Lab 2

Alexander Wigren & Pontus Hector

Fall 2022

Introduction

This report covers the second lab assignment in the course DD1351, given during the fall semester of 2022. The assignment entailed building a proof-checking program for natural deduction in Prolog. In the following sections we present how the algorithm works and a list with each predicate and their success criteria. The complete program code and example proofs may be found in the appendixes.

The proof checking algorithm

In this section we present the general workings of the Prolog program. Proofs which may be controlled by the program are structured as Prolog lists. The lists have a general form of one large list which contain sublists which represent the usual lines used when dealing with natural deduction on paper (The appendices may be viewed to see the exact structure). The algorithm employed may wander through the proof in several 'stages' which are presented here.

Bootstrapping

The program starts with a simple bootstrapping predicate `verify` which simply reads the input file and separates the content into the useful terms `Prems, Goal` and `Proof`. It then calls the predicate `valid_proof` which starts the proof checking in earnest.

Goal checking

The predicate valid_proof contains three clauses: if the first is fulfilled the proof is correct and 'Yes' is printed in the terminal. If any of the other two clauses succeed the proof is false and 'No' is printed in the terminal.

The first clause simply checks if the goal and conclusion of the are the same via `goal_achieved`, and then calls the predicate `proof_determined`

which works its way step by step through each line of the proof. The other two clauses will succeed either if the goal and conclusion do not match, or if any line of the proof is invalid.

Proof checking line by line

The predicate `proof_determined` contains three clauses. A base clause which succeeds and terminates recursion if it receives an empty list of lines, since that will only happen once the end of the given proof has been reached. The other two clauses will recursively walk through the proof line by line. The second is a special clause which is used to check boxes, which is explained in the next subsection. The third clause reads a line and verifies that it is correct by calling the predicate `valid_use_of_rule`, a large collection of clauses which check that any line has applied its associated rule correctly. If a line is correct, it is then added to a list of checked lines.

This list of checked lines is essential for the function of the program. It is first populated with any premises that are declared at the top. These are handled in a clause which simply checks if the associated rule of the line is 'premise' and makes sure the expression is a member of the premise list, it then adds it to the list of checked lines. This list is then used for actual deduction rules such as and-elimination or implication-introduction, which themselves gradually fill the list up.

The checking of rules is rather simple in themselves. Take for example the and-elimination-1 rule. The only clause which matches this rule in the `valid_use_of_rule` predicate takes the expression and checks the line number specified. If the expression is present inside of an `and(X,Y)` term on a line that has been checked, the rule has been successfully applied! The line may then be added to the list of checked lines.

Proof checking with boxes

Proof checking with boxes is somewhat more complicated since there are things to check when boxes are involved. For example, a line may not refer to a line in a box that has been previously closed. Certain proof rules also explicitly require some assumption to be made at the top of a box and a certain conclusion to have been reached at the bottom of the box for the rule to be successfully applied.

Box handling is initiated with the second clause of the `proof_determined` predicate. This predicate is triggered whenever a line containing an assumption followed by a list is occurred since an assumption indicates the start of a box. The box is then evaluated recursively through the predicate `box_determined` in a similar fashion as the predicate `proof_determined`, from the first line to the last line, until the end of the box is reached.

The `box_determined` predicate contains three clauses as well. The first

clause succeeds and terminates recursion if it receives an empty list of lines, which will only occur whenever the end of the box is reached. The second clause is triggered if a box inside of the current box is occurred, in which case the found box is recursively evaluated through the predicate `box_determined` from the start to the end of that box, before moving on to the evaluation of the rest of the current box. The third clause reads a line and verifies that it is correct by calling the predicate `valid_use_of_rule`. If all of the lines have been evaluated and confirmed, the box is saved as a list together with the previously checked proofs and since the evaluated box is stored as a list, referral to a row inside the box is not possible by referring to our list of checked proofs.

Some of the rules defined through the basic rules of induction require references to the first and last lines of boxes to be valid. To refer to a box the predicate `valid_box` is used. This predicate searches through the list of checked proofs for a list with the given line number and expression through the predicate `get_box`. If the targeted box could be found the box is evaluated in regard to whether the given start line and given end line exists in the box on their given line numbers.

List of predicates and their success criteria

`Predicate` — *Success criteria*

`valid_proof\3` — The given goal is equivalent to the last row and all the lines of the proof have been evaluated

`valid_proof\3` — The given goal is not equivalent to the last row

`valid_proof\3` — All the lines of the proof could not be validated

`goal_achieved\2` — The given goal is equivalent to the last row

`goal_achieved\2` — Line(s) exist under the current row

`proof_determined\3` — All the lines of the proof have been evaluated

`proof_determined\3` — All the lines of the occurred box have been evaluated and the rest of the proof is evaluated

`proof_determined\3` The given line is validated and the rest of the proof is evaluated

`box_determined\3` — All the lines of the box have been evaluated

`box_determined\3` — All the lines of the occurred box have been evaluated and the rest of the current box is evaluated

`box_determined\3` The given line is validated and the rest of the proof is evaluated

`valid_box\3` — The referred box is found and the given start line and end line could be found inside that box

`get_box\3` — The referred box is found

`valid_use_of_rule\3` — The given line is confirmed by referral to the given premises, previously validated lines, or previously validated boxes according to the basic rules of natural induction

`member\2` — The given object exists in the given list

Appendix A - Prolog Code

```
% XXXXX Extracting data from inputfile XXXXX
verify(InputFileName) :-
    see(InputFileName),
    read(Prems),
    read(Goal),
    read(Proof),
    seen,
    valid_proof(Prems, Goal, Proof).

% XXXXX Is the proof valid or not? XXXXX

% Valid
valid_proof(Prems, Goal, Proof) :-
    goal_achieved(Goal, Proof),
    proof_determined(Prems, Proof, []),
    write("Yes").

% Invalid
valid_proof(_, Goal, Proof) :-
    \+goal_achieved(Goal, Proof),!,
    write("No"),
    fail.

valid_proof(Prems, _, Proof) :-
    \+proof_determined(Prems, Proof, []),
    write("No"),
    fail.

% XXXXX Determination of the goal XXXXX

% Base case - Is the goal achieved?
goal_achieved(Goal, [[_, Last, _]|[]]) :-
    Goal = Last.

% Determination of the last line recursively
goal_achieved(Goal, [_|T]) :-
    goal_achieved(Goal, T).


% XXXXX Determination of the proof XXXXX
```

```
% Base case
proof_determined(_, [], _).

% Box
proof_determined(Prems, [[[LineNo, Expression, 'assumption']|TBOX]|T],
CheckedProofs) :-
    box_determined(Prems, TBOX, [[LineNo, Expression,
    'assumption']|CheckedProofs]),
    proof_determined(Prems, T, [[[LineNo, Expression,
    'assumption']|TBOX]|CheckedProofs]).

% Line
proof_determined(Prems, [H|T], CheckedProofs) :-
    valid_use_of_rule(Prems, H, CheckedProofs),
    proof_determined(Prems, T, [H|CheckedProofs]).

% XXXXX Boxhandling XXXXX

% Base case
box_determined(_, [], _).

% Box
box_determined(Prems, [[[LineNo, Expression,
'assumption']|TBOX]|T], CheckedProofs) :-
    box_determined(Prems, TBOX, [[LineNo, Expression,
    'assumption']|CheckedProofs]),
    box_determined(Prems, T, [[[LineNo, Expression,
    'assumption']|TBOX]|CheckedProofs]).

% Line
box_determined(Prems, [H|T], CheckedProofs) :-
     valid_use_of_rule(Prems, H, CheckedProofs),
     box_determined(Prems, T, [H|CheckedProofs]).

% Referral to a box
valid_box(LineNo, LineNo2, CheckedProofs, Expression, Expression2) :-
    get_box(LineNo, CheckedProofs, Box),
    member([LineNo, Expression, _], Box),
    member([LineNo2, Expression2, _], Box).

% Find the targeted box (if it exists)
get_box(LineNo, [H|_], H) :-
    member([LineNo, _, _], H).
get_box(LineNo, [_|T], Box) :-
```

```
    get_box(LineNo, T, Box).

% XXXXX Confirming the usage of rules XXXXX

% premise
valid_use_of_rule(Prems, [_, Expression, 'premise'], _) :-
    member(Expression, Prems).

% copy(x)
valid_use_of_rule(_,[_, Expression, copy(LineNo)], CheckedProofs) :-
    member([LineNo, Expression, _], CheckedProofs).

% andint(x,y)
valid_use_of_rule(_,[_, and(X, Y), andint(LineNo, LineNo2)],
CheckedProofs) :-
    member([LineNo, X, _], CheckedProofs),
    member([LineNo2, Y, _], CheckedProofs).

% andel1(x)
valid_use_of_rule(_,[_, Expression, andel1(LineNo)], CheckedProofs) :-
    member([LineNo, and(Expression, _), _], CheckedProofs).

% andel2(x)
valid_use_of_rule(_,[_, Expression, andel2(LineNo)], CheckedProofs) :-
    member([LineNo, and(_, Expression), _], CheckedProofs).

% orint1(x)
valid_use_of_rule(_,[_, or(X, _), orint1(LineNo)], CheckedProofs) :-
    member([LineNo, X, _], CheckedProofs).

% orint2(x)
valid_use_of_rule(_,[_, or(_, Y), orint2(LineNo)], CheckedProofs) :-
    member([LineNo, Y, _], CheckedProofs).

% orel(x,y,z,u,v)
valid_use_of_rule(_,[_, Expression, orel(LineNo, LineNo2, LineNo3,
LineNo4, LineNo5)], CheckedProofs):-
    member([LineNo, or(Expression2, Expression3), _], CheckedProofs),
    valid_box(LineNo2, LineNo3, CheckedProofs, Expression2, Expression),
    valid_box(LineNo4, LineNo5, CheckedProofs, Expression3, Expression).

% impint(x,y)
valid_use_of_rule(_,[_, imp(Expression, Expression2),
impint(LineNo, LineNo2)], CheckedProofs) :-
```

```
    valid_box(LineNo, LineNo2, CheckedProofs, Expression, Expression2).

% impel(x,y)
valid_use_of_rule(_,[_, Expression, impel(LineNo, LineNo2)],
CheckedProofs) :-
    member([LineNo, Expression2, _], CheckedProofs),
    member([LineNo2, imp(Expression2, Expression), _], CheckedProofs).

% negint(x,y)
valid_use_of_rule(_,[_, neg(Expression), negint(LineNo, LineNo2)],
CheckedProof):-
    valid_box(LineNo, LineNo2, CheckedProof, Expression, 'cont').

% negel(x,y)
valid_use_of_rule(_,[_, cont, negel(LineNo, LineNo2)], CheckedProofs) :-
    member([LineNo, Expression, _], CheckedProofs),
    member([LineNo2, neg(Expression), _], CheckedProofs).

% contel(x)
valid_use_of_rule(_,[_, _, contel(LineNo)], CheckedProofs) :-
    member([LineNo, 'cont', _], CheckedProofs).

% negnegint(x)
valid_use_of_rule(_,[_, neg(neg(Expression)), negnegint(LineNo)],
CheckedProofs) :-
    member([LineNo, Expression, _], CheckedProofs).

% negnegel(x)
valid_use_of_rule(_,[_, Expression, negnegel(LineNo)], CheckedProofs) :-
    member([LineNo, neg(neg(Expression)), _], CheckedProofs).

% mt(x,y)
valid_use_of_rule(_,[_, neg(Expression), mt(LineNo, LineNo2)],
CheckedProofs) :-
    member([LineNo, imp(Expression, Expression2),_], CheckedProofs),
    member([LineNo2, neg(Expression2), _], CheckedProofs).

% pbc(x,y)
valid_use_of_rule(_,[_, Expression, pbc(LineNo, LineNo2)],
CheckedProofs):-
    valid_box(LineNo, LineNo2, CheckedProofs, neg(Expression), 'cont').

% lem
valid_use_of_rule(_,[_, or(Expression, neg(Expression)), 'lem'], _).
```

Appendix B - Example proofs

valid.txt

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

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

[
  [1, imp(p, imp(q, r)), premise],
  [
    [2, and(q, neg(r)), assumption],
    [3, q, andel1(2)],
    [4, neg(r), andel2(2)],
    [
      [5, p, assumption],
      [6, imp(q,r),  impel(5,1)],
      [7, r, impel(3,6)],
      [8, cont, negel(7,4)]
    ],
    [9, neg(p), negint(5,8)]
  ],
  [10, imp(and(q, neg(r)), neg(p)), impint(2,9)]
].
```

Run of valid.txt

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- cd('C:/Users/tjeck/OneDrive/Dokument/Prolog').
true.

?- ['beviskoll.pl'].
true.

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

```
Yes
true
```

invalid.txt

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

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

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

Run of invalid.txt

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- cd('C:/Users/tjeck/OneDrive/Dokument/Prolog').
true.

?- ['beviskoll.pl'].
true.

?- verify('invalid.txt').
```

```
No
false.
```

Run of all the given proofs in the assignment

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- cd('C:/Users/tjeck/OneDrive/Dokument/Prolog').
true.

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

?- run_all_tests('beviskoll.pl').
valid01.txtYes passed
valid02.txtYes passed
valid03.txtYes passed
valid04.txtYes passed
valid05.txtYes passed
valid06.txtYes passed
valid07.txtYes passed
valid08.txtYes passed
valid09.txtYes passed
valid10.txtYes passed
valid11.txtYes passed
valid12.txtYes passed
valid13.txtYes passed
valid14.txtYes passed
valid15.txtYes passed
valid16.txtYes passed
valid17.txtYes passed
valid18.txtYes passed
valid19.txtYes passed
valid20.txtYes passed
valid21.txtYes passed
invalid01.txtNo passed
invalid02.txtNo passed
```

```
invalid03.txtNo passed
invalid04.txtNo passed
invalid05.txtNo passed
invalid06.txtNo passed
invalid07.txtNo passed
invalid08.txtNo passed
invalid09.txtNo passed
invalid10.txtNo passed
invalid11.txtNo passed
invalid12.txtNo passed
invalid13.txtNo passed
invalid14.txtNo passed
invalid15.txtNo passed
invalid16.txtNo passed
invalid17.txtNo passed
invalid18.txtNo passed
invalid19.txtNo passed
invalid20.txtNo passed
invalid21.txtNo passed
invalid22.txtNo passed
invalid23.txtNo passed
invalid24.txtNo passed
invalid25.txtNo passed
invalid26.txtNo passed
invalid27.txtNo passed
invalid28.txtNo passed
true
```
