Diar Sabri, 19940227-4196
DD1351 Lab 2

Inledning

Syftet med laborationen var att skapa ett Prolog-program för att testa korrektheten i bevis. Som indata ges ett bevis i naturlig deduktion till vilket programmet returnerar true eller false om beviset är giltigt eller inte.

Metod

Programmet kontrollerar beviset rad för rad och börjar uppifrån och traverserar ner. Alla tidigare rader sparas i en lista *validated*. Beviset är korrekt om alla rader är korrekta och om sista raden är vårt mål.

Generellt sett fungerar programmet genom att kontrollera alla regler med *rule_control*. När regeln är kontrollerad läggs raden till på *validated*. Om det är en premiss kollar vi enkelt om raden finns i *Prems*. Det kluriga med labben gällde hanteringen av boxar. Ifall vi har ett antagande så går vi in i ett “delprogram” som rekursivt går genom svansen av boxen och validerar den som ett vanligt bevis rekursivt. För fall där det inte är ett antagande, exempelvis i regler som PBC måste vi hämta boxen med *box_get*.

Reglerna skapades rätt intuitivt, genom att kolla på formelbladet och förstå hur indatat ser ut. I princip varje regel använde *member* som används för att kolla tillhörighet i en lista, i vårt fall *validated*.

Predikat	Sant
verify	Alla rader verifierade
valid_proof	Tom lista
valid_proof	Traverserat beviset utan problem och sista raden är slutmålet
valid_proof	Om första raden är ett antagande och boxen är korrekt
valid_proof	Om raden är en regel/premiss/antagande, och dessa är korrekta
box_control	Om BoxT är tom
box_control	Om det är ett antagande, och om det rekursiva anropet är korrekt
box_control	Om det är en regel, och resten av boxen är korrekt
box_get	Om boxen börjar på raden
box_get	Om boxen innehåller raden annars
rule_control	Om regeln är korrekt

I alla andra fall är det falskt.

Korrekt bevis

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

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

```
[
  [1, and(imp(p,q),imp(p,r)),  premise],
  [2, imp(p,q),            andel1(1)],
  [3, imp(p,r),            andel2(1)],
  [
        [4, p,             assumption],
        [5, q,             impel(4,2)],
        [6, r,             impel(4,3)],
        [7, and(q,r),            andint(5,6)]
  ],
  [8, imp(p,and(q,r)),           impint(4,7)]
].
```

Inkorrekt bevis

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

neg(q).

```
[

        [1, imp(p,q), premise  ],
        [2, neg(q),   premise  ],

        [
        [3, p,    assumption ],
        [4, q,    impel(1,2) ]
        ],
        [5, neg(q),       pbc(1,5)]
].
```

Kod

```
% Diar Sabri
% Lab2 DD1351

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

% Last line in the proof should be the goal
valid_proof(Prems, Goal, [], [[Row, Goal, Rule]|Validated]).

% If box in the first line then it's an assumption.
% Box-control then continues with proof
valid_proof(Prems, Goal, [[[Row, Result, assumption]|BoxT]|ProofT],
Validated) :-
    box_control(Prems, Goal, BoxT, [[Row, Result,
assumption]|Validated]),
    valid_proof(Prems, Goal, ProofT, [[[Row, Result,
assumption]|BoxT]|Validated]).
```

```
% If not box, checks rule and continues with proof
valid_proof(Prems, Goal, [ProofH|ProofT], Validated) :-
    rule_control(Prems, ProofH, Validated),
    valid_proof(Prems, Goal, ProofT, [ProofH|Validated]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% BOXES %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Base case (when proof ends)
box_control(Prems, Goal, [], Validated).

% If first row is an assumption.
% Recursive call again on rest of box contents
box_control(Prems, Goal, [[[Row, Result, assumption]|BoxT]|ProofT],
Validated) :-
    box_control(Prems, Goal, BoxT, [[Row, Result,
assumption]|Validated]).

% If not another box.
% Rule checking in the box, recursive.
box_control(Prems, Goal, [ProofH|ProofT], Validated) :-
    rule_control(Prems, ProofH, Validated),
    box_control(Prems, Goal, ProofT, [ProofH|Validated]).


% Retreives box on specified row
box_get(Row, [BoxH|Validated], BoxH) :-
    member([Row, _, _], BoxH).

% Recursive otherwise
box_get(Row, [_|Validated], Box) :-
    box_get(Row, Validated, Box).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% RULES %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Premise
```

```
% Controls that Res is a part of the premises
rule_control(Prems, [Row, Result, premise], Validated) :-
    member(Result, Prems).

% And-introduction
% andint(X,Y). X,Y = row numbers for And1 & And2
% Checks that And1 & And2 is on row X & Y
rule_control(Prems, [Row, and(And1, And2), andint(X,Y)], Validated) :-
    member([X, And1, _], Validated),
    member([Y, And2, _], Validated).

% And-elimination 1
% Checks whether Res is okay on row X
rule_control(Prems, [Row, Result, andel1(X)], Validated) :-
    member([X, and(Result, _), _], Validated).

% And-elimination 2
% Checks whether Res is okay on row X
rule_control(Prems, [Row, Result, andel2(X)], Validated) :-
    member([X, and(_, Result), _], Validated).


% Or-introduction 1
% Checks whether Res is true on row X
rule_control(Prems, [Row, or(Result, _), orint1(X)], Validated) :-
    member([X, Result, _], Validated).

% Or-introduction 2
% Checks whether Res is true on row X
rule_control(Prems, [Row, or(_, Result), orint2(X)], Validated) :-
    member([X, Result, _], Validated).

% Or-elimination
% X - or, Y - box1, V - box2, U&W - Res
rule_control(Prems, [Row, Result, orel(X,Y,U,V,W)], Validated) :-
    box_get(Y, Validated, Box1),                                    %
Box1
    box_get(V, Validated, Box2),                                    %
Box2
```

```
    member([X, or(First, Second), _], Validated),                    %
Check so that an OR exists
    member([Y, First, _], Box1),                                     %
Check y exists in box1
    member([U, Result, _], Box1),                                    %
Check u is Res in box1
    member([V, Second, _], Box2),                                    %
Check v esists in box2
    member([W, Result, _], Box2).                                    %
Check u is Res in box2


% Imply-introduction
% X,Y - Row numbers
rule_control(Prems, [Row, imp(Imp, Result), impint(X,Y)], Validated) :-
    box_get(X, Validated, Box),
    member([X, Imp, assumption], Box),                    % Row X should
have Imp as an assumption
    member([Y, Result, _], Box).                          % Row Y should
have Res

% Imply-elimination
% Checks existence of Imp and Imp->Res among Validated
rule_control(Prems, [Row, Result, impel(X,Y)], Validated) :-
    member([X, Imp, _], Validated),
    member([Y, imp(Imp, Result), _], Validated).


% Negate-introduction
% Get box -> check assumption -> check contradiction
rule_control(Prems, [Row, neg(Result), negint(X,Y)], Validated) :-
    box_get(X, Validated, Box),
    member([X, Result, assumption], Box),
    member([Y, cont, _], Box).

% Negate-elimination
% X has to have Q and Y has to have !Q -> contradiction
rule_control(Prems, [Row, cont, negel(X,Y)], Validated) :-
    member([X, Cont, _], Validated),
    member([Y, neg(Cont), _], Validated).
```

```
% Contradict-elimination
% Simple check for if we have a contradiction on row X
rule_control(Prems, [Row, _, contel(X)], Validated) :-
    member([X, cont, _], Validated).


% NegNeg-introduction
% Simple double negation introduction
rule_control(Prems, [Row, neg(neg(Result)), negnegint(X)], Validated)
:-
    member([X, Result, _], Validated).

% NegNeg-elimination
% Checks for existence of double negation
rule_control(Prems, [Row, Result, negnegel(X)], Validated) :-
    member([X, neg(neg(Result)), _], Validated).


% MT
% Checks for existence of implication and negation
rule_control(Prems, [Row, neg(P), mt(X,Y)], Validated) :-
    member([X, imp(P, Q), _], Validated),
    member([Y, neg(Q), _], Validated).

% PBC
% Checks for assumption of a negation and a contradiction at the end of
the box
rule_control(Prems, [Row, Result, pbc(X,Y)], Validated) :-
    box_get(X, Validated, Box),
    member([X, neg(Result), assumption], Box),
    member([Y, cont, _], Box).

% LEM
% Trivial
rule_control(Prems, [Row, or(X, neg(X)), lem], Validated).

% Copy
% Controls existence of what we're trying to copy
```

```
rule_control(Prems, [Row, Result, copy(X)], Validated) :-
    member([X, Result, _], Validated).
```