DD1351 Logic for Computer Scientists
Lab 2: Proof Checker with Prolog

Ayda Ghalkhanbaz
aydag@kth.se

Fatima Mohammad Ali
fatimama@kth.se

November 20, 2023

Introduction

The second lab in the course DD1351 focuses on constructing an algorithm to verify whether a proof given in natural deduction is correct or not. This report covers the implementation details of the program designed for this assignment. Furthermore, some run tests demonstrating the functionality of the program are presented in the later sections.

Proof Checker Algorithm

Natural deduction is a proof system that uses propositional logic rules to prove a sequence given its premises and goal. Each line of the proof includes a row reference (integer), a proposition, and the implemented logical operation on the row. The main approach of proving a sequence is to apply rules on relevant rows to achieve the original goal. If the given goal of a valid sequence is reached, the proof is valid, and in case of reaching a different goal, the proof is then invalid.

For this assignment, all logical rules presented in table 2 were used to construct the proof checking algorithm in Prolog. A line of proof could, for instance, look like "[3, and(p,q), andint(1,2)]", which shows how implementing and introduction on rows 1 and 2 results in the proposition and(p,q) as row 3.

The algorithm operates on a straightforward logical basis. The program is designed to examine a proof, top to bottom, by evaluating it line by line and return true or false based on its validation. The implementation is divided into several parts, each of which is comprehensively explained in the upcoming sections.

As a start, the program reads a file and stores the premises as a list (`Prems`), the goal as an atom (`Goal`), and the proof as a list (`Proof`). A predicate `valid_proof(Prems,Goal,Proof)` is then used to initiate the proof checking by calling the predicates `check_proof(Proof, Prems,[])` and `check_goal(Goal, Proof)`.

Goal Checking

This predicate simply checks if the determined goal is the same as the last row in the given proof. The `Check_goal`\2 predicate takes in two arguments: `Goal` and `Proof`. The last line of the proof can be accessed by recursively calling the `check_goal`\2 predicate until the base case is reached, and in case of the equality of the goal and the last line, the predicate returns `true`.

Proof Checking: Line by Line

The proof to be checked is a list consisting of lists representing lines in a proof and each line has to be valid for the proof to hold. To dissect each line and check for its validity, a predicate `check_proof`\3 is used. When `check_proof`\3 is first called in the `valid_proof`\3 predicate, it is sent the proof, the premises, and an empty list as arguments. The proof is iterated line by line by extracting its head recursively until the proof list is empty or an invalid line is discovered.

For each recursion, the head of the proof list (the current line) is extracted and sent as argument to the predicate that checks the validity of all logic rules, namely `check_rule(Prems, Line, ValidProofs)`. Said predicate will also receive the premises and the "empty list" (variable `ValidProofs`) as arguments. If a line is valid after the `check_rule`\3 call, it is appended to the `ValidProofs` list. For each recursion of `check_proof`\3, the empty list `ValidProofs` will hence build up and be used to store and retrieve all valid lines found so far.

Rule Checking

Each line of the proof includes a row number, a proposition, and an operation performed on rows and this rule syntax is considered while implementing the program. The main approach here is to search through all the defined rules to find the rule-pattern that matches the given line-pattern. In case of finding a match, the line is considered *valid*. If no rule-fitting pattern is found, the line then is considered *invalid*.

For instance, the *and introduction* rule has the general pattern: `[RowNr, and(X, Y), andint(Row1, Row2)]`, which corresponds to `[RowNr, X Y, i Row1, Row2]`. If a given line is `[3, and(p, q), andint(1, 2)]`, it will fit into the *and introduction* pattern and will unify the variables `X=p`, `Y=q`, `Row1=1`, and `Row2=2`. In contrast, the line `[3, or(p, q), orint(1, 2)]` would not fit into the *and introduction* pattern, which means that another rule should be checked for the line.

The `check_rule`\3 predicate has the rule-pattern directly as an argument to check whether a line fits a rule-pattern or not, e.g. for *and introduction* rule `check_rule(_,[_,and(X,Y),andint(Row1,Row2)], ValidProofs)`.

The first argument holds the premises (void for all rules except the premises and assumption check rule), the second argument is the given line, and the third argument is the list that holds all valid lines so far. If a line fits into the pattern in the middle argument, it is of said rule.

Next, its validity regarding terms and row numbers should be checked. This is done by utilizing the `ValidProofs` list. Depending on the rule, each term in the proposition should be found on a specific row. Using `[_, and(X, Y), andint(Row1, Row2)]` as an example, the variable `X` should be on `Row1`

and `Y` should be on `Row2`. If there exists a line in `ValidProofs` that contains a line with `Row1` and `X` as element `([Row1,X,_])`, and a line with `Row2` and `Y ([Row2,Y,_])`, then the given line is correct.

The membership to the `ValidProofs` list is done by using the built-in `members`\2 predicate (e.g. `member([Row1,X,_], ValidProofs)`). If the sought line cannot be found in `ValidProofs`, then the given line is incorrect and the entire proof-checking procedure is done and returned as `false`.

The rest of the rules that do not handle boxes are checked in a similar manner and explicitly presented in table 2.

Proof Checking: Boxes

A so-called "box" should be handled for all rules that require a subproof. In natural deduction, a box is opened when an assumption is made. Inside the box, a proof procedure, same as outside the box, is done. The conclusion of the box (resembling a goal) and the assumption (resembling a premise) should both be correct to close the box for a certain box-rule. For this assignment, a box is a line (same as the rest of lines), but the line itself consists of lists as elements. A box could look like the example in Appendix B. The box handling in the program is introduced when a line with an assumption pattern is found. What is desired here is to go through the box line by line, as previously, and check the validity of each line. The entire box is already given, which makes it possible to use the `check_proof`\3 predicate again for the box-checking. If the subproof (box) is entirely correct, the box is appended to the valid proof list.

The first element of a box should be an assumption, which is evaluated by the `check_rule`\3 predicate . The `check_rule`\3 receives the premises and valid proofs so far as arguments, as well as the current line to check. If the line is a list that includes lists as elements and the head (first line) of the line is an assumption, then the entire line is considered a box (subproof). What should be done now is to check whether the rest of the box is a valid proof. The subproof checking is done by sending the tail of the box as the "Proof" argument to `check_proof`\3. The `check_proof`\3 will also receive the `ValidProofs` list with the assumption line as a head. The `check_proof`\3 predicate will check all lines within the subproof and append each one to the valid proofs. If all are valid, the entire subproof will be appended (as one line) to `ValidProofs` and the subproof checking is exited to continue the outer proof-checking. That is how a valid box is checked.

The rules that utilize subproofs should appear after a box has been appended to the valid proofs list. For implication introduction for instance, there needs to be a valid subproof in the main proof for the rule to be applied. As an example, the check_rule predicate for implication introduction is implemented as follows:

```
check_rule(_, [_, imp(X, Y), impint(Row1,Row2)], ValidProofs):-
                member(Box, ValidProofs),
                member([Row1, X, assumption], Box),
                last(Box, Last), Last = [Row2, Y, _].
```

The `check_rule`\3 directly checks whether a given line fits the implication introduction pattern and if it does, the body of the rule is performed. The built-in `members`\2 predicate uses a non-initiated variable "Box" as a placeholder for a line that is in `ValidProofs`. In other words, `members`\2 "iterates" the list of valid proofs and unifies the current element to "Box". If the current element does not satisfy the following rules/predicates, the `members`\2 predicate will backtrack to explore other lines in `ValidProofs`. For the `check_rule`\3 predicate above, the first member predicate (`member(Box, ValidProofs)`) will unify the first line of `ValidProofs` to the variable "Box". Followingly, the second `members`\2 predicate (`member([Row1, X, assumption], Box)`) will check whether the desired assumption line is within the currently unified "Box". If there is no assumption line, `member`\2 will backtrack and explore another member. If there is an assumption line (the desired one), then a valid box with lists as elements has been found. What is required now is to find the desired conclusion of the box. To extract the last line of "Box", the built-in `last`\2 predicate is used and the conclusion-line is unified with the variable "Last" (`last(Box, Last)`). Lastly, the conclusion line is checked whether it is equal to the *implication introduction* conclusion (checks the correct row number and term). If every step is successfully executed, the rule is valid for the given proof-line and the next line of the main proof is checked as usual.

Other subproof-based rules are checked similarly to the implication introduction rule and can be found in Appendix A: Prolog Code.

List of Predicates

The main predicates are presented in table 1 showing in which scenarios they are true or false. Additionally, all helper predicates for checking the validation of logical rules in natural deduction are presented in table 2.

Table 1: A list of the main predicates and their success and failure criteria

Main Predicates		
Predicate	**Success Criteria**	**Failure Criteria**
`valid_proof`\3	both `check_proof`\3 and `check_goal`\2 are true; then prints out '*Yes*'	one or neither of the `check_proof`\3 and `check_goal`\2 are false; then prints out '*No*'
`check_proof`\3	all of the lines in a proof are valid	one or more lines in a proof are invalid
`check_goal`\2	if the goal is the same as the last line of the given proof	if the goal doesn't match the last line of the proof
`check_rule`\3	if the rules have been used properly and applied to the correct lines of a proof	if one or more rules have not been employed properly

Table 2: A list of helper predicates for checking the validation of the rules with arbitary statements X, Y and Z

Helper Predicates	
Predicate	**Success Criteria**
`check_rule(_,Box,_)`	True if all lines of the subproof in a box are valid
`check_rule(_,assumption,_)`	True if an assumption is within a box.
`check_rule(_,premise,_)`	True if the premise is already a member in `ValidProofs`
`check_rule(_,andint,_)`	True if both X and Y are separately already a member of `ValidProofs`
`check_rule(_,andel1,_)`	True if there a conjunction between X and any statement is a member of `ValidProofs`
`check_rule(_,andel2,_)`	True if there a conjunction between any statement and Y is a member of `ValidProofs`
`check_rule(_,orint1,_)`	True if X is already a member of `ValidProofs`
`check_rule(_,orint2,_)`	True if Y is already a member of `ValidProofs`
`check_rule(_,orel,_)`	True if an disjunction between X and Y in row A is a member of `ValidProofs`, two boxes (subproofs) are introduced, the assumption on row B is a member of the first box, the last row of the first box is the same as the conclusion in row C, the assumption on row D is a member of the second box and the last row of the second box is the same as the conclusion in row E.

Helper Predicates (continued)

Predicate	Success Criteria
`check_rule(_,impint,_)`	True if a box is started with assuming X and reaching Y
`check_rule(_,impel,_)`	True if both X and an implication between X and Y are already a member of `ValidProofs`
`check_rule(_,negint,_)`	True if a box is started with assuming X and reaching a contradiction
`check_rule(_,negel,_)`	True if both X and its negation are members of `ValidProofs`
`check_rule(_,negnegint,_)`	True if X is already a member of `ValidProofs`
`check_rule(_,negnegel,_)`	True if the double negation of X is already a member of `ValidProofs`
`check_rule(_,contel,_)`	True if a contradiction is a member of `ValidProofs`
`check_rule(_,copy,_)`	True if X is already a member of `ValidProofs`
`check_rule(_,mt,_)`	True if both an implication between X and Y and negation of Y are already a member of `ValidProofs`
`check_rule(_,lem,_)`	Can be true without any special pre-defined condition
`check_rule(_,pbc,_)`	True if a box is started with assuming the negation of X and reaching a contradiction

Appendix A: Prolog Code

```
% reading and dividing a file
verify(FileName):- see(FileName), % reads a file
                   read(Prems),   % first row is premise
                   read(Goal),    % second row is the goal
                   read(Proof),   % third row is proof
                   seen,          % close the file
                   valid_proof(Prems,Goal,Proof).  % proof checking

/*************** Valid Proof **************/
% print yes if proof is valid
valid_proof(Prems, Goal, Proof) :-
                check_proof(Proof, Prems,[]),
                check_goal(Goal, Proof),!,
                write("Yes").

% print no if proof is invalid
valid_proof(Prems,Goal,Proof):-
                ((not(check_proof(Proof,Prems,[])));
                (not(check_goal(Goal,Proof)))),!,
                write("No"),
                fail.

/************** Proof checking *************/
check_proof([], _, _). %base case, Proof list is empty
check_proof([Line|RestProof], Prems, ValidProofs) :-
        check_rule(Prems, Line, ValidProofs),
        append([Line], ValidProofs, Result),
        check_proof(RestProof, Prems, Result).

/************** Goal checking *************/
% base case
check_goal(Goal,[[_,LastRow,_]|[]]):- Goal = LastRow.
% finding the last row recursively
check_goal(Goal,[_|Tail]) :- check_goal(Goal,Tail).

/************** Box checking *************/
check_rule(Prems,[[RowNumber, X, assumption]|Box], ValidProofs):-
    check_proof(Box, Prems, [[RowNumber, X, assumption]|ValidProofs]).

/************** Rule checking *************/
```

```
% premise
check_rule(Prems,[_,Premise,premise],_):-
                member(Premise,Prems).

% andint
check_rule(_,[_,and(X,Y),andint(Row1,Row2)], ValidProofs):-
                member([Row1,X,_], ValidProofs),
                member([Row2,Y,_], ValidProofs).

% andel 1
check_rule(_,[_,X,andel1(RowNumber)], ValidProofs):-
                member([RowNumber,and(X,_),_], ValidProofs).

%andel 2
check_rule(_,[_,Y,andel2(RowNumber)], ValidProofs):-
                member([RowNumber,and(_,Y),_], ValidProofs).

% orint 1
check_rule(_,[_,or(X,_),orint1(RowNumber)], ValidProofs):-
                member([RowNumber,X,_], ValidProofs).

% orint 2
check_rule(_,[_,or(_,Y),orint2(RowNumber)], ValidProofs):-
                member([RowNumber,Y,_], ValidProofs).

% impel
check_rule(_,[_,Y,impel(Row1,Row2)], ValidProofs):-
                member([Row1,X,_], ValidProofs),
                member([Row2,imp(X,Y),_], ValidProofs).
% negel
check_rule(_,[_,cont, negel(Row1,Row2)],ValidProofs):-
                member([Row1, X,_], ValidProofs),
                member([Row2, neg(X),_], ValidProofs).

% copy
check_rule(_,[_,X, copy(RowNumber)], ValidProofs):-
                member([RowNumber,X,_], ValidProofs).

% negnegel
check_rule(_,[_,X,negnegel(RowNumber)], ValidProofs):-
                member([RowNumber,neg(neg(X)),_], ValidProofs).

% negnegint
check_rule(_,[_,neg(neg(X)),negnegint(RowNumber)], ValidProofs):-
```

```
                member([RowNumber,X,_], ValidProofs).

% contel
check_rule(_,[_,_, contel(RowNumber)], ValidProofs):-
                member([RowNumber, cont, _], ValidProofs).

% MT
check_rule(_,[_, neg(X), mt(Row1,Row2)], ValidProofs):-
                member([Row1, imp(X,Y),_], ValidProofs),
                member([Row2, neg(Y),_], ValidProofs).

% LEM (it is valid without any conditions)
check_rule(_,[_,or(X,neg(X)), lem], _).

% orel (box)
check_rule(_,[_,Z,orel(Row1,Row2,Row3,Row4,Row5)], ValidProofs):-
                member([Row1,or(X,Y),_], ValidProofs),
                member(BoxOne, ValidProofs), member(BoxTwo, ValidProofs),
                member([Row2,X,assumption], BoxOne),
                last(BoxOne,Conclusion1),Conclusion1 = [Row3,Z,_],
                member([Row4,Y,assumption], BoxTwo),
                last(BoxTwo,Conclusion2),Conclusion2 = [Row5,Z,_].

% impint(box)
check_rule(_, [_, imp(X, Y), impint(Row1,Row2)], ValidProofs):-
                member(Box, ValidProofs),
                member([Row1, X, assumption], Box),
                last(Box, Last), Last = [Row2, Y, _].

% negint(box)
check_rule(_,[_, neg(X), negint(Row1,Row2)], ValidProofs) :-
                member(Box, ValidProofs),
                member([Row1,X,assumption], Box),
                last(Box, Last), Last = [Row2, cont, _].

% PBC(box)
check_rule(_,[_, X, pbc(Row1,Row2)], ValidProofs) :-
                member(Box, ValidProofs),
                member([Row1,neg(X),assumption], Box),
                last(Box, Last), Last = [Row2, cont, _].
```

Appendix B: Example Tests

Example of a valid proof

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

Run of the valid proof

```
?- [lab2].
true.

?- verify('valid.txt').
Yes
true.
```

Example of an invalid proof

```
[imp(p, q), p].
q.
[
[1, imp(p,q), premise],
[2, q, assumption]
].
```

Run of the invalid proof

```
?- [lab2].
true.

?- verify('invalid.txt').
No
false.
```

Run of all test proofs given in the assignment

```
1 ?- ['run_all_tests.pl'].
true.

2 ?- run_all_tests('lab2.pl').
valid01.txt Yes passed
valid02.txt Yes passed
valid03.txt Yes passed
valid04.txt Yes passed
valid05.txt Yes passed
valid06.txt Yes passed
valid07.txt Yes passed
valid08.txt Yes passed
valid09.txt Yes passed
valid10.txt Yes passed
valid11.txt Yes passed
valid12.txt Yes passed
valid13.txt Yes passed
valid14.txt Yes passed
valid15.txt Yes passed
valid16.txt Yes passed
valid17.txt Yes passed
valid18.txt Yes passed
valid19.txt Yes passed
valid20.txt Yes passed
valid21.txt Yes passed
valid22.txt Yes passed
valid23.txt Yes passed
invalid01.txt No passed
invalid02.txt No passed
invalid03.txt No passed
invalid04.txt No passed
invalid05.txt No passed
invalid06.txt No passed
invalid07.txt No passed
invalid08.txt No passed
invalid09.txt No passed
invalid10.txt No passed
invalid11.txt No passed
invalid12.txt No passed
invalid13.txt No passed
invalid14.txt No passed
invalid15.txt No passed
```

```
invalid16.txt No passed
invalid17.txt No passed
invalid18.txt No passed
invalid19.txt No passed
invalid20.txt No passed
invalid21.txt No passed
invalid22.txt No passed
invalid23.txt No passed
invalid24.txt No passed
invalid25.txt No passed
invalid26.txt No passed
invalid27.txt No passed
invalid28.txt No passed
invalid29.txt No passed
invalid30.txt No passed
invalid31.txt No passed
invalid32.txt No passed
```
