DeBox Manual

Io Boye Pinnerup

September 5, 2016

Contents

List of Figures

Introduction

DeBox is a teaching tool for teaching propositional logic. It allows the user to write natural deduction proofs in (pseudo) natural language, by using a specified notation in English, rather than the Fitch-style notation presented in the course textbook. Additionally, DeBox provides proof validation of such natural language proofs. If valid, the proof is returned in BoxProof notation; if invalid, feedback is provided in the form of a concrete and detailed list of errors.

This manual provides a quick walk-through of how to formulate natural language proofs, and how to validate them using DeBox.

Writing proofs

All proofs follow the same structure, consisting of three main parts:

1. The title followed by a colon.
2. The sequent to be proven.
3. The steps by which the conclusion is reached.

The title can be any string consisting of alphanumerical characters, hyphens, and underscores, and *must* be followed by a colon. Like this:

```
example_title:
```

Figure 1: Example of a proof title.

The sequent should be phrased in one of three different ways, depending on the number of premises:

- **None:**
  We wish to prove `<formula>`.
- **One:**
  From the premise `<premise>`, we wish to prove `<formula>`.
- **Two or more:**
  From the premises `<premise 1, ...,>` and `<premise n>`, we wish to prove `<formula>`.

The sequent (like each step) *must* end with a period. Formulae (both premises and intended goal) are written using strings (alphanumeric characters, hyphens, underscores) for propositional atoms, and using symbols for logical operators and to signify absurdity. These can be written using `ASCII` characters or the appropiate Unicode character – see figure 2 below. For implication, both `=>` and `->` are accepted, to allow for individual preferences.

```
implication → =>  | → | ->
disjunction → \/  | ∨
conjunction → /\  | ∧
negation    →  ~  | ¬
absurdity   → _|_ | ⊥
```

Figure 2: Logical operators and ⊥

Proof steps

Each step is the natural language equivalent of a line in a fitch-style proof[1], and is either structural or deductive.

[1] With the exception of assumption discharges, since the domain of an assumption is illustrated by a framing box in fitch-style notation

Structural steps provide context for following steps, and come in three varieties:

- Stating premises
  · All premises must be stated before any other steps are taken.
  · Premises must be stated in the order in which they appear in the sequent.
- Stating assumptions
  · All assumptions must be discharged before the proof is closed.
- Discharging assumptions
  · Only the most recently made *undischarged* assumption may be disharged.

Deductive steps advance the proof by applying one of the rules of natural deduction in the current context. Each rule has very specific equivalent wording(s) in English (to emphasize the formalism of natural deduction proofs) and these can be found in the Quick-reference on page 5.

Rather than refer to previous steps by number, each step is given a unique reference id, listed in square brackets after the formula obtained in the step. These ids follow the same rules as titles and atoms, and are required in all steps except discharges (which optain no new formulae) and the concluding step of a proof.

Some rules reference a range of lines (`[<first>]-[<last>]`). These are the equivalent of boxes in fitch-style notation, and `[<first>]` should always refer to an assumption while `[<last>]` should always refer to the line immidiately before the discharge of that assumption.

To illustrate all this, figure 3 shows an example of a proof:

```
example_title:
From the premise (p→q)∧(p→r), we wish to prove p→q∧r.
We have the premise (p→q)∧(p→r) [prm].
By applying the first and-elimination rule to [prm], we get p→q [piq].
By applying the second and-elimination rule to [prm], we get p→r [pir].
Assume p [p].
    By applying the implication-elimination rule to [p] and [piq], we get q [q].
    By applying the implication-elimination rule to [p] and [pir], we get r [r].
    By applying the and-introduction rule to [q] and [r], we get q∧r [qar].
Discharge assumption [p].
By applying the implication-introduction rule to [p]-[qar], we conclude p→q∧r.
```

Figure 3: Example of a proof.

Note:
In terms of English grammar, the proof syntax accepts lists both with and without Oxford comma[2], but ensures the use of obligatory comma after an adverbial phrase when it precedes the main clause (e.g. in 'By applying ... **,** we get ...').

[2]Also known as the serial comma; the term denotes the comma before the connective in a list. With Oxford comma: 'propositions, formulae, and proofs'. Without: 'propositions, formulae and proofs'.

Validating proofs

To validate a proof, run the program `DeBox` with the relative path to your proof file as argument:

```
$ DeBox proofs/short_example.txt
```

Figure 4: Running `DeBox` on the short example proof provided

Valid proofs will return a BoxProver notation proof to the standard output stream (usually the terminal). This can be piped to a file, which can be opened directly from BoxProver.

Invalid proofs, if accepted, will return a list of errors to the standard error stream (usually also the terminal). This can be piped to an error-log file, if desired. If not accepted, either lexer or parser errors are returned. Lexer errors should be decipherable, but the parser errors can be more tricky. To help with those, a check-list is provided below:

Parser error checklist

- Parse error in col 0:
  - ? Did you forget to end the previous line with a period?
  - ? Did you mis-spell the rule on this line?
- Parse error on last line, col after formula:
  - ? Did you use ', we get' but have no `[<ref>]` for the line?
  - ? Did you use ', we conclude' and have a `[<ref>]` for the line?
- General:
  - ? Do you have double spaces somewhere on the line?

Quick-reference

Title, atoms, and references

Alphanumeric characters, hyphens, underscores. Title must be followed by a ':'.

Sequents

- We wish to prove `<formula>`.
- From the premise `<premise>`, we wish to prove `<formula>`.
- From the premises `<premise 1, ...,>` and `<premise n>`, we wish to prove `<formula>`.

Symbols

implication	→ =>	| → | ->
disjunction	→ \/	| ∨
conjunction	→ /\	| ∧
negation	→ ~	| ¬
absurdity	→ _|_	| ⊥

Steps

`premise` → We have the premise `<formula> [<id>].`
`assumption` → Assume `<formula> [<id>].`
`discharge` → Discharge assumption `[<id>].`

or one of

`copy` : By copying`[<id>]`
$\wedge_i$: By applying the and-introduction rule to `[<id>]` and `[<id>]`
By applying the conjunction-introduction rule to `[<id>]` and `[<id>]`
$\wedge_{e1}$: By applying the first and-elimination rule to `[<id>]`
By applying the first conjunction-elimination rule to `[<id>]`
$\wedge_{e2}$: By applying the second and-elimination rule to `[<id>]`
By applying the second conjunction-elimination rule to `[<id>]`
$\vee_{i1}$: By applying the first or-introduction rule to `[<id>]`
By applying the first disjunction-introduction rule to `[<id>]`
$\vee_{i2}$: By applying the second or-introduction rule to `[<id>]`
By applying the second disjunction-introduction rule to `[<id>]`
$\vee_e$: By applying the or-elimination rule to `[<id>]`, `[<id>]-[<id>]`, and `[<id>]-[<id>]`
By applying the disjunction-elimination rule to `[<id>]`, `[<id>]-[<id>]` and `[<id>]-[<id>]`
$\rightarrow_i$: By applying the implication-introduction rule to `[<id>]-[<id>]`
$\rightarrow_e$: By applying the implication-elimination rule to `[<id>]` and `[<id>]`
By applying Modus Ponens to `[<id>]` and `[<id>]`
$\neg_i$: By applying the negation-introduction rule to `[<id>]-[<id>]`
$\neg_e$: By applying the negation-elimination rule to `[<id>]` and `[<id>]`
$\neg\neg_i$: By applying introduction of double-negation to `[<id>]`
$\neg\neg_e$: By applying elimination of double-negation to `[<id>]`
$\bot_e$: By applying elimination of absurdity to `[<id>]`
`MT` : By applying Modus Tollens to `[<id>]` and `[<id>]`
`PBC` : By applying proof by contradiction to `[<id>]-[<id>]`
`LEM` : By applying the law of the excluded middle

followed by either , we get `<formula> [<id>].`
OR (if last line) , we conclude `<formula>.`