "box id" in a (potentially partial) list as
43 % computed by lines_collect_info/4
44 maxbox([[_,B,_]|Tail], Y):-
45     maxbox(Tail, B1),
46     (B1 > B -> Y = B1; Y = B).
47
48 maxbox([], 0).
49
50 % lines_collect_info/2 (+Proof, -Result)
51 % defers to lines_collect_info/4
52 %
53 % given a proof tree, computes a flat list with elements of the form
54 % [linenumber, boxnumber, boxlinenumber]
55 %
56 % example: [[1,0,0], [[2,0,0]], [3,0,0]] -> [1,0,0],[2,1,0],[3,0,1]
57 lines_collect_info(Proof, Result):-
58     lines_collect_info(Proof, 0, 0, 0, Result).
59
60 % lines_collect_info/4, see lines_collect_info/2
61 lines_collect_info([[Head|_]|Tail], BoxLevel, BoxCount, N, [[Head,BoxLevel,N]|More])
       :-
62     not(is_list(Head)),
63     !,
64     Next is N + 1,
65     lines_collect_info(Tail, BoxLevel, BoxCount, Next, More).
66
67 lines_collect_info([Head|Tail], BoxLevel, BoxCount, N, More):-
68     NewLevel is BoxCount + 1,
69     lines_collect_info(Head, NewLevel, NewLevel, 0, More1),
70     maxbox(More1, NewCount),
```

```
71      lines_collect_info(Tail, BoxLevel, NewCount, N, More2),
72      append(More1, More2, More).
73
74  lines_collect_info([], /*BoxLevel*/_, /*BoxCount*/_, /*N*/_, []).
75
76  % line_in_box/3 (+Proof, +LineNo, ?Box)
77  %
78  % given a proof tree and a linenumber, determine whether or not
79  % the linenumber is contained within a certain box
80  line_in_box(Proof, LineNo, Box):-
81      lines_collect_info(Proof, Lines),
82      member(Line, Lines),
83      Line = [LineNo, Box, /*Nth*/_].
84
85  % linenumbers/2 (+Proof, -Result)
86  %
87  % given a proof tree, computes a list of all the line numbers
88  % that appear in the tree
89  linenumbers([[Head|_]|Tail], [Head|More]):-
90      not(is_list(Head)),
91      !,
92      linenumbers(Tail, More).
93
94  linenumbers([Head|Tail], More):-
95      linenumbers(Head, More1),
96      linenumbers(Tail, More2),
97      append(More1, More2, More).
98
99  linenumbers([], []).
100
101 % increasing/1 (+List)
102 %
103 % true if the items in the list are (non-strictly) increasing
104 increasing([A,B|Tail]):-
105     A =< B,
106     increasing(Tail).
107
108 increasing([_|[]]).
109 increasing([]).
110
111 % lines_find_info/3 (+List, +LineNo, -Result)
112 % helper for line_get_info/3. extracts the data for a single line
113 % number from a list as computed by lines_collect_info/4.
114
115 % found the line
116 lines_find_info([[LineNo, BoxNo, BoxNth]|_], LineNo, [LineNo, BoxNo, BoxNth]).
117
118 % skip to next
119 lines_find_info([_|Tail], LineNo, Result):-
120     lines_find_info(Tail, LineNo, Result).
121
122 % line_get_info/3 (+Proof, +LineNo, -Result)
123 % extracts the data for a given line number in a given proof.
124 line_get_info(Proof, LineNo, Result):-
125     lines_collect_info(Proof, Lines),
126     lines_find_info(Lines, LineNo, Result).
127
128 % box_max_line/3 (+Proof, +Box, -Result)
129 % finds the largest line number contained in a given "box id"
130 box_max_line(Proof, Box, Result):-
131     lines_collect_info(Proof, Lines),
132     box_max_line(Lines, Box, 0, Result),!.
133
134 % box_max_line/4 (+List, +Box, ?Max, -Result)
135 % given a list as computed by lines_collect_info/4, finds the
136 % largest line number contained in a given "box id"
137
138 % in requested box, update result
139 box_max_line([Head|Tail], Box, Max, Result):-
140     [LineNo, Box, _] = Head,
141     (integer(Max) -> NewMax is max(LineNo, Max); NewMax is LineNo),
142     box_max_line(Tail, Box, NewMax, Result),!.
143
```

```
144 % in some other box, skip and keep going
145 box_max_line([_|Tail], Box, Max, Result):-
146     box_max_line(Tail, Box, Max, Result),!.
147
148 % end of recursion
149 box_max_line([], _, Max, Max).
150
151 % box_min_line/3 (+Proof, +Box, -Result)
152 % finds the smallest line number contained in a given "box id"
153 box_min_line(Proof, Box, Result):-
154     lines_collect_info(Proof, Lines),
155     box_min_line(Lines, Box, 9999999999999, Result),!.
156
157 % box_min_line/4 (+List, +Box, ?Min, -Result)
158 % given a list as computed by lines_collect_info/4, finds the
159 % smallest line number contained in a given "box id"
160
161 % in requested box, update result
162 box_min_line([Head|Tail], Box, Min, Result):-
163     [LineNo, Box, _] = Head,
164     (integer(Min) -> NewMin is min(LineNo, Min); NewMin is LineNo),
165     box_min_line(Tail, Box, NewMin, Result),!.
166
167 % in some other box, skip and keep going
168 box_min_line([_|Tail], Box, Min, Result):-
169     box_min_line(Tail, Box, Min, Result),!.
170
171 % end of recursion
172 box_min_line([], _, Min, Min).
173
174 % line_opens_box/2 (+Proof, +LineNo)
175 % true if a given line number opens a box in a given proof
176 line_opens_box(Proof, LineNo):-
177     line_get_info(Proof, LineNo, Info),
178     [LineNo, Box, 0] = Info,
179     Box > 0.
180
181 % line_closes_box/2 (+Proof, +LineNo)
182 % true if a given line closes a box in a given proof
183 line_closes_box(Proof, LineNo):-
184     lines_collect_info(Proof, Lines),
185     line_get_info(Proof, LineNo, Info),
186     [LineNo, Box, _] = Info,
187     Box > 0,
188     box_max_line(Lines, Box, LineNo).
189
190 % lines_delimit_box/3 (+Proof, +Line1, +Line2)
191 % true if two given lines are the starting and ending lines of
192 % some box in a given proof
193 lines_delimit_box(Proof, Line1, Line2):-
194     line_get_info(Proof, Line1, Info1),
195     line_get_info(Proof, Line2, /*Info2*/_),
196     [Line1, Box, 0] = Info1,
197     box_max_line(Proof, Box, Line2).
198
199 % first_opened_box_in_box/3 (+List, +OwnBox, -BoxNo)
200 % given a list as computed by lines_collect_info/4, finds the
201 % "box id" of the first box that is opened inside a given "box id"
202 % todo: is this broken? how about boxes containing no boxes?
203
204 % skip items at start of list until finding a box id larger than
205 % the requested box id
206 first_opened_box_in_box([[/*LineNo*/_, BoxNr, /*BoxLineNr*/_]|Tail], OwnBox, BoxNo):-
207     BoxNr =< OwnBox,
208     !,
209     first_opened_box_in_box(Tail, OwnBox, BoxNo).
210
211 % first variant of predicate failed, so we found what we're looking for
212 first_opened_box_in_box([[_, BoxNr, _]|/*Tail*/_], _, BoxNr).
213
214 % most_recently_closed_box/3 (+Proof, +LineNo, -BoxNo)
215 % given a proof and a line number, finds which "box id" was most
216 % recently closed prior to the appearance of the given line number
```

```
217 most_recently_closed_box(Proof, LineNo, BoxNo):-
218     line_in_box(Proof, LineNo, OwnBox),
219     lines_collect_info(Proof, Lines),
220     first_opened_box_in_box(Lines, OwnBox, BoxNo).
221
222 % proof_nth_line/3 (+Proof, +N, -Line)
223 % given a proof and a line number N, finds the data for the
224 % item with line number N if it exists
225
226 % found it
227 proof_nth_line([Head|_], N, Line):-
228     [N, Statement, Motivation] = Head,
229     !,
230     Line = [N, Statement, Motivation].
231
232 % found a box, dig deeper
233 proof_nth_line([Head|_], N, Line):-
234     Head = X,
235     proof_nth_line(X, N, Line).
236
237 % keep going
238 proof_nth_line([_|Tail], N, Line):-
239     proof_nth_line(Tail, N, Line).
240
241 % proof_nth_statement/3 (+Proof, +N, -Statement)
242 % given a proof and a line number N, finds the statement part
243 % of the item with line number N if it exists
244 proof_nth_statement(Proof, N, Statement):-
245     proof_nth_line(Proof, N, LineN),
246     [_, Statement, _] = LineN.
```

4.3 natrules.pl

```
1  andel1(and(A, /*B*/_), A). % and elimination for first variable
2
3  andel2(and(/*A*/_, B), B). % and elimination for second variable
4
5  andint(A, B, and(A, B)). % and introduction
6
7  contel(cont, _). % contradiction elimination
8
9  copy(A, A). % copy
10
11 impel(A, imp(A, B), B). % implication elimination
12
13 impint(A, B, imp(A, B)). % implication introduction
14
15 lem(or(A,neg(A))). % law of excluded middle
16
17 mt(imp(A,B),neg(B),neg(A)). % modus tollens
18
19 negel(A, neg(A), cont). % negation elimination
20
21 negint(A, cont, neg(A)). % negation introduction
22
23 negnegel(neg(neg(A)), A). % double negation elimination
24
25 negnegint(A, neg(neg(A))). % double negation introduction
26
27 orel(or(A,B), A, X, B, X, X). % or elimination
28
29 pbc(neg(A), cont, A). % proof by contradiction
```

4.4 ourvalid.txt

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

q.

[
  [1, imp(neg(r),and(p,or(q,r))),    premise],
  [2, p,                  premise],
  [3, neg(r),             premise],
  [4, and(p,or(q,r)),     impel(3,1)],
  [5, or(q,r),            andel2(4)],
  [
    [6, neg(q),           assumption],
    [
      [7, q,              assumption],
      [8, cont,           negel(7,6)]
    ],
    [
      [9, r,              assumption],
      [10, cont,           negel(9,3)]
    ],
    [11, cont,            orel(5,7,8,9,10)]
  ],
  [12, q,                 pbc(6,11)]
].
```

4.5 ourinvalid.txt

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

and(p,r).

[
[1, q, premise],
[2, imp(p, q), premise],
[3, imp(q, r), premise],
[4, r, impel(1, 3)],
[
  [5, p, assumption],
  [6, q, copy(1)],
  [7, p, copy(5)]
],
[8, imp(q, p), impint(6, 7)],
[9, p, impel(1,8)],
[10, and(p, r), andint(4,9)]
].
```