Lab 2 Report

Logic for Computer Scientists, DD1351

Group Allan & Mateusz

2024-11-20

Project members:
Mateusz Kaszyk
Allan Al Saleh

Introduction

This assignment has the goal of creating a Prolog program that verifies natural deduction proofs given to it by .txt files. It needs to have the ability to verify that all rules are being followed and that the correct statement has been proven at the end of the proof outputting either "Yes" or "No". It furthermore needs to be able to handle nested sub-proofs (i.e boxes within boxes).

Approach

We started out by making a skeleton version of the program that could handle the example proof given in the instructions:

```
[p].
p.
[[1, p, premise]].
```

The file reading was simple enough as it was given in the instructions for the assignment. We decided to split up the verification into verifying the proof itself and verifying that the goal is reached in a separate predicate. The proof verification was done by verifying each line of the proof separately and recursively calling on itself. When the program could handle this very simple proof we start adding the simple rules to test them along the way with our own proofs as well as the example proofs that where provided. Finally, we tackled the box handling along with more complicated rules testing. When it done running through all the provided proofs as a final test. The structure of the program makes it very easy to modify, remove or add rules if there ever would be a need for it.

Results

validate_proof

The "validate_proof" function is designed to verify whether a given proof is valid and meets the specified goal. For this purpose the code includes three arguments (Prems, Goal and Proof). The Prems represents premises. The Proof represents the actual proof which will conclude with the goal. The output will be either "Yes" or "No":

If the proof is valid and achieves the goal, Prolog prints "Yes". If the proof is either invalid or the goal is not matched, Prolog prints "No".

```
%Validate proof:
%Is the proof valid?
%if yes:
validate_proof(Prems, Goal, Proof) :-
    proofCorrect(Prems, Proof, []),
    goalMet(Goal, Proof),
    write("Yes"), !.
%if no:
validate_proof(Prems, _, Proof) :-
    \+ proofCorrect(Prems, Proof, []),
    write("No"), !.
validate_proof(_, Goal, Proof) :-
    \+ goalMet(Goal, Proof),
    write("No"), !.

validate_proof(_, Goal, Proof) :-
    \+ goalMet(Goal, Proof),
    write("No"), !.
```

rowCorrect

The program validates all rules as well as the box handling with the "rowCorrect" predicate. Each line (or in the case of a box the whole box) is sent by the "proofCorrect" predicate to "rowCorrect" that then checks wheter the line matches one of its definitions. If it does the line checked is added to a list for future reference by other lines and the next line is checked by "rowCorrect.

Box Handling

The box detection first checks if the "row" given is a lists of lists where the first member is structured correctly and refers to the "assumption" rule as the first element always should use it. It then proceeds to check the whole sub-proof (box) is correct by sending it to the "proofCorrect" predicate along with the predicates and previously checked lines from the main proof. It also adds "[Nr, boxStart]" to the copy of the checked lines as a reference. When done the main proof continues to be checked and the whole box gets added to the previously checked lines as one element.

Arguments:
Prems: The premises of the proof.
[[Nr, X, assumption]|T]: The box. It starts with an assumption on the first line ("Nr") with the content "X". The "T" represents the rest of the box.
PreviousLinesCopy: A copy of the lines preceding the current proof.

```
%Box Detection
rowCorrect(Prems, [[Nr, X, assumption]|T],PreviousLinesCopy):-
    %always starts with assumption
    proofCorrect(Prems, [[Nr, X, assumption]|T], [[Nr, boxStart]|PreviousLinesCopy]).
    %+ check if rest of the box is correct
%[Nr, boxStart] is added to check on which line the box starts
%When the whole box is checked the temporary list is removed and the whole box is added
```

assumption

This part ensures that a assumption is valid and occurs at the start of the current box (subproof). It checks that the row is formatted correctly and refers to the "assumption" rule. The rule then validates that the line is in a box and that it's the first element by searching for a "boxStart" element with the same line number as the current statement.

```
%assumption
rowCorrect(_, [Nr, _, assumption], PreviousLines) :-
    member([Nr, boxStart], PreviousLines).
```

or elimination

This rule examines or-elimination (disjunction elimination). It ensures that both parts of a disjunction "or(X, Y)" separately reach the same result "Z". First it checks whether "or(X, Y)"is a statement at line "Nr1". It then goes on to validate whether there is a box assuming "X" and concluding "Z" between line "Nr2" and "Nr3" and does the same for "Y" between line "Nr4" and "Nr5".

```
%orel
rowCorrect(_, [_, Z, orel(Nr1, Nr2, Nr3, Nr4, Nr5)], PreviousLines) :-
    member([Nr1, or(X, Y), _], PreviousLines),
    member([[Nr2, X, _]|TX], PreviousLines),
    lastLine([[Nr2, X, _]|TX], LastLineX),
    [Nr3, Z, _] = LastLineX,
    member([[Nr4, Y, _]|TY], PreviousLines),
    lastLine([[Nr4, Y, _]|TY], LastLineY),
    [Nr5, Z, _] = LastLineY.
```

Table with predicates

Predicate	**Validity criteria**
select \3	True if a list with a variable removed results in a new list.
member \2	True if the first variable is a member of the second one.
lastLine \2	True if the first variable is the last member of the second one.
verify \1	True if the given document is found, the data from it is divided into 3 lists and "validate_proof \3" is true given those lists.
validate_proof \3	True if a given proof is correct and "Yes" is written out or if it's incorrect and "No" is written out.
proofCorrect \3	True if a given proof not accounting for the desired goal is correct.
rowCorrect \3	True if a row in the proof is correct (i.e., if a rule is used correctly).
goalMet \2	True if the last line of the proof shows the desired outcome of the proof.

Appendix A - Source code

```
%Helper functions
select(X,[X|T],T).
select(X,[Y|T],[Y|R]) :-
    select(X,T,R).

member(X,L) :- select(X,L,_).

%Get the last line of the proof
lastLine([H|[]], H). %base case with one line
lastLine([_|T], LastLine) :-
    lastLine(T, LastLine).
%End of helper functions


%Reading the proof
verify(InputFileName) :-
    see(InputFileName),
    read(Prems), read(Goal), read(Proof),
    seen,
    validate_proof(Prems, Goal, Proof).


%Is the proof valid?
%if yes:
validate_proof(Prems, Goal, Proof) :-
    proofCorrect(Prems, Proof, []),
    goalMet(Goal, Proof),
    write("Yes"), !.
%if no:
validate_proof(Prems, _, Proof) :-
    \+ proofCorrect(Prems, Proof, []),
    write("No"), !.
validate_proof(_, Goal, Proof) :-
    \+ goalMet(Goal, Proof),
    write("No"), !.


%Is the proof (not accounting for the goal) correct?
proofCorrect(_, [], _). %base case - end of proof
proofCorrect(Prems, [H|T], PreviousLines) :- %list is the proof
    rowCorrect(Prems, H, PreviousLines),
    proofCorrect(Prems, T, [H|PreviousLines]).
```

```
%Box Detection
%+ check if rest of the box is correct
%box always starts with assumption
rowCorrect(Prems, [[Nr, X, assumption]|T], PreviousLinesCopy) :-
    proofCorrect(Prems, [[Nr, X, assumption]|T], [[Nr, boxStart]PreviousLinesCopy]).
%[Nr, boxStart] is added to check on which line the box starts
%When the whole box is checked the temporary list is removed and the whole box is added

%premise
rowCorrect(Prems, [_, X, premise], _) :-
    member(X, Prems).

%assumption
rowCorrect(_, [Nr, _, assumption], PreviousLines) :-
    member([Nr, boxStart], PreviousLines).

%copy();
rowCorrect(_, [_, X, copy(Nr)], PreviousLines) :-
    member([Nr, X, _], PreviousLines).

%andint
rowCorrect(_, [_, and(X,Y), andint(Nr1, Nr2)], PreviousLines) :-
    member([Nr1, X, _], PreviousLines),
    member([Nr2, Y, _], PreviousLines).

%andel1
rowCorrect(_, [_, X, andel1(Nr)], PreviousLines) :-
    member([Nr, and(X, _), _], PreviousLines).

%andel2
rowCorrect(_, [_, X, andel2(Nr)], PreviousLines) :-
    member([Nr, and(_, X), _], PreviousLines).

%orint1
rowCorrect(_, [_, or(X, _), orint1(Nr)], PreviousLines) :-
    member([Nr, X, _], PreviousLines).

%orint2
rowCorrect(_, [_, or(_, X), orint2(Nr)], PreviousLines) :-
    member([Nr, X, _], PreviousLines).
```

```
%orel
rowCorrect(_, [_, Z, orel(Nr1, Nr2, Nr3, Nr4, Nr5)], PreviousLines) :-
    member([Nr1, or(X, Y), _], PreviousLines),
    member([[Nr2, X, _]|TX], PreviousLines),
    lastLine([[Nr2, X, _]|TX], LastLineX),
    [Nr3, Z, _] = LastLineX,
    member([[Nr4, Y, _]|TY], PreviousLines),
    lastLine([[Nr4, Y, _]|TY], LastLineY),
    [Nr5, Z, _] = LastLineY.

%impint
rowCorrect(_, [_, imp(X, Y), impint(Nr1, Nr2)], PreviousLines) :-
    member([[Nr1, X, _]|T], PreviousLines),
    lastLine([[Nr1, X, _]|T], LastLine),
    [Nr2, Y, _] = LastLine.

%impel
rowCorrect(_, [_, Y, impel(Nr1, Nr2)], PreviousLines) :-
    member([Nr1, X, _], PreviousLines),
    member([Nr2, imp(X, Y), _], PreviousLines).

%negint
rowCorrect(_, [_, neg(X), negint(Nr1, Nr2)], PreviousLines) :-
    member([[Nr1, X, _]|T], PreviousLines),
    lastLine([[Nr1, X, _]|T], LastLine),
    [Nr2, cont, _] = LastLine.

%negel
rowCorrect(_, [_, cont, negel(Nr1, Nr2)], PreviousLines) :-
    member([Nr1, X, _], PreviousLines),
    member([Nr2, neg(X), _], PreviousLines).

%contel
rowCorrect(_, [_, _, contel(Nr)], PreviousLines) :-
    member([Nr, cont, _], PreviousLines).

%negnegint
rowCorrect(_, [_, neg(neg(X)), negnegint(Nr)], PreviousLines) :-
    member([Nr, X, _], PreviousLines).

%negnegel
rowCorrect(_, [_, X, negnegel(Nr)], PreviousLines) :-
    member([Nr, neg(neg(X)), _], PreviousLines).
```

```
%mt
rowCorrect(_, [_, neg(X), mt(Nr1, Nr2)], PreviousLines) :-
    member([Nr1, imp(X, Y), _], PreviousLines),
    member([Nr2, neg(Y), _], PreviousLines).

%pbc
rowCorrect(_, [_, X, pbc(Nr1, Nr2)], PreviousLines) :-
    member([[Nr1, neg(X), _]|T], PreviousLines),
    lastLine([[Nr1, neg(X), _]|T], LastLine),
    [Nr2, cont, _] = LastLine.

%lem
rowCorrect(_, [_, or(X, neg(X)), lem], _).


%Is the last line of the proof the same as the goal?
goalMet(Goal, Proof) :-
    lastLine(Proof, LastLine),
    [_, Goal, _] = LastLine.
```

Appendix B - Example proofs

B.1 Incorrect proof

```
[imp(neg(s), k), imp(k, n), imp(and(k, n), neg(s))].

or(and(k, neg(s), and(s, neg(k)))).

[
 [1, imp(neg(s), k), premise],
 [2, imp(k, n), premise],
 [3, imp(and(k, n), neg(s)), premise],
        [[4, neg(s), assumption],
         [5, k, impel(4, 1)],
         [6, and(k, neg(s)), andint(5, 4)]
        ],
 [7, imp(neg(s), and(k, neg(s))), impint(4, 6)],
 [8, or(s, neg(s)), lem],
 [9, or(s, and(k, neg(s))), impel(8, 7)],
        [[10, s, assumption],
                [[11, k, assumption],
                  [12, n, impel(11, 2)],
                 [13, and(k, n), andint(11, 12)],
                 [14, neg(s), impel(13, 3)],
                 [15, cont, negel(10, 14)]
                ],
         [16, neg(k), negint(11, 15)],
         [17, and(s, neg(k))]
        ],
 [18, imp(s, and(s, neg(k))), impint(10, 17)],
 [19, or(and(k, neg(s), and(s, neg(k)))), impel(9, 18)]
].
```

B.2 Correct proof

```
[imp(neg(s), k), imp(k, n), imp(and(k, n), neg(s))].

or(and(k, neg(s)), and(s, neg(k))).

[
 [1, imp(neg(s), k), premise],
 [2, imp(k, n), premise],
 [3, imp(and(k, n), neg(s)), premise],
 [4, or(s, neg(s)), lem],
        [
         [5, neg(s), assumption],
         [6, k, impel(5, 1)],
         [7, and(k, neg(s)), andint(6, 5)],
         [8, or(and(k, neg(s)), and(s, neg(k))), orint1(7)]
        ],
        [
         [9, s, assumption],
                [[10, k, assumption],
                  [11, n, impel(10, 2)],
                 [12, and(k, n), andint(10, 11)],
                 [13, neg(s), impel(12, 3)],
                 [14, cont, negel(9, 13)]
                ],
         [15, neg(k), negint(10, 14)],
         [16, and(s, neg(k)), andint(9, 15)],
         [17, or(and(k, neg(s)), and(s, neg(k))), orint2(16)]
        ],
 [18, or(and(k, neg(s)), and(s, neg(k))), orel(4, 9, 17, 5, 8)]
].
```