Azer Hojlas

Lab assignment 2- Proof checker

Description and purpose:

I have written a program in prolog with it's purpose being the ability to conclude whether a proof in propositional logic (and natural deduction) is valid. The command prompt will return either 'true' or 'false' depending on the validity of the input (proof). The program does this by running the proof against a database of all the rules of natural deduction.

The assignment is already partly predefined for us, in order to start easier. It starts with a verify predicate where premises (Prems), proof and the goal of the proof are inputted into the program through a specific file format that has been predetermined for this assignment. These are then forwarded to 'validate_proof' which tests whether the proof in question is valid or not. This is done with the help of two further predicates, 'match_goal' and 'verify_proof', that as their respective names suggest, check if the goal matches the result and if the proof is correctly written (by running it through the before mentioned database). If both predicates are true, then the proof in question is true.

Box handling predicate:

If a row in the proof happens to be a box, it will be verified by the assumption rule through the 'verify_row' predicate. For this rule I have divided the proof into a head (read *assumption*) and a tail '*BoxTail'*, which will be evident soon as to why. In my program, if there is a row (list) inside of a list with an assumption, then the 'verify_proof' will be called upon itself again, i.e it will iterate through the box by treating the box as a new proof. Thus it will recheck all the rules for the individual rows in the box as if it was a proof all on its own. The head with '*assumption'* will be added to the list of *'VerifiedRows'*, and the new proof will be the tail '*BoxTail'*, i.e the rest of the box. This way the program will ensure that the logic inside of the box is correct. This also works for boxes inside of boxes, as the rule will just call upon itself again.

With rules that are dependent on the box rule, I have created a specific variable, in most cases called '*BoxFinder'*, that will be assigned the specific box in question through the predefined member rule. Depending on the rule, various operations will then be performed with this *'BoxFinder'* variable.

Here is the specific code so that the reader can follow my thought process easier.

```
verify_row(Prems,[[_,_,assumption]|BoxTail],VerifiedRows):-
   verify_proof(Prems,BoxTail,[[_,_,assumption]|VerifiedRows]).
```

Predicates chart:

predicate	**True when**	**False when**	**Purpose**
Verify	The file in question is correctly formatted according to a beforehand agreed on format	It isn’t	Takes an input file and saves the file’s premises, goal and proof in the variables Prems, Goal and proof. These are then sent to the other predicates that start the control of the proof.
validate_proof	Both match_goal and verify_proof.	Either match_goal, verify_proof or both are false	This rule takes in prems, goal and proof. This rule is pivotal in the code, as it is the rule that concludes if the proof is true or not. It’s outcome is decided by the two predicates match_goal and verify_proof.
match_goal	The goal of the file matches the last row in the proof, i.e also the goal, which should be identical.	The goal and the last row aren’t identical.	From the validate_proof predicate, Proof and Goal are forwarded. It’s purpose is to control if the last line of the proof matches the goal. This is done through the predefined ‘last’ and ‘nth1’. Last adds the last line in proof to a new list LastRow. Nth checks the second element in that same last row to see if its identical to the goal

addList	The arguments are a list element, a list and a new list.	It isn’t	This predicate puts an element to the bottom of a list, using appendEl.
AppendEl	Similar to the predicate above.	Similar to the predicate above.	Similar to append, but whereas an append rule adds to the front of the list, appendel adds to the last.
verify_row	The rows in the proof abide by the rules that I have defined.	The predicate can’t recognize the logic in the proof, i.e the rules.	This predicate is what I use to check all the existing rules of propositional logic in natural deduction. All the list elements in the proof have to match some rule that I have made.

Limitations:
The code can only run files written in a specific predefined format that contains premises (Prems), proof and a goal. Trying to prove tautologies would not work, because these only have a goal.

The program also probably would not be able to generate proofs for sequences because I do not see this as a possibility. The code is made only to take input and return either true or false. Generation is out of the question because that is a long stretch from validation which is the main and only function of the program.

Appendix

Code:

```
%___________________ Predefined code ________________%

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

validate_proof(Prems, Goal, Proof):-
   match_goal(Goal,Proof),
   verify_proof(Prems,Proof,[]), !.

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




%_____________________Help functions _________________%


addList(H,VerifiedRows,NewList):-
   appendEl(H,VerifiedRows,NewList).

appendEl(X,[],[X]).
appendEl(X,[H|T],[H|Y]):-
   appendEl(X,T,Y).

%___________________ Verifies the proof ______________________%

verify_proof(_,[],_).
verify_proof(Prems,[H|T],VerifiedRows):-
   verify_row(Prems,H,VerifiedRows),
   addList(H,VerifiedRows,NewList),
   verify_proof(Prems,T,NewList).


%__________________Predefined rules_____________________%

/* These rules are already defined
imp(p, q)
and(or(p, q), r)
imp(neg(p), cont)
and(and(and(p,q),r),s)
```

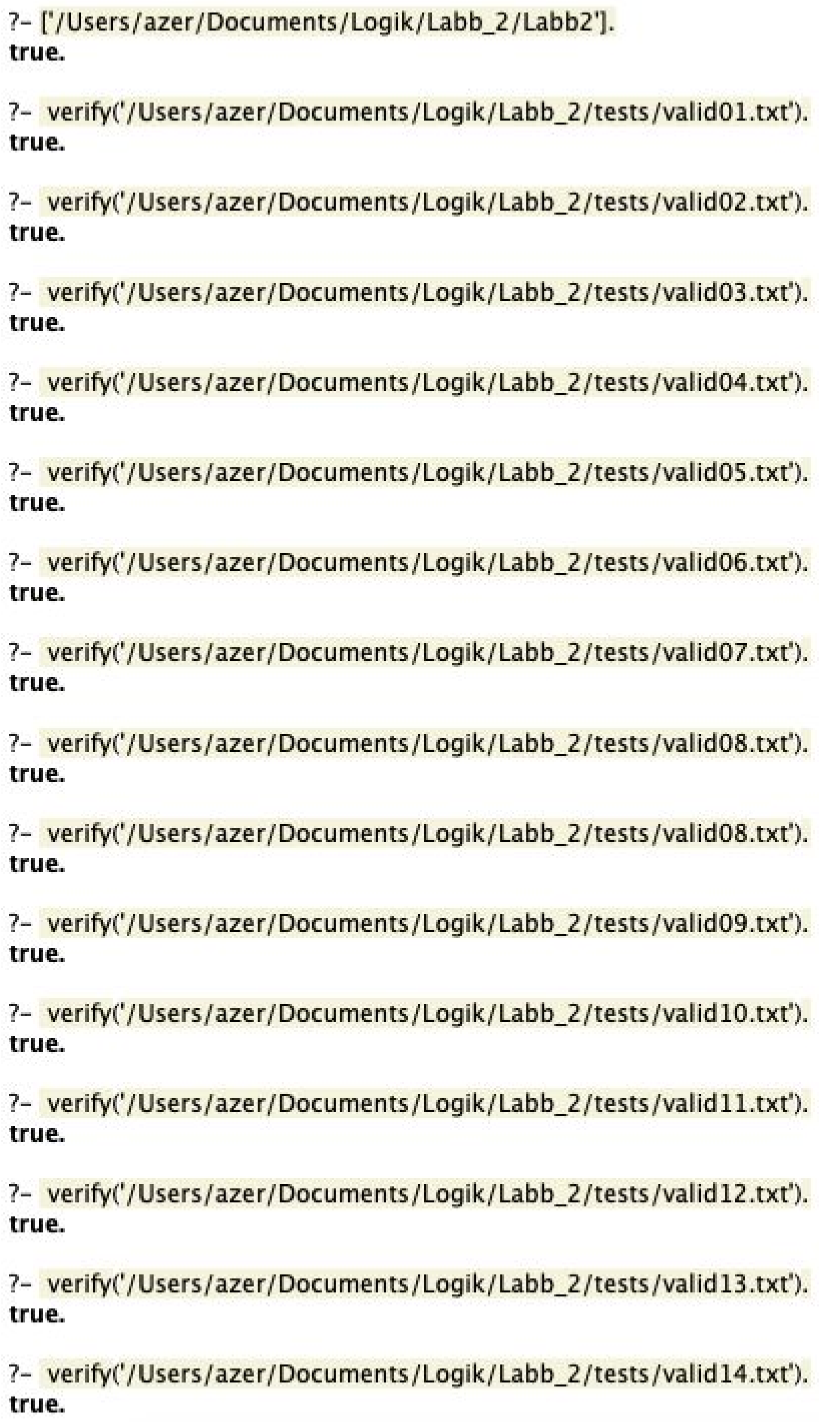
```
cont
*/

%___________________________Rules_________________________________________%

%_____Premise rule_____%
verify_row(Prems,[_,Rule, premise],_):-
   member(Rule,Prems).


%_________________________ Assumptions, i,e Boxes _____________________%


verify_row(Prems,[[_,_,assumption]|T],VerifiedRows):-
   verify_proof(Prems,T,[[_,_,assumption]|VerifiedRows]).


%____________________AND rules___________________________________________%




%_____________And introduction______%
verify_row(_,[_,and(Atom1,Atom2),andint(RowNumber1,RowNumber2)],VerifiedRows):-
   member([RowNumber1,Atom1,_],VerifiedRows),
   member([RowNumber2,Atom2,_],VerifiedRows).


%______AND elimination 1_____%
verify_row(_,[_,Atom,andel1(RowNumber)],VerifiedRows):-
   member( [RowNumber,and(Atom,_),_], VerifiedRows).


%______AND elimination 2_____%
verify_row(_,[_,Atom,andel2(RowNumber)],VerifiedRows):-
   member([RowNumber,and(_,Atom),_], VerifiedRows).


%_______________________________OR rules_____________________________%

% ___OR introduction 1______%
verify_row(_,[_,or(Atom,_),orint1(RowNumber)],VerifiedRows):-
   member([RowNumber,Atom,_],VerifiedRows).


%_____OR introduction 2_______%
verify_row(_,[_,or(_,Atom),orint2(RowNumber)],VerifiedRows):-
   member([RowNumber,Atom,_],VerifiedRows).
```

```
% _____ Or Elimination Rule ___%

verify_row(_,[_,Conclusion,orel(RowNumber,AssumptionNumber,ConclusionNumber1,Assu
mptionNumber2,ConclusionNumber2)],VerifiedRows):-
   member( [RowNumber, or(EliminatedAtom1,EliminatedAtom2),_], VerifiedRows),
   member( BoxFinder, VerifiedRows),
   member([AssumptionNumber,EliminatedAtom1,assumption],BoxFinder),
   member([ConclusionNumber1,Conclusion,_],BoxFinder),
   member(BoxFinder2,VerifiedRows),
   member([AssumptionNumber2, EliminatedAtom2, assumption],BoxFinder2),
   member([ConclusionNumber2,Conclusion,_],BoxFinder2).
```

```
%____________Implication rules_____________%

%___ Implication introduction___%
verify_row(_,[_,imp(Atom1,Atom2),impint(RowNumber1,RowNumber2)],VerifiedRows):-
   member(BoxFinder,VerifiedRows),
   member([RowNumber1,Atom1,assumption],BoxFinder),
   member([RowNumber2,Atom2,_],BoxFinder).

%___ implication elimination_______%
verify_row(_, [_, Atom1, impel(RowNumber1, RowNumber2)], VerifiedRows) :-
        member([RowNumber1, Atom2, _], VerifiedRows),
        member([RowNumber2, imp(Atom2,Atom1),_], VerifiedRows).


%_____________________ Negation Rules___________________________________%

%_______Negation introduction_____%
verify_row(_,[_,neg(Atom),negint(RowNumber1,RowNumber2)],VerifiedRows):-
   member(BoxFinder,VerifiedRows),
   member([RowNumber1,Atom,assumption],BoxFinder),
   member([RowNumber2,cont,_],BoxFinder).


%______Negation negation introduction___%
verify_row(_,[_,neg(neg(Atom)),negnegint(RowNumber)],VerifiedRows):-
```

```
   member([RowNumber,Atom,_],VerifiedRows).


%______Negation elimination______%
verify_row(_,[_,cont,negel(RowNumber1,RowNumber2)],VerifiedRows) :-
         member([RowNumber1,Atom,_],VerifiedRows),
         member([RowNumber2,neg(Atom),_],VerifiedRows).


%__________Negation negation elimination
verify_row(_,[_,Atom,negnegel(RowNumber)],VerifiedRows):-
   member([RowNumber,neg(neg(Atom)),_],VerifiedRows).



%___________________Various other rules%_____________


%___ Copy Rule___%
verify_row(_,[_,Atom,copy(RowNumber)],VerifiedRows):-
   member([RowNumber,Atom,_],VerifiedRows).


%___Contradiction Elimination___%
verify_row(_,[_,_,contel(RowNumber)],VerifiedRows):-
   member([RowNumber,cont,_],VerifiedRows).


%___LEM___%
verify_row(_,[_,or(Atom,neg(Atom)),lem],_).



%___Modus Tollens______%
verify_row(_,[_,neg(Atom1),mt(RowNumber1,RowNumber2)],VerifiedRows):-
   member([RowNumber1,imp(Atom1,Atom2),_],VerifiedRows),
   member([RowNumber2,neg(Atom2),_],VerifiedRows).


%___ Proof by contradiction ___%
verify_row(_,[_,Atom,pbc(RowNumber1,RowNumber2)],VerifiedRows):-
   member(BoxFinder,VerifiedRows),
member([RowNumber1,neg(Atom),assumption],BoxFinder),
member([RowNumber2,cont,_],BoxFinder).


%__________________________END OF CODE__________________________%
```

Results of tests:

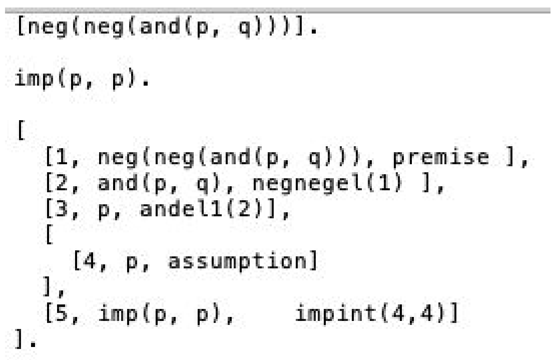


```
?- ['/Users/azer/Documents/Logik/Labb_2/Labb2'].
true.

?-  verify('/Users/azer/Documents/Logik/Labb_2/tests/valid01.txt').
true.

?-  verify('/Users/azer/Documents/Logik/Labb_2/tests/valid02.txt').
true.

?-  verify('/Users/azer/Documents/Logik/Labb_2/tests/valid03.txt').
true.

?-  verify('/Users/azer/Documents/Logik/Labb_2/tests/valid04.txt').
true.

?-  verify('/Users/azer/Documents/Logik/Labb_2/tests/valid05.txt').
true.

?-  verify('/Users/azer/Documents/Logik/Labb_2/tests/valid06.txt').
true.

?-  verify('/Users/azer/Documents/Logik/Labb_2/tests/valid07.txt').
true.

?-  verify('/Users/azer/Documents/Logik/Labb_2/tests/valid08.txt').
true.

?-  verify('/Users/azer/Documents/Logik/Labb_2/tests/valid08.txt').
true.

?-  verify('/Users/azer/Documents/Logik/Labb_2/tests/valid09.txt').
true.

?-  verify('/Users/azer/Documents/Logik/Labb_2/tests/valid10.txt').
true.

?-  verify('/Users/azer/Documents/Logik/Labb_2/tests/valid11.txt').
true.

?-  verify('/Users/azer/Documents/Logik/Labb_2/tests/valid12.txt').
true.

?-  verify('/Users/azer/Documents/Logik/Labb_2/tests/valid13.txt').
true.

?-  verify('/Users/azer/Documents/Logik/Labb_2/tests/valid14.txt').
true.
```

```
?- verify('/Users/azer/Documents/Logik/Labb_2/tests/valid15.txt').
true.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/valid16.txt').
true.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/valid17.txt').
true.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/valid18.txt').
true.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/valid19.txt').
true.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/valid20.txt').
true.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/valid21.txt').
true.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid01.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid02.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid03.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid04.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid05.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid06.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid07.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid08.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid09.txt').
false.
```

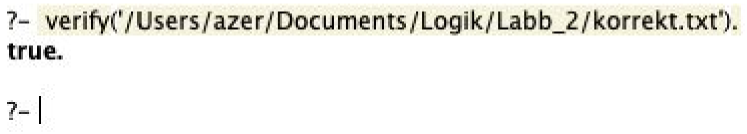
```
?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid09.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid10.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid11.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid12.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid13.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid14.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid15.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid16.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid17.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid18.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid19.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid20.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid21.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid22.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid23.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid24.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid25.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid26.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid27.txt').
false.

?- verify('/Users/azer/Documents/Logik/Labb_2/tests/invalid28.txt').
false.
```

Correct proof:



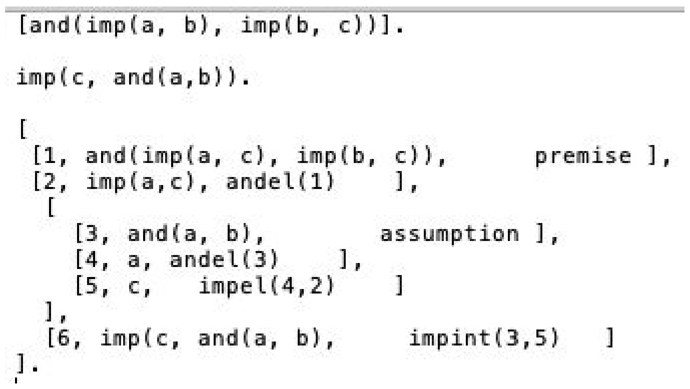
```
[neg(neg(and(p, q)))].

imp(p, p).

[
  [1, neg(neg(and(p, q))), premise ],
  [2, and(p, q), negnegel(1) ],
  [3, p, andel1(2)],
  [
    [4, p, assumption]
  ],
  [5, imp(p, p),    impint(4,4)]
].
```

result:

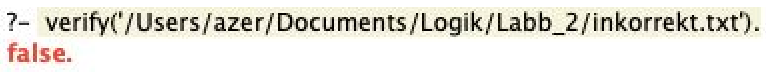


```
?- verify('/Users/azer/Documents/Logik/Labb_2/korrekt.txt').
true.

?- |
```

Incorrect proof:



```
[and(imp(a, b), imp(b, c))].

imp(c, and(a,b)).

[
 [1, and(imp(a, c), imp(b, c)),      premise ],
 [2, imp(a,c), andel(1)    ],
  [
    [3, and(a, b),        assumption ],
    [4, a, andel(3)    ],
    [5, c,   impel(4,2)    ]
  ],
  [6, imp(c, and(a, b),     impint(3,5)    ]
].
```



```
?- verify('/Users/azer/Documents/Logik/Labb_2/inkorrekt.txt').
false.
```