Laboration 2: Natural Proof-check using Prolog

Emil Göransson

August 21, 2023

Introduction

This report will cover one of the many ways you can check natural proofs using the programming language prolog. Given a proof, and a goal the program will answer ”true” if the proof is valid, or ”false” if the proof is invalid.

A valid proof

A proof is deemed valid if the following conditions are all fulfilled:

1. The last line of the proof contains the goal that we wish to prove.
2. That the format of the proof is correct.
3. That when referencing a previous statement line of proof exists.
4. That the formulas used are used according to their rules.

Utilizing recursion

The algorithm uses a step by step process so assess that the conditions stated in A valid proof section are followed. The program is called using the function

```
valid_proof(Premise, Goal, Proof)
```

The function is the first step to the algoritm. It initializes the algorithm by first checking that the last line of the proof contains the goal. After the first step is passed the function calls valid_proof/5.

```
valid_proof(Premise, Goal, Proof, ProofUntilNow, AssumptProof)
```

Premise, Goal and Proof are all the same as in valid_proof/3. ProofUntilNow keeps track of the currently known information in the proof. The reason behind using ProofUntilNow instead of Proof is to avoid being able to access a statement of proof that has not been proven yet.

The algoritm works the the following way. The proof is read from top from bottom reading one line at a time. There are 19 different valid_proof/5 functions. The correct function is selected based on pattern matching. The general idea is to check either that the premise matches the current proof by using the built in function member/2. Or checking the current trail of proof up until this point in the proof.

If the member function is true, the current line of proof is appended using the standard library function append/2 to the variable ProofUntilNow.

After the row has been appended valid_proof is called again to check the next line of proof, now with the updated ProofUntilNow context.

```
valid_proof(Premise, Goal, Rest, NewProofUntilNow, AssumptProof).
```

When every line in the proof has been checked the program will return true which means that the proof is correct.

Handling assumptions

An important part of natural deduction is the the possibility of making assumptions. The tricky part of this is that when leaving a box, later lines of the proof should not be able to access the things ”proven” inside the boxes. To solve this issue a separate list under the variable name AssumptProof is used to separate the ”real proof” and the assumption-proofs. The implementation I used to validate any assumption-proofs is to treat each of the assumption-proofs as the same as a non-assumption proof, and check their validity using simple recursion.

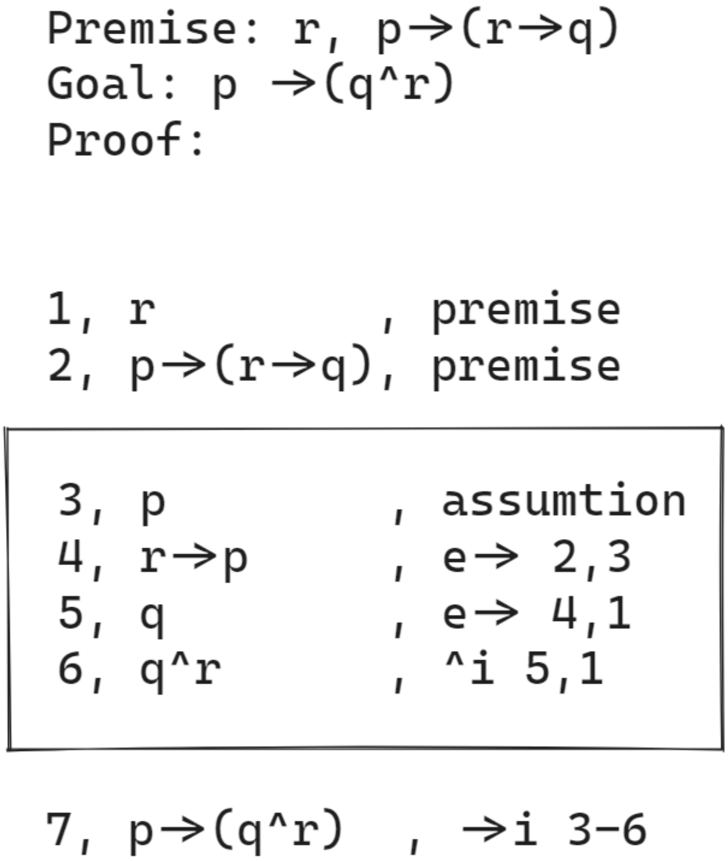
Since every assumption always starts with a indent in the form of a separate list we can by using that and the assumption tag identify the start of a box/assumption.

Predicate/#arguments	**True**	**False**
verify/1	When the file is supplied in the correct format	If it can't find the file or it isn't in the correct format
goal_is_last/3	If the goal is at the last line of the proof	If the goal is not at the last line of the proof
valid_proof/3	If goal_is_last/3 is true & valid_proof/5 is true, and if the proof is supplied in the correct format for every line in the proof (can pattern-match)	If goal_is_last/3 is false OR valid_proof/5 is false
valid_proof/5	If it can "get" to the end of Proof (Proof = `[]`)	Can't find any pattern which matches OR member/2 fails
member/2	If argument_1 exists inside the list argument_2	Argument_1 does not exist in the list argument_2

Table 1: Explanation of Predicates

Table of predicates

Appendix



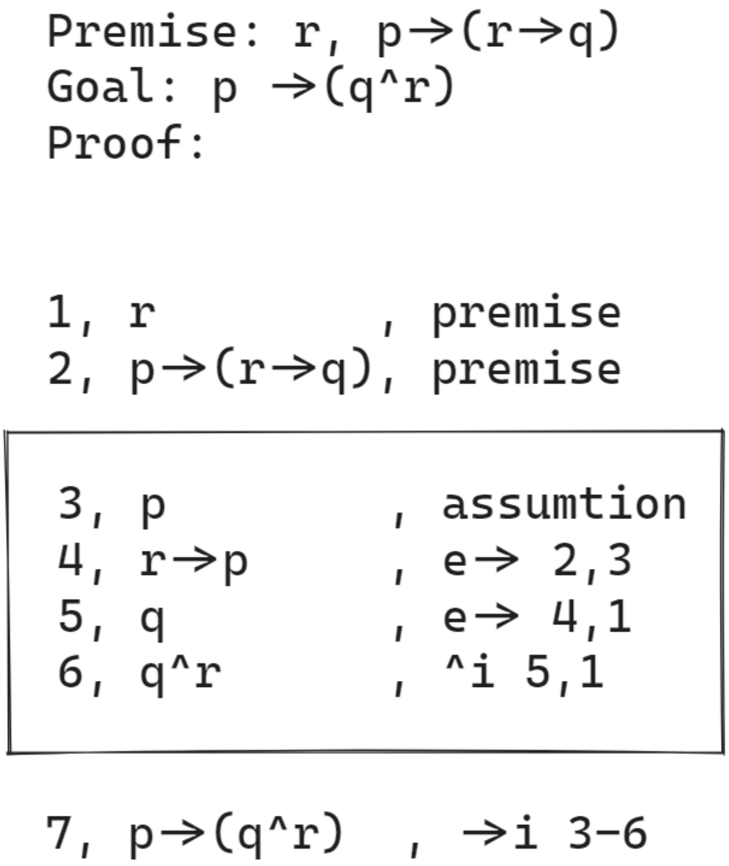
```
Premise: r, p->(r->q)
Goal: p ->(q^r)
Proof:


1, r          , premise
2, p->(r->q), premise

 3, p          , assumtion
 4, r->p       , e-> 2,3
 5, q          , e-> 4,1
 6, q^r        , ^i 5,1

7, p->(q^r)   , ->i 3-6
```

Figure 1: Non trivial valid proof



```
Premise: r, p->(r->q)
Goal: p ->(q^r)
Proof:

1, r         , premise
2, p->(r->q), premise

  3, p         , assumtion
  4, r->p      , e-> 2,3
  5, q         , e-> 4,1
  6, q^r       , ^i 5,1

7, p->(q^r)   , ->i 3-6
```

Figure 2: Non trivial invalid proof

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

%checks if goal exists at end of list.
goal_is_last(Goal, [[_,Goal,_]|[]]).
goal_is_last(Goal, [_|Tail]):-
    goal_is_last(Goal, Tail).

valid_proof(Premise, Goal, Proof):-
    goal_is_last(Goal, Proof),
    valid_proof(Premise, Goal, Proof, [], []).

valid_proof(_,_,[],_,_).

%premise
valid_proof(Premise, Goal, [[Row, CurProof, premise]|Rest],
↪  ProofUntilNow, AssumptProof):-
    member(CurProof, Premise)  ,
    append(ProofUntilNow,[[Row, CurProof, premise]], NewProofUntilNow),
    valid_proof(Premise, Goal, Rest, NewProofUntilNow, AssumptProof).

%imp, impel, implication elimination
valid_proof(Premise, Goal, [[Row, CurProof, impel(Row1,Row2)]|Rest],
↪  ProofUntilNow, AssumptProof):-
    member([Row1, AltProof,_], ProofUntilNow)  ,
    member([Row2, imp(AltProof,CurProof),_], ProofUntilNow)  ,
    append(ProofUntilNow, [[Row, CurProof, impel(Row1, Row2)]],
    ↪  NewProofUntilNow),
    valid_proof(Premise, Goal, Rest, NewProofUntilNow, AssumptProof).

%andel1
valid_proof(Premise, Goal, [[Row, CurProof, andel1(Row1)]|Rest],
↪  ProofUntilNow, AssumptProof):-
    member([Row1, and(CurProof,_),_], ProofUntilNow)  ,
    append(ProofUntilNow, [[Row, CurProof, andel1(Row1)]],
    ↪  NewProofUntilNow),
    valid_proof(Premise, Goal, Rest, NewProofUntilNow, AssumptProof).

%andel2
valid_proof(Premise, Goal, [[Row, CurProof, andel2(Row1)]|Rest],
↪  ProofUntilNow, AssumptProof):-
    member([Row1, and(_,CurProof),_], ProofUntilNow)  ,
    append(ProofUntilNow, [[Row,CurProof,andel2(Row1)]],
    ↪  NewProofUntilNow),
    valid_proof(Premise,Goal,Rest,NewProofUntilNow, AssumptProof).

%negel
valid_proof(Premise,Goal, [[Row, CurProof, negel(Row1, Row2)]|Rest],
↪  ProofUntilNow, AssumptProof):-
    member([Row1, AltProof, _], ProofUntilNow)  ,
    member([Row2, neg(AltProof),_], ProofUntilNow)  ,
    append(ProofUntilNow, [[Row, CurProof, negel(Row1, Row2)]],
    ↪  NewProofUntilNow),
    valid_proof(Premise, Goal, Rest, NewProofUntilNow, AssumptProof).
%contel
```

```
valid_proof(Premise, Goal, [[Row, CurProof, contel(Row1)]|Rest],
↪  ProofUntilNow, AssumptProof):-
    member([Row1, cont, _], ProofUntilNow)  ,
    append(ProofUntilNow, [[Row, CurProof, contel(Row1)]],
      ↪  NewProofUntilNow),
    valid_proof(Premise,Goal,Rest,NewProofUntilNow, AssumptProof).
%negnegel
valid_proof(Premise,Goal, [[Row, CurProof, negnegel(Row1)]|Rest],
↪  ProofUntilNow, AssumptProof):-
    member([Row1, neg(neg(CurProof)),_], ProofUntilNow)  ,
    append(ProofUntilNow, [[Row, CurProof, negnegel(Row1)]],
      ↪  NewProofUntilNow),
    valid_proof(Premise, Goal, Rest, NewProofUntilNow, AssumptProof).
%andint
valid_proof(Premise,Goal,[[Row, and(CurProof1, CurProof2),
↪  andint(Row1,Row2)]|Rest], ProofUntilNow, AssumptProof):-
    member([Row1, CurProof1, _], ProofUntilNow)  ,
    member([Row2, CurProof2, _], ProofUntilNow)  ,
    append(ProofUntilNow, [[Row, and(CurProof1, CurProof2),
      ↪  andint(Row1,Row2)]], NewProofUntilNow),
    valid_proof(Premise, Goal, Rest, NewProofUntilNow, AssumptProof).

%orint1
valid_proof(Premise,Goal,[[Row, or(CurProof1, CurProof2),
↪  orint1(Row1)]|Rest], ProofUntilNow, AssumptProof):-
    member([Row1, CurProof1, _], ProofUntilNow)  ,
    append(ProofUntilNow, [[Row, or(CurProof1, CurProof2),
      ↪  orint1(Row1)]], NewProofUntilNow),
    valid_proof(Premise,Goal, Rest,NewProofUntilNow, AssumptProof).

%orint2
valid_proof(Premise,Goal,[[Row, or(CurProof1, CurProof2),
↪  orint2(Row1)]|Rest], ProofUntilNow, AssumptProof):-
    member([Row1, CurProof2, _], ProofUntilNow)  ,
    append(ProofUntilNow, [[Row, or(CurProof2, CurProof1),
      ↪  orint2(Row1)]], NewProofUntilNow),
    valid_proof(Premise,Goal, Rest,NewProofUntilNow, AssumptProof).

%LEM
valid_proof(Premise, Goal, [[Row, or(CurProof, neg(CurProof)),
↪  lem]|Rest], ProofUntilNow, AssumptProof):-
    append(ProofUntilNow, [[Row, or(CurProof, neg(CurProof), lem)]],
      ↪  NewProofUntilNow),
    valid_proof(Premise, Goal, Rest, NewProofUntilNow, AssumptProof).

%negnegint
valid_proof(Premise,Goal, [[Row, neg(neg(CurProof)), negnegint(Row1)]|Rest], ProofUntilNow, AssumptProof):-
    member([Row1, CurProof, _], ProofUntilNow)   ,
    append(ProofUntilNow, [[Row, neg(neg(CurProof)), negnegint(Row1)]], NewProofUntilNow),
    valid_proof(Premise, Goal, Rest, NewProofUntilNow, AssumptProof).

%MT
valid_proof(Premise, Goal, [[Row, neg(CurProof), mt(Row1, Row2)]|Rest], ProofUntilNow, AssumptProof):-
    member([Row1, imp(CurProof, AltProof),_], ProofUntilNow)   ,
    member([Row2, neg(AltProof), _], ProofUntilNow)   ,
    append(ProofUntilNow, [[Row, neg(CurProof), mt(Row1, Row2)]], NewProofUntilNow),
    valid_proof(Premise, Goal, Rest, NewProofUntilNow, AssumptProof).

%Copy
valid_proof(Premise, Goal, [[Row, CurProof, copy(Row1)]|Rest], ProofUntilNow, AssumptProof):-
    member([Row1, CurProof, _], ProofUntilNow) ,
    append(ProofUntilNow, [Row, CurProof, copy(Row1)], NewProofUntilNow),
    valid_proof(Premise, Goal, Rest, NewProofUntilNow, AssumptProof).

%Assumption

valid_proof(Premise, Goal, [[[Row, CurProof, assumption]|RestBox]|Rest2], ProofUntilNow, AssumptProof):-
    append(ProofUntilNow, [[Row, CurProof, assumption]], NewProofUntilNow),
    append(AssumptProof, [[Row, CurProof, assumption]|RestBox],NewAssumptProof),
    %here we check so that the boxproof follows all the rules using reqursion,
    %we treat it like a normal proof besides the fact that goal_is_last is ignored/3.

    valid_proof(Premise, [], RestBox, NewProofUntilNow, []),
    valid_proof(Premise, Goal, Rest2, ProofUntilNow, NewAssumptProof).

```

```
%impint
valid_proof(Premise, Goal, [[Row, imp(CurProof1, CurProof2), impint(Row1, Row2)]|Rest], ProofUntilNow, AssumptProof):-
    member([Row1, CurProof1, assumption], AssumptProof) ,
    member([Row2, CurProof2, _], AssumptProof) ,
    append(ProofUntilNow, [[Row, imp(CurProof1, CurProof2), impint(Row1, Row2)]], NewProofUntilNow),
    valid_proof(Premise, Goal, Rest, NewProofUntilNow, AssumptProof).
%negint
valid_proof(Premise, Goal, [[Row, neg(CurProof), negint(Row1, Row2)]|Rest], ProofUntilNow, AssumptProof):-
    member([Row1, CurProof, assumption], AssumptProof) ,
    member([Row2, cont, _], AssumptProof) ,
    append(ProofUntilNow, [[Row, neg(CurProof), negint(Row1, Row2)]], NewProofUntilNow),
    valid_proof(Premise, Goal, Rest, NewProofUntilNow, AssumptProof).

%orel
valid_proof(Premise, Goal, [[Row, CurProof, orel(Row1, Row2, Row3, Row4, Row5)]|Rest], ProofUntilNow, AssumptProof):-
    member([Row1, or(AltProof1, AltProof2), _], ProofUntilNow) ,
    member([Row2, AltProof1, assumption], AssumptProof) ,
    member([Row3, CurProof, _], AssumptProof) ,
    member([Row4, AltProof2, assumption], AssumptProof) ,
    member([Row5, CurProof, _], AssumptProof) ,
    append(ProofUntilNow, [[Row, CurProof, orel(Row1, Row2, Row3, Row4, Row5)]], NewProofUntilNow),
    valid_proof(Premise, Goal, Rest, NewProofUntilNow, AssumptProof).
%pbc
valid_proof(Premise, Goal, [[Row, CurProof, pbc(Row1, Row2)]|Rest], ProofUntilNow, AssumptProof):-
    member([Row1, neg(CurProof), assumption], AssumptProof) ,
    member([Row2, cont, _], AssumptProof) ,
    append(ProofUntilNow, [[Row, CurProof, pbc(Row1, Row2)]], NewProofUntilNow),
    valid_proof(Premise, Goal, Rest, NewProofUntilNow, AssumptProof).

```