DD1351 Logic for Computer Scientists
Alexander Järvheden 0211013693 jarvh@kth.se
Joachim Olsson 9908192876 joaco@kth.se
KTH Royal Institute of Technology
B.Sc Computer Science & Engineering
17 November 2023

DD1351 Logic for Computer Scientists

Lab 2: Natural Deduction Proof Check with Prolog

1. Algorithm and box implementation

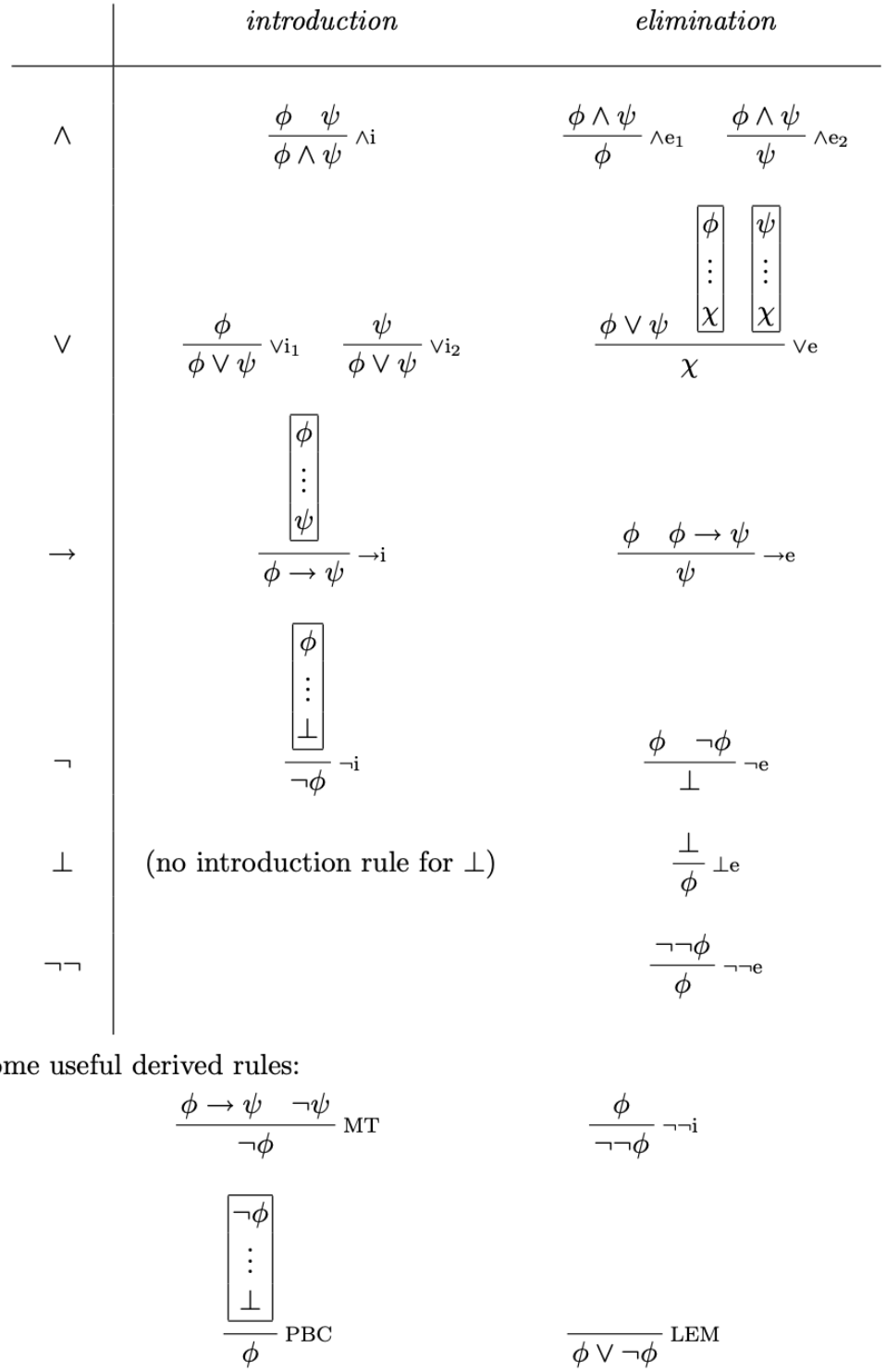
The verify predicate takes as input the name of a file containing premises, a goal, and a proof. It then proceeds to confirm the validity of the proof through the valid_proof predicate. The main logic of the code unfolds in the control_proof predicate, which examines each row in the proof using the control_row predicate. The list CheckedRows keeps track of validated rows as the proof is analyzed. The control_row predicate is the core of the validation process. It accounts for the rules in natural deduction taught in our course *Logic for Computer Scientists* (Appendix). A distinctive feature is the handling of assumption boxes through the open_box predicate. This accommodates scenarios where certain rows are treated as assumptions within a confined context and need to be appropriately managed. The predicates like second_last and extract_lists contribute to the overall functionality, enhancing the code's robustness. Another important part of the algorithm is making sure that the goal and that the last row actually contains the goal, which is checked through the predicate check_goal.

A box is opened when the first element in a row is a list, and these rows are required to contain an assumption on its third index. The contents inside the box are validated to make sure every statement is correct. The box has a temporary CheckedRows list which extends the CheckedRows to allow its rows to reference to rows outside the box and rows within the box. When the last row in the box has been reached the box closes, and the program keeps checking the following rows. In the CheckedRows, the programme appends it with the whole box, formatted as a list. This is so that rules that can not reference to rows in a box if they do not have permission to it. Exceptions are for some rules, and in this case their control_row predicate is implemented so that the code extracts the whole box/list from CheckedRows in order to access the rows in the box.

2. Implemented Predicates

Predicate	**True**	**False**
verify(inputFileName)	When the input file has at least three terms and valid_proof is true.	When valid_proof is false.
check_goal(Goal, Proof)	When the formula of the last row in Proof is the Goal.	When it's not the goal.
valid_proof(Prems, Goal, Proof)	When the last row in proof is the goal and the control_proof is true.	Either when the goal is not on the last row in the proof or if the proof is invalid.
control_proof(Prems, Proof, CheckedRows)	When every row in the proof is valid.	When a row in the proof is invalid.
control_row(Prems, Row, CheckedRows)	When the implemented rules validate the row.	If a row breaks the rules, except for assumption which breaks if it is fulfilled.
open_box(Box, CheckedRows, LastBoxRow)	When the rule is either assumption or valid.	When it's not an assumption or an invalid rule.
second_last(List, SecondLastElement)	When the second last element is a list (a row)	When the second last element is not a list
extract_lists(List, ListOfLists)	When it finds lists in the list	Never

3. Appendix: Natural Deduction rules, code and proof examples



Some useful derived rules:

$\frac{\phi \to \psi \quad \neg\psi}{\neg\phi}\text{MT}$ $\qquad$ $\frac{\phi}{\neg\neg\phi}\neg\neg\text{i}$

$\frac{\boxed{\begin{array}{c}\neg\phi\\ \vdots\\ \bot\end{array}}}{\phi}\text{PBC}$ $\qquad$ $\frac{}{\phi \vee \neg\phi}\text{LEM}$

Figure 1.2. Natural deduction rules for propositional logic.

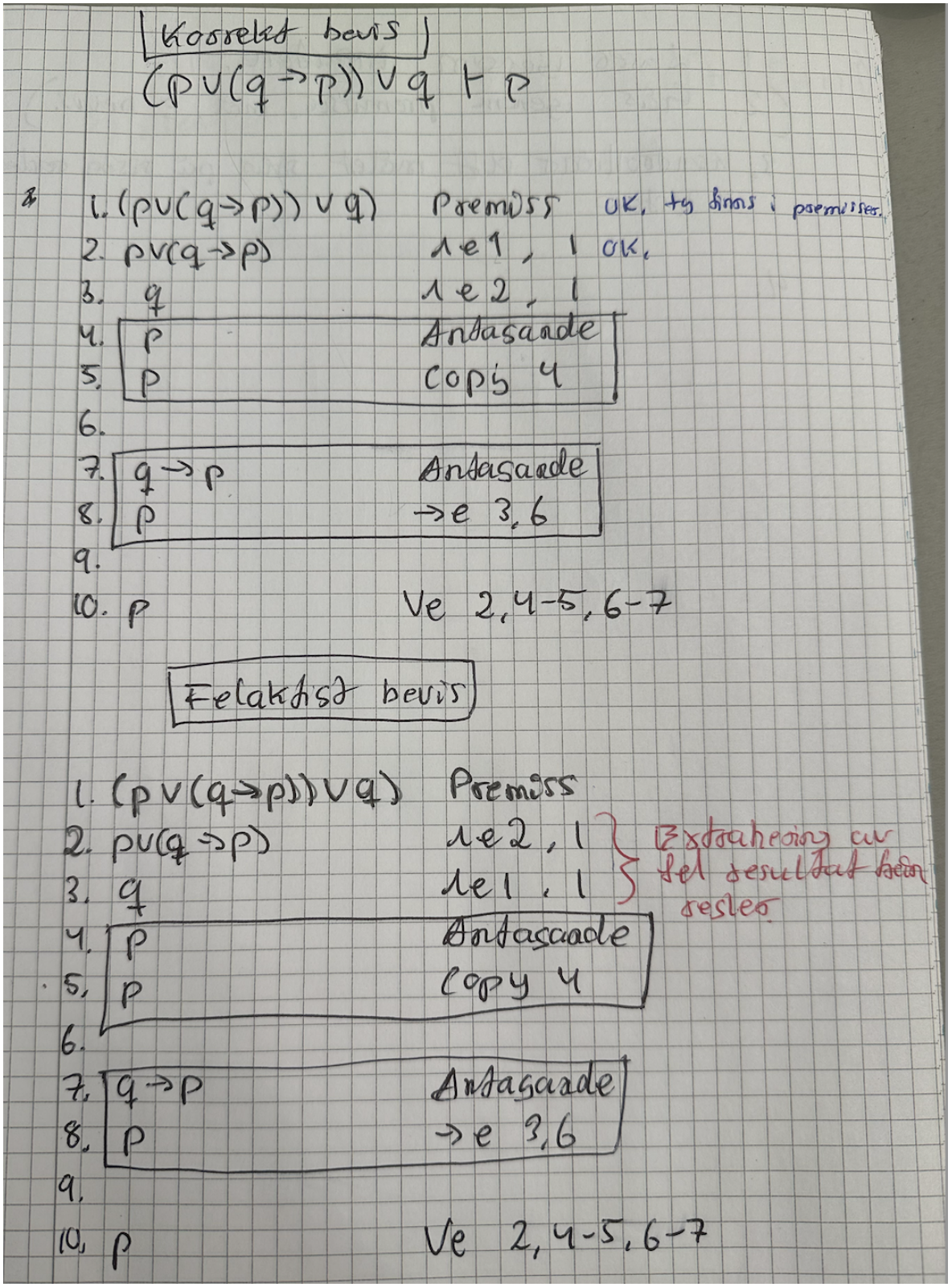
Copy

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

% Check that the last element is the goal
check_goal(Goal, Proof):- last(Proof, Last), nth1(2, Last, Goal).

% Validate proof by first checking that the last row is the goal and
then the rest of the proof
valid_proof(Prems, Goal, Proof):- check_goal(Goal, Proof),
control_proof(Prems, Proof, []), !.

% Control the whole proof
control_proof(_, [],_).
control_proof(Prems, [H | T], CheckedRows):-
    control_row(Prems, H, CheckedRows),
    append(CheckedRows,[H], AddedRows),
    control_proof(Prems, T, AddedRows), !.

% Set assumption rows to false, will be looked at in the open_box
predicate
control_row(_, [_, _, assumption], _):- !, false.

% Will open a box if the Row is a list/box
control_row(_, H, CheckedRows):-
    nth1(1, H, Elem),
    is_list(Elem),
    nth1(3, Elem, assumption),
    last(H, LastRowBox),
    open_box(H, CheckedRows, LastRowBox),
    findall(X, member([X, _, assumption], H), AssumptionList),
    length(AssumptionList, 1).

% Premises
control_row(Prems, [_, Form, premise], _):- member(Form, Prems), !.

% Copy
control_row(_, [_, Form, copy(X)], CheckedRows):- member([X, Form, _],
CheckedRows), !.

% And introduction
control_row(_, [_, and(FormX, FormY), andint(X,Y)], CheckedRows):-
    member([X, FormX, _], CheckedRows),
    member([Y, FormY, _], CheckedRows), !.
```

```
% And elimination
control_row(_, [_, Form, andel1(X)], CheckedRows):- member([X, and(Form,
_), _], CheckedRows), !.

control_row(_, [_, Form, andel2(X)], CheckedRows):- member([X, and(_,
Form), _], CheckedRows), !.

% Or introduction
control_row(_, [_, or(Form, _), orint1(X)], CheckedRows):- member([X,
Form, _], CheckedRows), !.

control_row(_, [_, or(_, Form), orint2(X)], CheckedRows):- member([X,
Form, _], CheckedRows), !.

% Implication elimination
control_row(_, [_, Form, impel(X, Y)], CheckedRows):-
    member([X, A, _], CheckedRows),
    member([Y, imp(A, Form), _], CheckedRows), !.


% Negation elimination
control_row(_, [_, cont, negel(X, Y)], CheckedRows):-
    member([X, Form, _], CheckedRows),
    member([Y, neg(Form), _], CheckedRows), !.

% Contradiction elimination
control_row(_, [_, _, contel(X)], CheckedRows):- member([X, cont, _],
CheckedRows), !.

% Negation-negation introduction
control_row(_, [_, neg(neg(Form)), negnegint(X)], CheckedRows):-
member([X, Form, _], CheckedRows), !.

% Negation-negation elimination
control_row(_, [_, Form, negnegel(X)], CheckedRows):- member([X,
neg(neg(Form)), _], CheckedRows), !.

% Modus Tollens
control_row(_, [_, neg(A), mt(X, Y)], CheckedRows):-
    member([X, imp(A, B), _], CheckedRows),
    member([Y, neg(B), _], CheckedRows), !.

% Lem
control_row(_, [_, or(A, neg(A)), lem], _).
```

```
% Or elimination
control_row(_, [_, Form, orel(X,Y,U,V,W)], CheckedRows):-
    extract_lists(CheckedRows, ListOfLists),
    second_last(ListOfLists, BoxRows1),
    last(ListOfLists, BoxRows2),
    member([X, or(A, B), _], CheckedRows),
    member([Y, A, _], BoxRows1),
    member([U, Form, _], BoxRows1),
    member([V, B, _], BoxRows2),
    member([W, Form, _], BoxRows2), !.

% Implication introduction
control_row(_, [_, imp(A, B), impint(X,Y)], CheckedRows):-
    extract_lists(CheckedRows, ListOfLists),
    last(ListOfLists, BoxRows),
    member([X, A, _], BoxRows),
    member([Y, B, _], BoxRows),
    last(BoxRows, LastRow),
    nth1(1, LastRow, LastRowNr),
    Y == LastRowNr,
    !.

% Negation introduction
control_row(_, [_, neg(A), negint(X,Y)], CheckedRows):-
    last(CheckedRows, BoxRows),
    member([X, A, _], BoxRows),
    member([Y, cont, _], BoxRows), !.

% PBC
control_row(_, [_, A, pbc(X,Y)], CheckedRows):-
    last(CheckedRows, BoxRows),
    member([X, neg(A), _], BoxRows),
    member([Y, cont, _], BoxRows), !.

% Open a box
open_box([H | _], CheckedRows, LastRowBox):- % Control last row, then
close the box
    H == LastRowBox,
    (nth1(3, H, assumption) ; control_row(_, H, CheckedRows)), !.

open_box([H | T], CheckedRows, LastRowBox):-
    \+(H == LastRowBox),
    (nth1(3, H, assumption), append(CheckedRows, [H], TempList);
```

```
    control_row(_, H, CheckedRows), append(CheckedRows, [H], TempList)),
    open_box(T, TempList, LastRowBox), !.

% Find second to last element in a list
second_last(L, X) :-
    append(_, [X, _], L),
    nth1(1, X, Elem),
    is_list(Elem).

% Filter a list with mix of elements and return a list containing only
lists
extract_lists([], []).
extract_lists([X|Rest], Lists) :-
    is_list(X),
    extract_lists(Rest, RestLists),
    Lists = [X|RestLists].
extract_lists([_|Rest], Lists) :-
    extract_lists(Rest, Lists).
```


Korrekt bevis
(p v (q → p)) v q ⊢ p
1. (p v (q → p)) v q Premiss OK, ty finns i premisser.
2. p v (q → p) ∨e 1, 1 OK,
3. q ∨e 2, 1
4. p Antagande
5. p Copy 4
6.
7. q → p Antagande
8. p →e 3, 6
9.
10. p ∨e 2, 4-5, 6-7
Felaktigt bevis
1. (p v (q → p)) v q Premiss
2. p v (q → p) ∨e 2, 1
3. q ∨e 1, 1
Extrahering av det resultat vi vill visa
4. p Antagande
5. p Copy 4
6.
7. q → p Antagande
8. p →e 3, 6
9.
10. p ∨e 2, 4-5, 6-7