Adapting ProofCheck to the author's needs

Bob Neveln and Bob Alps

Abstract

ProofCheck is a system for writing and checking mathematical proofs. Theorems and proofs are contained in a plain TeX or LaTeX document. Parsing and proof checking are accomplished through Python programs which read the source file. Although the use of these programs has never been restricted to any particular logic or mathematical language, the work required to actually implement an author's choices in these matters, especially in the logic, and to make the necessary modifications of the supporting files have been sufficiently laborious as to pose an obstacle to the use of ProofCheck. This paper describes updates to the system whose purpose is to alleviate these labors to the extent possible so as to facilitate the use of ProofCheck in a logical and linguistic setting of the author's choice.

1 What is ProofCheck?

ProofCheck is a Python package, available at `http://www.proofcheck.org`. It checks mathematical proofs written in TeX or LaTeX. Previous versions were described in [2] and [3]. The mode of operation of ProofCheck is extremely simple. It requires proofs to be structured into lines which have a justification at the right margin. Each line with its justification is matched against rules of inference from a list. If a match is found the line is checked. If all the lines of a proof check the proof checks.

This simplicity is reflected in the file structure of the system as shown in Figure 1. The rectangular boxes represent TeX files. The rule matcher and the unifier consist of Python code. We now briefly describe these TeX files.

1.1 (La)TeX documents

An author's document may be written in either plain TeX or in LaTeX. Figures 2 and 3 show a fictitious proof as it would appear in the source and output files respectively. Ordinary text may be used relatively freely in a proof. Mathematical expressions in TeX may also be included even if they play no role in the checking.

Structuring a proof so that it can be checked requires six proof structuring macros in TeX, five in LaTeX. Figure 2 shows all those needed in LaTeX. For plain TeX, these are used: `\chap`, `\prop`, `\note`, `\line[a-z]`, `\By`, `\Bye`. No other markup is needed in structuring proofs.

The logical and mathematical symbols shown

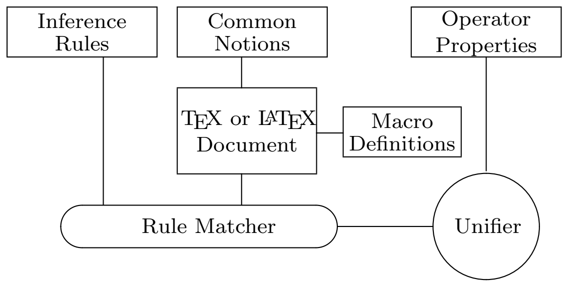

Figure 1: Main Proof Checking Files

```
\prop 4.23 $\Not (x,y,z \in \nats \setdif \{0\}
     \And 2 < n \in \nats
     \And x \toThe n + y \toThe n = z \toThe n)$
\lineb Proof:  We begin by taking as given:
\note 1 $(x,y,z \in \nats \setdif \{0\})$  \By G
\linea and
\note 2 $(2 < n \in \nats)$ \By G
\linea From these givens we get our contradiction that
\note 3 $(x \toThe n + y \toThe n = z \toThe n $
\lined $ \c \false)$ \By .1,.2
\lineb \Bye .3 H .1,.2
```

Figure 2: Fictitious Proof: Source File

4.23 $\neg(x, y, z \in \mathbf{N} \backsim \{0\} \wedge 2 < n \in \mathbf{N} \wedge x \wedge n + y \wedge n = z \wedge n)$

Proof: We begin by taking as given:

.1 $(x, y, z \in \mathbf{N} \backsim \{0\})$ ‡G

and

.2 $(2 < n \in \mathbf{N})$ ‡G

From these givens we get our contradiction that

.3 $(x \wedge n + y \wedge n = z \wedge n$

$\rightarrow \perp)$ ‡.1,.2

Q.E.D. .3 H .1,.2

Figure 3: Fictitious Proof: Output

in the fictitious proof are those used in the rules of inference and common notions files, respectively. In sections 2.1 and 2.2 it is shown how to adapt ProofCheck so that it works with symbols of the author's choosing.

1.2 Macro definitions files

There are TeX definition files with extension `.tdf` or LaTeX definition file with extension `.ldf` that need to be included at the beginning of a checkable source file using a TeX `\input` statement. These include the `utility.tdf` file which contains the proof structuring macros and sets up some math fonts and the `common.tdf` file which contains the TeX definitions of the symbols used in the rules of inference and common notions files. Further, any document which contains many new symbols should also have its own `.tdf` file. Besides TeX macros these files may also contain ProofCheck directives, which are TeX comments beginning with '%' allowing authors to customize the behavior of the system:

ProofCheck directives

- Symbol substitution:
  `%def_symbol \forall \Each`
- Operator precedence specification:
  `%set_precedence \toThe 17`
- External file referencing:
  `%set_ref gr  graphs`
- Primitive term and formula specification:
  `%undefined_term: supremum`
- Term and formula definor specification:
  `%term_definor: =`
- LaTeX theorem section-level specification:
  `%major_unit: section`

1.3 Rules of inference file

The rules of inference file is a TeX file which is searched every time a line of a proof is checked. It consists of logical rules of inference.

The prime example of a rule of inference is the modus ponens rule. In the rules file it looks like this:

$$(\underline{p} \rightarrow \underline{q})\ ;\ \underline{p} \vdash \underline{q}$$

In order for checkable proofs to be of reasonable length it is important that there be many rules. We now have over 1500 in the default `rules.tex` file. This file which has been supplied by default with the package is based on the logic in [1], which is not in wide use. The size of this file makes it difficult to simply edit desired changes. In section 2.5 an improvement is described which makes it possible to obtain a variety of standard and non-standard rules of inference files.

1.4 Operator properties file

When the rules of inference file is searched each rule is compared with the line to be checked using a unifier. A unifier is a program which determines whether there is some substitution which makes two given expressions the "same", which ordinarily means identical. But it is important to allow expressions such as $((A \wedge B) \wedge C)$ and $(A \wedge (C \wedge B))$ to be considered the same.

The unifier may assume that certain operators, such as conjunctions, are commutative and associative, or transitive, such as equality, if theorems to this effect are stored in the `properties.tex` file. This file can be edited and such theorems may be commented out if desired for the logic under consideration.

1.5 Common notions file

The common notions file consists of mathematics which is either taken for granted, or at least is outside the scope of the document.

2 Adapting ProofCheck

It is very desirable that an author be able to check mathematics without being required to use the same symbols as those in the rules of inference and common notions files.

2.1 Example of a symbol definition

In the source file shown in Figure 2, the macro `\setdif` produces the symbol '$\backsim$' in the output file shown in Figure 3. To obtain '$\mathbf{N} \setminus \{0\}$' instead of '$\mathbf{N} \backsim \{0\}$' in the output, while still using '`\setdif`' in the source file one can use the TeX definition:

```
\def\setdif{\setminus}
```

This modification affects only the output file and does not free the author of the need to match the symbol in the source file with the corresponding symbol in the rules or common notions files.

2.2 Example of a symbol substitution

Unlike symbol definition, symbol substitution allows an author to use a freely chosen symbol in the source file without forgoing a match with ProofCheck files. To use '`\setminus`' (for '$\setminus$') in the source file and still match '`\setdif`' in the ProofCheck files an author could use the following symbol substitution:

```
%def_symbol \setminus \setdif
```

In cases where the issue can be resolved by a symbol-for-symbol replacement either a symbol definition or a symbol substitution can solve the problem. In propositional logic for example, since notation is almost universally infix, symbol-for-symbol replacements are enough. Since the rules of inference file consists of logic, and quantifiers like propositional logic share a common syntax, syntactical agreement with the rules of inference file can almost always be accomplished with such replacements. Semantic issues are discussed in section 2.5.

2.3 Setting operator precedence

Infix operators are constants which, whether logical or mathematical, require precedence values in order to avoid fully parenthesizing. To establish the precedence of a new infix operator, or reset that of an old one, a line is inserted into a macro definitions file. To set the precedence of the exponential operator used in Figure 2 to 17 the following line suffices:

```
%set_precedence \toThe 17
```

The `viewdfs` script may be run to see all currently established precedence values.

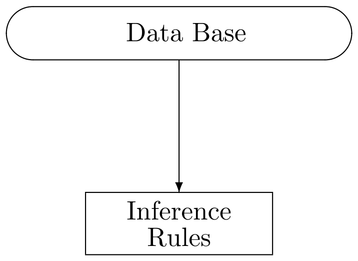

Figure 4: Rules File and Database

2.4 New terms and formulas

Much of the work of making mathematical proofs checkable lies in constructing definitions for the terms and formulas used in the work, which satisfy the requirements of formality. Definitions recognized by ProofCheck must have the form:

`(` ⟨*definiendum*⟩ ⟨*definor*⟩ ⟨*definiens*⟩ `)`

Term and formula definor symbols can be established using the directives:

```
%term_definor:
%formula_definor:
```

The parser then "learns" the author's terms and formulas when it reads a definition. A definition should be marked using `\prop` and given a `\By D` justification.

Terms and formulas may also be presented to the parser without definitions using ProofCheck directives:

```
%undefined_term:
%undefined_formula:
```

2.5 Rules database

The rules of inference used in the original implementation of ProofCheck are those of a logic system consisting of the sentence logic of tautologies and a free predicate logic which allows non-denoting terms and includes definite and indefinite descriptions. Recognizing that most potential users work with more traditional logic, ProofCheck has been modified to allow a user to select a rules of inference file which implements the user's preferred logic system. A database has been built to store rules of inference, where each rule is flagged with respect to the 20 different attributes shown in Figure 5. Rules have been added to the database which are valid only in more traditional logic. The user can create a query to select rules based on these attributes. We have identified 8 logic systems for which queries have been pre-defined. Each of the 8 systems uses tautologies for sentence logic, but differ in the area of predicate logic. The systems are as follows:

Sentence logic
- tau - tautology rule
- trf - rule contains True symbol (⊤) or False symbol (⊥)
- ent - entailment rule (relevance and necessity)
- tui - intuitionistic rule
- mdl - contains a modal operator

Predicate logic
- prd - predicate logic rule (contains a predicate or quantifier)
- std - standard predicate logic rule
- fre - free logic rule
- uni - universal logic rule
- equ - contains the equals symbol (=)
- idn - contains the identity symbol (≡)
- def - contains definite description (the)
- idf - contains indefinite description (an)
- cas - contains case symbol (◊)
- nul - contains the Nul symbol (Nul)
- exs - contains the exists predicate symbol (ex)
- unv - universalization rule

Miscellaneous
- prn - has parentheses as reference punctuators
- mlt - multi-goal rule
- gvh - Given : Hence rule

Figure 5: Attributes of Inference Rules

2.5.1 Eight logic systems

Here, SPL means Standard Predicate Logic, and FPL means Free Predicate Logic.

1. SPL without equality or descriptions (Kelley, Morse)
2. SPL with equality (Suppes)
3. SPL with definite descriptions (Bernays)
4. SPL with indefinite descriptions (Bourbaki)
5. FPL without identity, equality, or descriptions
6. FPL with identity and equality
7. FPL with definite descriptions
8. FPL with indefinite descriptions (Alps–Neveln)

2.5.2 Query examples

For example, to produce the rules of standard predicate logic with equality and without descriptions the following query is used:

```
(mdl=0 And (tau=1 Or (std=1 And def=0)))
```

As another example, to produce rules for Alps–Neveln logic, the needed query is:

```
(mdl=0 And (fre=1 Or tau=1))
```

2.6 The common notions file

Using a different logic affects the validity of theorems compiled in the common notions file. Therefore, these theorems must be reviewed and modified to suit the logic being used.

The work of the authors uses a set/class theory similar to that of Morse and Kelley, and this is reflected in the theorems of the common notions file. The use of a different set theory, such as Zermelo–Fraenkel, would require that the common notions be modified for consistency with the chosen set theory. Work is underway to create a common notions file compatible with SPL with definite descriptions and Zermelo–Fraenkel set theory.

2.6.1 External references in proofs

External references to the common notions file can be identified by a leading '`0`'. In the marginal justification of the following note, a theorem numbered 11.7 in the common notions file is referred to:

.12 $(x \in A \to x \in B)$ ‡011.7; .10

Multiple external reference files are permitted with an appended identification code established using a ProofCheck directive. The command

```
%set_ref gr  graphs
```

causes ProofCheck to refer to the file `graphs.tex` when references using `gr` are used as in the justification of the following note:

.12 $(x \in A \to x \in B)$ ‡11.7gr; .10

3 Conclusion

A widely-held view is that formal proofs are by necessity lengthy and intractable. A fairly recent logic text, for example, claims that:

> In principle, all of known mathematics can be formalized in terms of the symbols and axioms. But in everyday practice, most ordinary mathematicians do not completely formalize their work; to do so would be highly impractical. Even partial formalization of a two-page paper on differential equations would turn into a 50 page paper. For analogy, imagine a cake recipe written by a nuclear physicist, describing the locations and quantities of the electrons, protons, etc., that are included in the butter, the sugar, etc.[1]

Existing proof assistants[2] tend to support this notion, conveying the impression that "computer proofs" are massive, and the packages themselves tend to be massive.

But as in [2] and in [3] we again claim that checkable proofs can be done which are less than an order of magnitude longer than proofs which are considered rigorous by prevailing standards. We seek mathematicians interested in trying ProofCheck. We will gladly provide assistance to anyone seeking to use it.

References

[1] Robert A. Alps and Robert C. Neveln. A predicate logic based on indefinite description and two notions of identity. *Notre Dame Journal of Formal Logic*, 22(3), 1981.

[2] Bob Neveln and Bob Alps. Writing and checking complete proofs in TeX. *TUGboat*, 28(1), 2007, pp. 80–83. `http://tug.org/TUGboat/tb28-1/tb88neveln.pdf`.

[3] Bob Neveln and Bob Alps. ProofCheck: Writing and checking complete proofs in LaTeX. *TUGboat*, 30(2), 2009, pp. 191–195. `http://tug.org/TUGboat/tb30-2/tb95neveln.pdf`.

[4] Eric Schechter. *Classical and Nonclassical Logics*. Princeton University Press, Princeton, New Jersey, 2005.

◇ Bob Neveln
Widener University
`neveln (at) cs dot widener dot edu`

◇ Bob Alps
Evanston, Illinois
`BobAlps (at) aol dot com`

[1] The quoted text is on page 28 of [4].

[2] See Wiedijk's list: `www.cs.ru.nl/~freek/digimath`