

How to type-set logic and natural deductions using GNU `troff`, `pic` and `eqn`

Sigrid Lundberg
sigfrid (at) sigfrid-lundberg.se

Introduction

`troff` is a software system for type-setting using Unix™ and related operating systems (Ossanna and Kernighan, 1994). Brian W. Kernighan was one of the creators of Unix and the C programming language. `pic` (Kernighan, 1982) and `eqn` (Kernighan and Cherry, 1975) are systems for typesetting graphs and mathematics, also created by Brian Kernighan and his friends. This group created a full set of tools for the type-setting of scientific text, graphs and diagrams, mathematics, chemistry, reference management and complex tables. GROFF aka GNU `troff` is the implementation I am using (FSF, 1990). There are other competitors, in particular the slightly younger TeX and LaTeX, I am not able to give you advice on how to use those for logic or science. There are alternative implementations of `troff`, but the GNU one is the one I have used for 35 years.

These systems are old, but they are readily available for Linux and Apple iOS computers, because they are based on the Unix™ operating system (guess that you can get them for Windows as well). `troff` were developed by software developers for software developers, scientists and mathematicians. It is not very difficult to use, just very different from what modern users expect. And you must write your document in a programmer's editor. `troff` works on shell command lines.

Fitch is a notation for natural deduction (Pelletier and Hazen, 2024). It has got its name after its inventor, Fredric Fitch. This notation seems to be a de facto standard: It is used in all the text books I have been able to find electronically, and seems to be taught at logics courses in mathematics as well as philosophy. I started to write this tutorial while learning Fitch while reading philosophy; my intention is to demonstrate how to write predicate and propositional logic, and deduction on this platform. I cannot teach you how to format scientific text in general (that is large job), neither can I give an introduction to logic and natural deduction (I am not qualified for that).

This document lives in an open project at github.

<https://github.com/siglun/logic-and-groff>

Feel free to use the snippets of code and examples.

Writing text and equations

First, we need to be able to write our texts (there is a good tutorial by Kollar and Robinson (2023)) and then we can continue with formulas and sentences using `eqn` (Harding, 2011). At a first glance, all of them are “equations”, or anyone who are not familiar with mathematics and logic and the differences between the two will regard them as such. Logic is

Table 1. Unicode characters for logical signs and operators. On some operating systems you can type them by pressing `ctrl-shift-u` and then the four character code (following `u+` in the table). The Groff name is usually better to use than the Unicode character, but takes a long time to type. The eqn macros are for easier typing and I have tried to adjust spacings for a nicer look.

| Unicode | Character | Groff name | eqn macro | Comment |
|---------|-----------|------------|-------------|---|
| U+00AC | ¬ | \[no] | not | |
| U+2227 | ∧ | \[AN] | and | |
| U+2228 | ∨ | \[OR] | or | |
| U+2200 | ∀ | \[fa] | any | |
| U+2200 | ∀ | \[fa] | forall(x) | |
| U+2203 | ∃ | \[te] | some | |
| U+2203 | ∃ | \[te] | exists(x) | |
| U+2192 | → | \[->] | implies | |
| U+2194 | ↔ | \[<>] | iff | |
| U+2194 | ↔ | \[<>] | equiv | |
| U+21D4 | ⇔ | \[hA] | | |
| U+22A5 | ⊥ | \[pp] | falsum | |
| U+22A2 | ⊢ | \[/-] | vdash† | syntactic consequence, turnstile requires font support* |
| U+22A2 | ⊢ | \[/-] | entails† | |
| U+22A8 | ⊨ | \[/--] | vDash† | semantic consequence, double turnstile requires font support* |
| U+22A8 | ⊨ | \[/--] | satisfies† | |
| U+2261 | ≡ | \[==] | identicalto | |
| U+25A1 | □ | \[sq] | nece | |
| U+25A1 | □ | \[sq] | necessarily | |
| U+25C7 | ◇ | \[lz] | possi | |
| U+25C7 | ◇ | \[lz] | possibly | |
| U+2234 | ∴ | \[tf] | therefore | |
| U+2205 | ∅ | \[es] | empty | |
| U+2208 | ∈ | \[mo] | member | |
| U+2209 | ∉ | \[nm] | notmember | |
| U+2286 | ⊆ | \[ib] | subset | |
| U+2118 | ℘ | \[wp] | powerset | |

*) I use the math font from Scientific and Technical Information Exchange (STIX) font creation project <https://www.stixfonts.org/>

†) These operators are names used in eqn only. `vdash` and `vDash` are used in TeX/LaTeX for \vdash and \vDash respectively. I have also added the operators `entails` and `satisfies` because I feel that I can remember them and their functions.

actually a special genre of its own when it comes to formulas or equations.

To write these formulas, you need to use either the unicode characters, their Groff names or macros I have defined in order to simplify typing. See Table 1.

Here is a set of predicate logic sentences, first formatted and then eqn source,

Leibniz' laws of identity: If everything which is true about A are true for B , and vice versa, then $A = B$. Here are some variations upon that theme. (i) The indiscernibility of identicals

$$\forall x \forall y \left[x = y \rightarrow \forall F (Fx \rightarrow Fy) \right]$$

Here is the source of that equation

```
.EQ
forall(x) forall(y) left [ x=y implies forall(F) (Fx implies Fy) right ]
.EN
```

That is, if x equals y then y has all the properties of x .

(ii) The identity of indiscernibles:

$$\forall x \forall y \left[\forall F (Fx \rightarrow Fy) \rightarrow x = y \right]$$

The source code

```
.EQ
forall(x) forall(y) left [ forall(F) (Fx implies Fy) implies x=y right ]
.EN
```

If for all x and y each property of x also is also a property of y , then this implicates that x and y are identical.

(iii) Biconditional version of "Leibniz's Law":

$$\forall x \forall y \left[x = y \leftrightarrow \forall F (Fx \leftrightarrow Fy) \right]$$

The source code

```
.EQ
forall(x) forall(y) left [ x=y iff forall(F) ( Fx iff Fy) right ]
.EN
```

For each x and y it is true that $x = y$ if and only all properties of x are also properties of y . This is what we refer to as Leibniz' law.

Using logics in tables and graphs

See Table 2 and Figure 1. These things are here just for demonstrating how typographic elements can be combined.

Writing Fitch arguments

Authoring natural deduction is similar to embedding formulas in a graph, as demonstrated in Figure 1.

Any proof is initialized by calling this macro, which informs all scripts on the number of steps in the proof and its maximum depth, i.e., how deep the hierarchy of proofs is. That is, how many inside proofs, within proofs ... do we have. You better add one or the references at the right will come to close to the logical statements.

```
set_steps_and_depths(8,3)
```

Any proof (the root proof or any sub-proof) starts with the `start_proof()` macro, which also names that proof. After we have started the proof, we add its premises, and end it with `premis_end()`.

Table 2. Some useful equivalents if you are doing logic. They are presented here as an example how you can embed formulas in a table.

| | | |
|---------------------------|---|---|
| Assertion of Universality | $\forall x Ax \leftrightarrow \neg \exists x \neg Ax$ | If everything is, there exists nothing that is not. |
| Denial of Universality | $\neg \forall x Ax \leftrightarrow \exists x \neg Ax$ | If not everything is, there exists something that is not. |
| Denial of Existence | $\forall x \neg Ax \leftrightarrow \neg \exists x Ax$ | If everything is not, there exists nothing that is. |
| Assertion of Existence | $\neg \forall x \neg Ax \leftrightarrow \exists x Ax$ | If not everything is not, there exists something that is. |

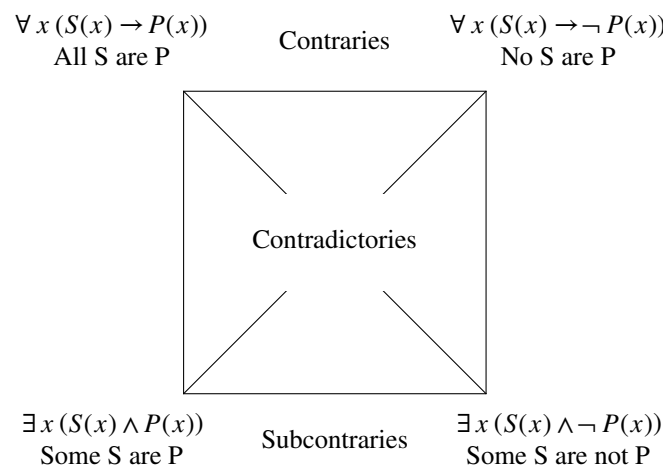


Figure 1. The traditional aristotelian syllogisms, the figure shows how one can embed formulas into a graph.

```

start_proof (START);
add_premis (START, "$A or B$");
add_premis (START, "$not A$");
premis_end (START);

```

After ending the premiws section, we enter the body of our proofs. In this case we start the sub-proofs

```

start_proof (SUB1);
add_premis (SUB1, "$A$");
premis_end (SUB1);

```

In the body of a proof, we use the `add_step()` macro, which has three argument: (i) the name of the current proof, (ii) the result of the step, and finally (iii) the references to earlier steps needed for the step.

```

add_step (SUB1, "$falsum$", "\perp Intro: 3, 2");
add_step (SUB1, "$B$", "\perp Elim: 4");
end_proof (SUB1);

```

| | | |
|---|------------|------------------------|
| 1 | $A \vee B$ | |
| 2 | $\neg A$ | |
| 3 | A | |
| 4 | \perp | \perp Intro: 3,2 |
| 5 | B | \perp Elim: 4 |
| 6 | B | |
| 7 | B | Reit: 6 |
| 8 | B | \vee Elim: 6-7,3-5,1 |

Figure 2. Proof that $A \vee B, \neg A \therefore B$. The line numbering is in the left-most margin. Then there is a vertical line, as long as the proof. The step 1-2 in the proof is where the premises lives. The horizontal line after step 2 is usually referred to as the *fitch line*. The two groups, 3–5 and 3–6 are sub-proofs, with their own premisses, vertical lines and fitch lines

We end a proof (be it a sub_proof or a proof) with the `end_proof()` macro, which needs the name of the current proof as an argument. Now we start another subproof.

```
start_proof(SUB2);
add_premis(SUB2, "$B$");
premis_end(SUB2);
add_step(SUB2, "$B$", "Reit: 6");
end_proof(SUB2);
```

After we have completed the two sub-proofs, return to the main proof and completes it with a nice \vee elimination.

```
add_step(START, "$B$", "\vee Elim: 6-7, 3-5, 1");
end_proof(START)
```

Note that the macros do not check your references. Sanity checks and proof reading is your job. There are two more examples below, in Figure 3–4.

References

- FSF, Free Software Foundation, *Groff* (1990). <https://www.gnu.org/software/groff/>.
- Harding, Ted, *A Guide to Typesetting Mathematics using GNU eqn* (2011). <https://lists.gnu.org/archive/html/groff/2013-10/pdfTyBN2VWR1c.pdf>.
- Kernighan, Brian W., “PIC — A language for typesetting graphics,” *Software: Practice and Experience* **12** (1982). <https://doi.org/10.1002/spe.4380120102>.

| | | |
|----|-----------------|------------------------|
| 1 | $A \vee B$ | |
| 2 | $\neg B \vee C$ | |
| 3 | A | |
| 4 | $A \vee C$ | \vee Intro:3 |
| 5 | B | |
| 6 | $\neg B$ | |
| 7 | \perp | \perp Intro:6,5 |
| 8 | $A \vee C$ | \perp Elim:7 |
| 9 | C | |
| 10 | $A \vee C$ | \vee Intro:9 |
| 11 | $A \vee C$ | \vee Elim:2,6-8,9-10 |
| 12 | $A \vee C$ | \vee Elim:1,3-4,5-11 |

Figure 3. A slightly longer example: Prove that $A \vee B, \neg B \vee C \therefore A \vee C$.

- Kernighan, Brian W. and Cherry, Lorinda L., “A System for Typesetting Mathematics,” *Scientific Applications* **18** (1975). <https://doi.org/10.1145/360680.360684>.
- Kollar, Larry and Robinson, G. Branden, *Using groff with the ms Macro Package* (2023). <https://lists.gnu.org/r/groff/2023-04/pdfkxOTeCk24w.pdf>.
- Ossanna, Joseph F. and Kernighan, Brian W., “Troff User’s Manual,” *Computing Science Technical Report* **54** (1994). <https://wolfram.schneider.org/bsd/7thEdManVol2/trofftut/trofftut.pdf>.
- Pelletier, Francis Jeffry and Hazen, Allen, “Natural Deduction Systems in Logic” in *The Stanford Encyclopedia of Philosophy (Spring 2024 Edition)*, ed. Zalta, E. N. and U. Nodelman (2024). <https://stanford.io/4jpc5KF>.

| | | | | | |
|----|---|-----------------------------------|-----------------------|----------------------|-------------------|
| 1 | | | | | |
| 2 | | $\forall x Fx$ | | | |
| 3 | | | $\exists x \neg Fx$ | | |
| 4 | | | | $[a] \neg Fa$ | |
| 5 | | | | Fa | \forall Elim 2 |
| 6 | | | | \perp | \perp Intro 4,5 |
| 7 | | | \perp | \exists Elim 3,4-6 | |
| 8 | | $\neg \exists x \neg Fx$ | | | |
| 9 | | $\neg \exists x \neg Fx$ | | | |
| 10 | | | $[a] \neg Fa$ | | |
| 11 | | | $\exists x \neg Fx$ | \exists Intro 10 | |
| 12 | | | \perp | \perp Intro 9,11 | |
| 13 | | | $\forall x Fx$ | \perp Elim 12 | |
| 14 | | $\forall x Fx$ | \forall Intro 10-13 | | |
| 15 | $\forall x Fx \leftrightarrow \neg \exists x \neg Fx$ | \leftrightarrow Intro 2-8, 9-14 | | | |

Figure 4. A proof using predicate logic. It proves without premisses one of the de Morgan formulas: $\forall x Fx \leftrightarrow \neg \exists x \neg Fx$, All x are F if and only if there is no x which is not F .

Github project

Scan QR to get project at <https://github.com/siglun/logic-and-groff>

