

# Natural Deduction Proof Verifier

Konrad M. L. Claesson

konradcl@kth.se

Lab 2 – DD1351

November 21, 2019

## 1 Introduction

Natural deduction is a proof calculus comprised of a set of inference rules that are used to infer formulas from other formulas. The rules can be categorized into introduction and elimination rules. The former combine or negate formulas by introducing connectives; the latter decompose them by eliminating connectives. These rules are defined in terms of formula patterns that must hold true for a connective to be introduced or eliminated. For instance, to claim the formula  $\phi \wedge \psi$ , it must be known that both of the formulas  $\phi$  and  $\psi$  are true. Prolog is a logic programming language that enables easy expression of logical statements. To exemplify, the Prolog code

```
rule(_, [_ , and(X, Y), andint(R1, R2)], Verified) :-  
    member([R1, X, _], Verified),  
    member([R2, Y, _], Verified).
```

states that  $\phi \wedge \psi$  can be concluded, by the and introduction rule,

The application of said rules are controlled

All rules of natural deduction operate on propositional atoms, and recursively on the formulas that can be constructed by applying the rules on the atoms, to create new formulas by either introducing or eliminating connectives. The rules are defined in terms of the

For instance, it may be concluded that  $\phi : \neg(p \vee q)$  is logically equivalent to  $\psi : \neg p \wedge \neg q$  in the sense that  $\phi$  is true if, and only if,  $\psi$  is true. These

### Valid Proof of de Morgan's Law

The following is a valid proof of one of de Morgan's laws. Specifically, it proves the sequent  $\neg(p \vee q) \vdash \neg p \wedge \neg q$ .

|    |                        |                 |
|----|------------------------|-----------------|
| 1  | $\neg(p \vee q)$       | premise         |
| 2  | $p$                    | assumption      |
| 3  | $p \vee q$             | $\vee_i 2$      |
| 4  | $\perp$                | $\neg e 3, 1$   |
| 5  | $\neg p$               | $\neg i 2-4$    |
| 6  | $q$                    | assumption      |
| 7  | $p \vee q$             | $\vee_i 6$      |
| 8  | $\perp$                | $\neg e 7, 1$   |
| 9  | $\neg q$               | $\neg i 6-8$    |
| 10 | $\neg p \wedge \neg q$ | $\wedge i 5, 9$ |

In Prolog notation, this proof is written:

```
[1,  neg(or(p, q)),      premise],
[
  [2,  p,                assumption],
  [3,  or(p, q),          orint1(2)],
  [4,  cont,              negel(3, 1)]
],
[5,  neg(p),             negint(2, 4)],
[
  [6,  q,                assumption],
  [7,  or(p, q),          orint2(6)],
  [8,  cont,              negel(7, 1)]
],
[9,  neg(q),             negint(6, 8)],
[10, and(neg(p), neg(q)), andint(5, 9)]
```

The proof verification algorithm (view appendix) successfully identifies this as a valid proof. To do this, the algorithm does the following:

1. `verify(InputFileName)` is called and reads in the proof from the specified file. The premises are stored in an array called `Premises`, the sequent's conclusion is stored in `Conclusion`, and the array containing the proof is stored in `Proof`.
2. When Prolog has parsed the file, the rule `valid_proof(Premises, Conclusion, Proof)` is invoked, which in turn calls `verify_end(Conclusion, Proof)` and `verify_proof(Premises, Proof, [])`, in the given order.
3. `verify_end(Conclusion, Proof)` ensures that the final line of the proof is the same as the sequent's conclusion. This is achieved by first applying Prolog's built-in `last/3` predicate to extract the last element (line) in `Proof`, and then utilizing `nth0/3` to select the formula present on the last line. If the proof is valid, this formula is equal to the sequent's conclusion. Accordingly, `verify_end` concludes with an equality check that ensures that the described equality holds.
4. `verify_proof(Premises, Proof, Verified)` recursively validates the provided proof line-by-line, from top to bottom. It validates each line (element in `Proof`) by invoking `rule/3` with the premises, line, and previously verified lines as arguments. `rule/3` then returns `true` whenever the line follows by natural deduction from the premises and previously verified lines, and false otherwise. If a line is valid, it gets appended to the `Verified` array and `verify_proof` is called again with the yet unverified lines and the new array of verified lines.
5. `rule/3` takes in the array of premises, a line or assumption box, and the, as of then, verified lines of a proof. It then ensures that the line or assumption box follows by natural deduction from the premises and previous lines of the proof. It does this by inspecting the last element of each line, which specifies the deduction rule that was applied to deduce the formula of that line, and then checking that the prerequisites for

using the rule are fulfilled. To exemplify, in the above proof the first line invokes the premise verification rule

```
rule(Premises, [_ , Formula, premise], _) :-
    member(Formula, Premises).
```

which checks that **Formula** (the formula on the given line) is a member of **Premises**. If **Formula** is a member, the line is valid and **rule** returns **true**. Line two is an assumption and thus opens a box. It is handled by the assumption rule

```
rule(Premises, [[R, Formula, assumption] | T], Verified) :-
    append(Verified, [[R, Formula, assumption]], VerifiedNew),
    verify_proof(Premises, T, VerifiedNew).
```

which returns **true** if the sub-proof of the assumption box is valid. To verify that the assumption box contains a valid proof the assumption line is appended to the array of verified lines, and **verify\_proof/3** is called to validate the sub-proof. Within the box, line three is verified by ensuring that its application of the first disjunction introduction rule is valid. The below code does this

```
rule(_, [_ , or(X, _), orint1(R)], Verified) :-
    member([R, X, _], Verified).
```

by confirming that the formula  $p$  has been deduced earlier in the proof. Line four is deduced by administering the rule of negation elimination. The below code verifies that the rule can be employed

```
rule(_, [_ , cont, negel(R1, R2)], Verified) :-
    member([R1, X, _], Verified),
    member([R2, neg(X), _], Verified).
```

by checking that both  $p \vee q$  and  $\neg(p \vee q)$  occur previously in the proof (are members of **Verified**). When the sub-proof of the assumption

box has been validated, the entire box is appended to **Verified** before continuing to verify the next line or assumption box. This ensures that no single line that is predicated on the assumption can be referenced without also referencing the assumption. In the above proof, the fifth line, which also is the first line following the upper assumption box, is deduced by applying the rule of negation introduction. The subsequent code verifies that the rule can be applied

```
rule(_, [_ , neg(X), negint(R1, R2)], Verified) :-
    member([[R1, X, assumption] | T], Verified),
    last(T, BoxConclusion),
    [R2, cont, _] = BoxConclusion.
```

by establishing that the assumption box starts with  $p$ , and that it ends with a contradiction.

Lines six through nine repeat the same deductive pattern as lines two through five. The concluding line of the proof is arrived at by utilizing the rule of conjunction introduction. The employment of this rule is validated by the below code

```
rule(_, [_ , and(X, Y), andint(R1, R2)], Verified) :-
    member([R1, X, _], Verified),
    member([R2, Y, _], Verified).
```

which states that the rule can be administered if, and only if, both  $\neg p$  and  $\neg q$  occur previously in the proof (exist in **Verified**).

### **Invalid Proof of de Morgan's Law**

The following is an invalid proof of the same sequent as above. Lines one through three of the proposed lines are deduced by rules that we have already discussed, and hence they will not be discussed any further. Nevertheless, the fourth and final line incorrectly applies the rule of negation introduction. The subsequent text examines how the proof verification algorithm handles this in detail.

|   |                        |               |
|---|------------------------|---------------|
| 1 | $\neg(p \vee q)$       | premise       |
| 2 | $p \vee q$             | assumption    |
| 3 | $\perp$                | $\neg e$ 2, 1 |
| 4 | $\neg p \wedge \neg q$ | $\neg i$ 2-3  |

Recall that the negation introduction rule can be employed to conclude  $\neg\phi$ , for some formula  $\phi$ , if, and only if, there exists a box starts with an assumption of  $\phi$  and ends in a contradiction. The proof passes the latter requirement, which is enforced by the code

```
last(T, BoxConclusion),
[R2, cont, _] = BoxConclusion.
```

but fails the former, as the assumption  $\phi = p \vee q$  combined with the contradiction and negation introduction rule, can only be used to conclude  $\neg(p \vee q)$ , and not  $\neg p \vee \neg q$ .

| Predicate    | Parameters                       | Truth Conditions   |
|--------------|----------------------------------|--|
| verify       | InputFileName                    | valid_proof  |
| valid_proof  | Premise<br>Conclusion<br>Proof   | verify_end<br>verify_proof   |
| verify_end   | Conclusion<br>Proof              | Proof ends with sequent's conclusion.  |
| verify_proof | Premises<br>Proof<br>Verified    | The first line, and recursively all subsequent lines, are provable by natural deduction.   |
| rule         | Premises<br>Line/Box<br>Verified | <p>If the second argument is an assumption box, <b>rule</b> is true if the sub-proof of the box is valid (i.e. if <b>verify_proof</b> returns true when invoked with the sub-proof as its <b>Proof</b> argument).</p> <p>If the second argument is a line, the conditions determining the return value of <b>rule</b> differ based on what deduction rule was used to arrive at the formula on the line.</p> <p>For lines that are <i>premises</i>, the <b>Formula</b> on the line must exist in the <b>Premises</b> of the proof.</p> <p>For lines that are deduced by the <i>copy</i> rule, the copied formula must appear previously in the proof (i.e. be a member of <b>Verified</b>).</p> <p>For lines with a formula <math>\neg\neg\phi</math>, that is deduced by <i>double negation, introduction</i>, the formula <math>\phi</math> must appear previously in the proof (i.e. exist in <b>Verified</b>).</p> <p>For lines with a formula <math>\phi</math> that is deduced by <i>double negation elimination</i>, the formula <math>\neg\neg\phi</math> must appear previously in the proof (i.e. exist in <b>Verified</b>).</p> |

| Predicate | Parameters | Truth Conditions  |
|-----------|------------|---|
|           |            | <p>For a line with a formula <math>\phi \wedge \psi</math> that is deduced by <i>and introduction</i>, the formulas <math>\phi</math> and <math>\psi</math> must appear previously in the proof (i.e. exist in <b>Verified</b>).</p> <p>For a line with a formula <math>\phi</math>, that is deduced by <i>first and elimination</i>, a formula of the form <math>\phi \wedge \psi</math> must appear previously in the proof (i.e. exist in <b>Verified</b>).</p> <p>For a line with a formula <math>\phi</math>, that is deduced by <i>second and elimination</i>, a formula of the form <math>\psi \wedge \phi</math> must appear previously in the proof (i.e. exist in <b>Verified</b>).</p> <p>For a line with a formula <math>\phi \vee \psi</math>, that is deduced by <i>first or introduction</i>, a formula <math>\phi</math> must appear previously in the proof (i.e. exist in <b>Verified</b>).</p> <p>For a line with a formula <math>\phi \vee \psi</math>, that is deduced by <i>second or introduction</i>, a formula <math>\psi</math> must appear previously in the proof (i.e. exist in <b>Verified</b>).</p> <p>For a line with a formula <math>\chi</math>, that is deduced by <i>or elimination</i>, a formula <math>\phi \vee \psi</math>, a box assuming <math>\phi</math> and ending in <math>\chi</math>, and a box assuming <math>\psi</math> and ending in <math>\chi</math>, must appear previously in the proof (i.e. exist in <b>Verified</b>).</p> <p>For a line with a formula <math>\phi \rightarrow \psi</math>, that is deduced by <i>implication introduction</i>, a box assuming <math>\phi</math> and concluding in <math>\psi</math> must appear previously in the proof (i.e. exist in <b>Verified</b>).</p> |



| Predicate | Parameters | Truth Conditions  |
|-----------|------------|---|
|           |            | <p>For a line with a formula <math>\psi</math> that is deduced by <i>implication elimination</i> a formula <math>\phi</math>, and a box assuming <math>\phi</math> and ending in <math>\psi</math>, must appear previously in the proof (i.e. exist in <b>Verified</b>).</p> <p>For a line with a formula <math>\neg\phi</math> that is deduced by <i>negation introduction</i> a box assuming <math>\phi</math> and concluding in <math>\perp</math> must appear previously in the proof (i.e. exist in <b>Verified</b>).</p> <p>For a line with a contradiction <math>\perp</math> that is deduced by <i>negation elimination</i> two formulas <math>\phi</math> and <math>\neg\phi</math> must appear previously in the proof (i.e. exist in <b>Verified</b>).</p> <p>For a line with a formula <math>\phi</math> that is deduced by <i>contradiction elimination</i> a line with a contradiction <math>\perp</math> must appear previously in the proof (i.e. exist in <b>Verified</b>).</p> <p>For a line with a formula <math>\neg\phi</math> that is deduced by <i>modus tollens</i> a formula <math>\phi \rightarrow \psi</math>, and a formula <math>\neg\psi</math>, must appear previously in the proof (i.e. exist in <b>Verified</b>).</p> <p>For a line with a formula <math>\phi</math> that is derived by a <i>proof by contradiction</i> a box assuming <math>\neg\phi</math> and ending in a contradiction <math>\perp</math> must appear previously in the proof (i.e. exist in <b>Verified</b>).</p> <p>By the <i>law of the excluded middle</i> a line with a formula of the form <math>\phi \vee \neg\phi</math> is always valid.</p> |