Lab 2- Proof checker
Logic for Computer Scientists, DD1351

Hiba Qutbuddin Habib
Ayah Babiker

November 2022

Introduction

The aim of this lab is to create a program, that given various different proofs written using natural deduction should verify if this proof is valid.The proof will be provided in its whole with the "sekvent" and all the lines that compromise the proof. To check if the proof is indeed valid one must check that all the rules that exist in the scope of natural deduction are followed. If the rules have been followed throughout the whole proof and there are no deviations from this then the proof can be classified as correct. However, if the proof makes an invalid assumption or uses a rule in the wrong way or state something that is not logically correct the proof should be deemed incorrect. The algorithm and program will be written using the programming language Prolog.

Method for Program Formation

To be able to create the program that accurately checks the proof one can follow different steps to create a working proof checker. The first step is to check the goal of the "sekvent" and compare it to the last line, this is done because if the proof does not end with the goal you can immediately say that the proof is incorrect and you can end it there. This is important to have since you will not waste time searching the whole proof for no reason since you already know that the end result of the proof is not what it is supposed to be. After this you can do the actual proof and rule checking. The proof is represented as a list so one will traverse the list line for line and validate that the line is acceptable. Once you have checked a line you will put it in another list which will contain the part of the proof you have already checked and considered legitimate. One will have finished going through the proof once the proof list is empty and one can then terminate the proof control. However, the program needs to know the rules it should follow to be able to decide what is right and what is wrong. This means you have to define the different rules that are used when doing a natural

deduction proof. Using predicates one will represent the things that are allowed when using a certain rule and if this is not followed you can end the proof check.

Handling Boxes

In natural deduction when using certain rules you need to create a box, for example when you have an assumption you must open a box and then close it once you have stopped using the assumption. Additionally, one has to use boxes for example for the rules or elimination, implication introduction and PBC. Since proofs can have boxes the program needs to somehow take care of these and there are various ways to do this. One way which was implemented was to look for sub-lists in the actual proof. Since the representation of the proof is created in a way that when a box is started a new list was created inside the general proof. So to know when a box is reached you have to check if there is a sub-list or not and when you do have one you can check the last and first lines of the box and when you have a rule that needs a box you use a box checking to make sure that the rule is followed.

Implementation of Program

To implement the proof checker one would have to follow the method of the program formation to then translate this to actual Prolog code to create the program. This will be predicates. First we need the predicate that will check the "sekvent" and the last line of the code to make sure the first step is fulfilled and if not the proof checking can be ended which is done with a goal checking predicate. The next step is to do the rule checking as mentioned in the method and this means that you need to define each rule. This was done using all the different proof validation to define the rules and how they are supposed to be used. The specific predicates will be shown in a table below. Then we also had to take care of the boxes and this was done with a box checking predicate would know when a box is found since you will have a sub-list inside the proof.

Results/Predicate Table

Predicate	Validity
verify	This predicate is the one that allows us to read the proof and get the information about all the components of the proof
valid_proof	This one check if the proof that we have received is in fact a correct proof or not it checks if the last element is goal uses proof checking and goal checking will be true if both goal checking and proof checking are true
proof_checking	This one goes through the proof and the rules to see if the rules are followed will be true when there are no errors in the proof that do not follow the rules
goal_checking	This one compares the goal and the "sekvent" will be true when they are the same and it is not an assumption
box_cheking	Checks if a box is found and will be true when you find a sub-list in the proof
proof_validation(prems)	This one checks and handles the premise will be true when you receive one
proof_validation(assumption)	Checks and handles assumptions will be true when you receive and assumption
proof_validation(copy)	Checks and handles copy will be true when copy is used and the 2 variables are the same one in the current line and previous
proof_validation(andint)	Will be true when both variables are found separately in the proof before and they were true
proof_validation(andel1)	Taking away the first variable from an and formula and will be true when an and formula is present in proof
proof_validation(andel2)	Taking away the second variable from an and formula and will be true when an and formula is present in proof
proof_validation(orint1)	Adding an or when the first variable has been found to be true earlier in the proof
proof_validation(orint2)	Adding an or when the second variable has been found to be true earlier in the proof
proof_validation(orel)	Removing an or when an or statement can be found and a box has been opened and you assume the first variable and reach a result and then do the same thing with the second variable they should reach the same result, will be true when this is achieved
proof_validation(impel)	You need to have the implication earlier in the proof and the first variable is true then it will be true
proof_validation(impint)	Will be true when you have opened a box and assumed the first variable and then you should come to the second variable then this predicate will be true
proof_validation(negel)	Will be true when a variable and its negations is found earlier the proof
proof_validation(negint)	Opened a box where you have a variable and it is an assumption and then you should reach a contradiction then this will be true
proof_validation(negnegel)	Will be true when a negneg is found earlier in the proof
proof_validation(contel)	When a contradiction is found in the proof this predicate will be true you can introduce any variable
proof_validation(MT)	Earlier in the proof there should be an implication and there should also be a negation of the second variable then you can reach the negation of the first variable
proof_validation(PBC)	Open a box and in the beginning you should have a negated variable which is an assumption and you should reach a contradiction and this will give the non-negated variable.
proof_validation(LEM)	For this to be true you will reach an or statement for a variable and its negation

Appendix

The example proof

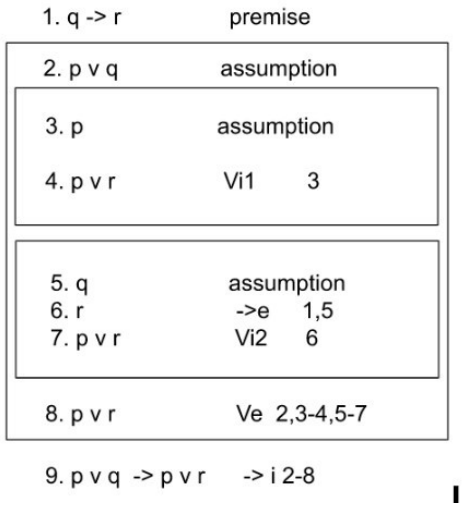

Figure 1: Non trivial valid proof

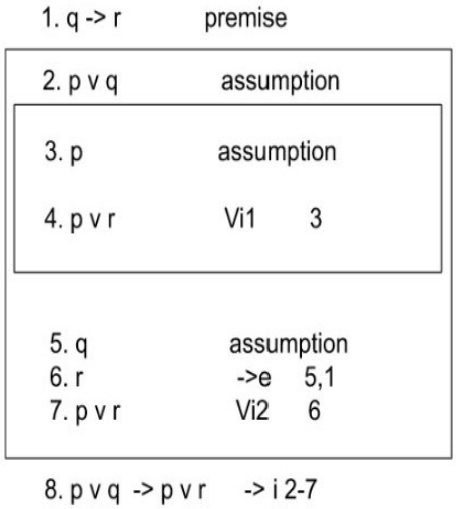

Figure 2: Non trivial invalid proof

[1]

Code for the proof checker

```
:- discontiguous proof_validation/3.

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


%cheking if the last element is goal and cheking is the proof is correct
valid_proof(Prems,Goal,Proof) :- goal_checking(Proof,Goal),
proof_Checking(Prems,Proof,[]), !.


% Cheking if the goal and the last line is the same, if the goal and the last line
% is the same but its en assumption then it fails
goal_checking(Proof,Goal):-last(Proof,[_,Goal,assumption]),!,fail.
goal_checking(Proof,Goal):-last(Proof,[_,Goal,_]).


% checking if the proof is valid, if we have empty list then we are done
proof_Checking(_,[],_).

proof_Checking(Prems,[H|T],Already_checked):-
proof_validation(Prems,H,Already_checked),
append(Already_checked,[H],NewList),
proof_Checking(Prems,T,NewList).


% box: cheking is we have a sublist in the proof and the first line and last line
% in the box
box_cheking(First, Last, Already_checked):-
member([First|T],Already_checked),last(T,Last).

box_cheking(A,A,Already_checked):-member([A],Already_checked).

% handle premise
proof_validation(Prems, [_,Atom,premise], _):-member(Atom,Prems).

% handle assumption
```

[1] Michael Huth, Mark Ryan Logic in Computer ScienceLinks to an external site. Cambridge University Press 2004 (2nd edition) ISBN 0 521 54310X

```
proof_validation(Prems,[[_,_,assumption]|T],Already_checked):-
append(Already_checked,[[_,_,assumption]],NewList),
proof_Checking(Prems,T,NewList).

% handle copy (copy(X))
proof_validation(_,[_,A,copy(X)],Already_checked):-
member([X,A,_],Already_checked).

% handle and introduktion (andint(X,Y))
proof_validation(_,[_,and(A,H),andint(X,Y)],
Already_checked):-member([X,A,_],Already_checked),
member([Y,H,_],Already_checked).

% handle and elimination 1 (andel1(X))
proof_validation(_,[_,A,andel1(X)],Already_checked):-
member([X,and(A,_),_],Already_checked).

% handle and elimination 2 (andel2(X))
proof_validation(_,[_,H,andel2(X)],Already_checked):-
member([X,and(_,H),_],Already_checked).

% handle or introduktion 1 (orint1(X))
proof_validation(_,[_,or(A,_),orint1(X)], Already_checked):-
member([X,A,_],Already_checked).

% handle or introduktion 2 (orint2(X))
proof_validation(_,[_,or(_,H), orint2(X)], Already_checked):-
member([X,H,_],Already_checked).

% handle or elimination (orel(X,Y,U,V,W))
proof_validation(_,[_,A,orel(X,Y,U,V,W)],Already_checked):-
member([X, or(B,H),_],Already_checked),
box_cheking([Y,B,assumption],
[U,A,_],Already_checked),
box_cheking([V,H,assumption],[W,A,_],Already_checked).

% handle implikations elimination (impel(X,Y))
proof_validation(_,[_,B,impel(X,Y)],Already_checked):-
member([X,A,_],Already_checked),
member([Y, imp(A,B),_],Already_checked).

% handle implikations introduktion (impint(X,Y))
proof_validation(_,[_,imp(A,H),impint(X,Y)],Already_checked):-
box_cheking([X,A,assumption],[Y,H,_],Already_checked).

% handle negations elimination (negel(X,Y))
```

```
proof_validation(_,[_,cont,negel(X,Y)],Already_checked):-
member([X,A,_],Already_checked),
member([Y,neg(A),_],Already_checked).

% handle negations introduktion (negint(X,Y))
proof_validation(_,[_,neg(A),negint(X,Y)],Already_checked):-
box_cheking([X,A,assumption],[Y,cont,_],Already_checked).

% handle negations negations introduktion (negnegint(X))
proof_validation(_,[_,neg(neg(Atom)),negnegint(X)],Already_checked):-
member([X,Atom,_],Already_checked).

% handle negations negations elimination (negnegel(X))
proof_validation(_,[_,Atom,negnegel(X)],Already_checked):-
member([X,neg(neg(Atom)),_],Already_checked).

% handle contraduction (contel(X))
proof_validation(_,[_,_,contel(X)],Already_checked):-
member([X,cont,_],Already_checked).

% handle MT (mt(X,Y))
proof_validation(_,[_,_,mt(X,Y)],Already_checked):-
member([X,imp(_,B),_],Already_checked),
member([Y,neg(B),_],Already_checked).

% handle PBC (pbc(X,Y))
proof_validation(_,[_,A,pbc(X,Y)],Already_checked):- box_cheking([X,
neg(A),assumption],[Y,cont,_],Already_checked).

% handle LEM (lem)
proof_validation(_,[_,or(A,neg(A)),lem],_).
```