ℕaproche Tutorial

Peter Koepke
University of Bonn

2021

1 Introduction

ℕaproche (Natural Proof Checking) is a mathematical proof assistant for texts in the input language ForTheL (Formula Theory Language). ForThel aims to approximate the language, structure and appearance of common mathematical texts; FortheL texts are checked by ℕaproche for syntactic, ontological and logical correctness. ℕaproche is a component of Isabelle PIDE (Proof Interactive Development Environment) which provides comfortable text editing and interactive checking through a continuously running ℕaproche server.

Ideally, a user edits a mathematical text in LaTeX, and ℕaproche automatically checks the correctness of those parts which are in `\begin{forthel}` ... `\end{forthel}` environments and gives feedback to the user, similar to a continuous spellchecker. With the current ℕaproche it is possible to write university-level mathematical texts in a natural language and style whose ForTheL segments are logically verified by ℕaproche. In this tutorial and in example texts which are distributed together with ℕaproche these segments are typeset on a grey background.

ℕaproche is a derivative of the ground-breaking System for Automated Deduction (SAD). ℕaproche still supports the ASCII format `.ftl` of SAD which allows rapid experiments without worrying about LaTeX particulars, and we use that format for a Quick Start and some exercises.

This tutorial is an introduction to the principles and use of the ℕaproche prover. After the Quick Start and some general information we explain ForTheL commands and statements and the structuring of ForTheL texts along a proof of the infinitude of primes. Actually the tutorial is a version of Euclid's proof, preceded by necessary number-theoretic and set-theoretic preliminaries. After studying this material, a reader should be able to understand the other example texts and start writing similar formalizations.

Let us, however, warn the reader that writing formally complete and unambiguous mathematical texts in a readable natural language is subtle and requires ample exercise, as one may already guess by looking at legal texts.

We suggest that the tutorial is read as pdf and simultaneously followed interactively in the Isabelle PIDE (Proof Interactive Development Environment), with the source `TUTORIAL.ftl.tex` opened in one buffer, using other buffers for experiments and exercises. Moreover one may use a LaTeX compiler for displaying `.ftl.tex` formalizations.

2 Quick Start: $1 + 2 = 3$

Let us prove our first theorem in Isabelle/Naproche:

- Download and install Isabelle2025 from `https://isabelle.in.tum.de/` (this may take several minutes).
- Start Isabelle jEdit which may again some time and start editing a new file `temp.ftl`; `.ftl` files are recognized as ForTheL files and are continuously checked by Isabelle/Naproche.
- Copy the following bit of number theory into `temp.ftl`:

```
[synonym number/numbers]
Signature. A natural number is a mathematical object.
Signature. 0 is a natural number.
Signature. 1 is a natural number.
Signature. Assume that k,l are natural numbers.
k + l is a natural number.
Definition. 2 = 1 + 1.
Definition. 3 = 2 + 1.
Axiom. Assume that k,l,m are natural numbers.
(k+l)+m=k+(l+m).
Theorem. 1 + 2 = 3.
```

- The Output window of Isabelle/Naproche displays the checking process, ending in `verification successful` and some statistics. Further feedback is given by coloured highlighting of the buffer and by ballons when hovering over sentences with the mouse pointer.
- The theorem $1 + 2 = 3$ has been verified!

2.1 Explanation of the ForTheL Text

The above text consists of a sequence of ForTheL "commands" that steer its parsing and logical processing. The ForTheL language is designed in a way that appears natural to human mathematicians although it is a completely formal language like a programming language.

- `[synonym number/numbers]` is a linguistic command which identifies the identifiers `nummber` and `nummbers` in the subsequent text. This *allows* to use grammatically correct singular or plural forms. On the other hand ℕaproche does not enforce grammatical correctness.

- `Signature` commands define and extend the mathematical "vocabulary" of the text. The command
  `Signature. A natural number is a mathematical object.`
  has the following effects:

  - The noun phrase "natural number" is added to the vocabulary of the parser; by the previous `[synonym ...]` command one can also write "natural numbers" instead.
  - Internally, a unary predicate symbol `aNaturalNumber` is generated, to be used by the first-order logical processing. One can see this symbol by hovering the mouse over the command: a pop-up balloon will show a translation to a first-order formula which contains the subformula `aNaturalNumber(v0)` with a variable `v0` inserted.
  - The first-order translation
    ```
    forall v0 ((HeadTerm :: aNaturalNumber(v0))
    implies aObject(v0))
    ```
    is added to internal representation of the text. Note that ℕaproche has an inbuilt predicate `aObject` to encompass all mathematical objects to be used in a text. This predicate is addressed by the phrases "object" or "mathematical object". The first-order translation expresses that the collection of natural numbers is a subcollection of the collection of all mathematical objects.
  - The newly introduced predicate is tagged by `HeadTerm`, to be used in further processing.

- `Signature. 0 is a natural number.` extends the language by a new constant symbol `0` which is intended to be a natural number. This can

be seen in the translation: `forall v0 ((HeadTerm :: v0 = 0) implies aNaturalNumber(v0))`.

- `Signature. Assume that k,l are natural numbers. k + l is a natural number.`
  generates an internal binary function symbol `+`. The translation shows that the (universally quantified) variables `k` and `l` are natural numbers, and that the resulting term `k+l` is a natural number. So the `+` operation is of type $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$, where $\mathbb{N}$ is the standard notation for the class / set / type of natural numbers.

- The command `Definition. 2 = 1 + 1.` also extends the language. A new constant symbol `2` is added together with an obvious definitional equivalence `forall v0 ((HeadTerm :: v0 = 2) iff v0 = 1+1)`.

- The only axiom of our small number theory is added by the command
  `Axiom. Assume that k,l,m are natural numbers. (k+l)+m=k+(l+m).`
  The first-order translation includes the guards
  `((aNaturalNumber(k) and aNaturalNumber(l)) and aNaturalNumber(m))`
  for the universally quantified variables, as explained above.

- Finally `Theorem. 1 + 2 = 3.` requests that the statement $1 + 2 = 3$ be proved. The system sends the previous first-order translations as "hypotheses" and the thesis "$1+2 = 3$" as "conjecture" to the external automated theorem prover (ATP) in the first-order format TPTP:

```
fof(m_,hypothesis,( ! [W0] : (aNaturalNumber(W0) => aObject(W0)))).
fof(m_,hypothesis,aNaturalNumber(sz0)).
fof(m_,hypothesis,aNaturalNumber(sz1)).
fof(m_,hypothesis,( ! [W0] : ( ! [W1] : ((aNaturalNumber(W0) &
      aNaturalNumber(W1)) => aNaturalNumber(sdtpldt(W0,W1)))))).
fof(m_,hypothesis,((sz2 = sdtpldt(sz1,sz1)) & ( ! [W0] :
      ((W0 = sdtpldt(sz1,sz1)) => (W0 = sz2))))).
fof(m_,hypothesis,((sz3 = sdtpldt(sz2,sz1)) & ( ! [W0] :
      ((W0 = sdtpldt(sz2,sz1)) => (W0 = sz3))))).
fof(m_,hypothesis,( ! [W0] : ( ! [W1] : ( ! [W2] :
      (((aNaturalNumber(W0) & aNaturalNumber(W1)) & aNaturalNumber(W2))
      => (sdtpldt(sdtpldt(W0,W1),W2) = sdtpldt(W0,sdtpldt(W1,W2)))))))).
fof(m__,conjecture,(sdtpldt(sz1,sz2) = sz3)).
```

- Naproche uses internal reasoning and the eprover ATP to derive the conjecture along the equality chain:

$$1 + 2 = 1 + (1 + 1) = (1 + 1) + 1 = 2 + 1 = 3,$$

as one could see from the eprover proof output.

2.2 Exercises

Exercise. Extend the above number theory by further constants and prove equalities like $2 + 3 = 5$. What about $2 + 2 = 5$? What about $0 + 0$? Can you prove that $2 + 2 \neq 5$? (In the ftl-language, $\neq$ is expressed by the ASCII combination `!=`.)

Exercise. One can define the order relation $\leq$ in our number theory by

```
Definition. k \leq l iff there exists a natural number m
such that k + m = l.
```

Which axioms for partial or linear orders can now be proved about `\leq`?

Exercise. (Pretyping of variables) In our number theory, the variables `k,l,m` in the definition of + or in the associativity axiom are typed by explicit assumptions. Instead one can "pretype" variables until the end of a file by a command of the form

```
Let k,l,m denote natural numbers.
```

Eliminate the type assumptions by pretyping. What happens, if a variable does not have a previous type assumption?

Exercise. Experiment with your `.ftl` texts. What happens if one exchanges `Definition` by `Signature` and vice versa?

3 ForTheL and LaTeX

ForTheL is intended to be faithful to the mathematical language found in mathematical books and papers. These are usually typeset in LaTeX which in particular allows the rendering of complex symbolism. This motivates the integration of ForTheL into LaTeX. Currently, Naproche accepts two dialects of the Formula Theory Language: the ASCII-based legacy version indicated by the .ftl file ending, as used above, and the LaTeX-oriented version with a .ftl.tex file ending. The latter can be immediately processed by LaTeX for high-quality mathematical typesetting.

The development of Naproche is directed towards the LaTeX-format, also because typesetting contains semantic information that may be exploited in the proof checking process. In LaTeX, e.g., the variable "a" is distinguished from the word "a" by an italic font. One may also introduce custom LaTeX environments to signal specific processing of parts of texts. Translating `.ftl`-files to `.ftl.tex` is usually straightforward. ForTheL-constructs like `Definition. ...` or `Theorem. ...` obviously correspond to definition or theorem environments in LaTeX.

In this tutorial, we mainly use the `.ftl.tex`-format. The ASCII-format will, however, be supported for some time since it allows for experimentation with ℕaproche without the effort of correct LaTeX syntax. Whereas finished ℕaproche formalizations should be written in LaTeX format and be accompanied by pdf-printouts, the ASCII format might be particularly important in didactical situations which emphasize logical and mathematical content.

3.1 Natural Numbers in .ftl.tex

We introduce the language of arithmetic in the LaTeX dialect: start a new file with the LaTeX-ending `.ftl.tex`, containing:

```
\begin{forthel}

[synonym number/numbers]

\begin{signature}  A natural number is a mathematical object.
\end{signature}

Let $m,n$ denote natural numbers.
\begin{signature} $0$ is a natural number.
\end{signature}

Let $x$ is nonzero stand for $x \neq 0$.

\begin{signature} $1$ is a nonzero natural number.
\end{signature}

\begin{signature} $m + n$ is a natural number.
\end{signature}

\begin{signature} $m * n$ is a natural number.
\end{signature}
\end{forthel}
```

This file which will also be the beginning of our formalization of Euclid's theorem on the infinitude of primes.

At the same time it is a LaTeX file which can by typeset as a LaTeX document in the context of an adequate LaTeX preamble

```
\documentclass[11pt]{article}
```

```
\usepackage{amssymb}
\usepackage{exercises}
\usepackage{url}
\usepackage{naproche}

\begin{document}
.....
.....
\end{document}
```

The LaTeXstyle file `../lib/tex/naproche` defines the forthel environment so that ForTheL content, i.e., the logical core of a text, is typeset against a light grey background:

[synonym number/numbers]

Signature 1. A natural number is a mathematical object.

Let m, n denote natural numbers.

Signature 2. 0 is a natural number.

Let x is nonzero stand for $x \neq 0$.

Signature 3. 1 is a nonzero natural number.

Signature 4. $m + n$ is a natural number.

Signature 5. $m * n$ is a natural number.

Exercise. Convert the initial formalization of arithmetic to the LaTeX format.

Exercise. Experiment with the LaTeX formalization to see whether greek letters, letters in other fonts like bold, fracture, ..., other symbols, possibly with decorations like π^+ or $\tilde{\otimes}$ can be used as variables or operation symbols.

3.2 Literate Formalizations

Since only content in

```
\begin{forthel} ... \end{forthel}
```

environments is passed to Naproche one can write arbitrary material outside those environments. This allows to insert arbitrary comments, including sectioning commands or even pictures, into `.ftl.tex` formalizations.

As was mentiioned before, the LaTeX source `TUTORIAL.ftl.tex` of this document is a proper ForTheL text which proof-checks successfully in Naproche. One can experiment with this text by loading it into the Isabelle-PIDE and editing the forthel environments. It may be convenient to deactivate some of these environments by replacing the outer `\begin` and `\end` by `begin` and `end`.

Exercise. Augment the LaTeXformalization of arithmetic so that it prettyprints with a title, abstract, some comments, and a small bibliography whilst still being a correct ForTheL formalization.

Exercise. Write a formalization of the real numbers as a ring in a language that has $+, \cdot, -, 0, 1$. You may use the ASCII format (observe that in that format, lines starting with a `#` are interpreted as comment).

4 General Principles of Naproche

4.1 Natural Language Processing

Naproche interprets ForTheL input texts in first-order logic and also proposes proof methods to be used by the internal reasoner of Naproche or by the external ATP. ForTheL leverages a number of natural language mechanisms to capture formal content in a compact, user-friendly and natural way. This corresponds to usual natural language features, where the phrase "white horse that belongs to Mary" with its adjective, noun and relative sentence corresponds to a first-order statement like

$$horse(x) \wedge white(x) \wedge property - of(x, Mary)$$

with a (hidden) variable x, predicates $horse()$, $white()$, and $property{-}of(,)$, and a constant $Mary$. Naproche extracts this formal context whilst reading the input sentence by sentence. Previous sentences provide the context of already introduced language components, in which the new sentence is to be interpreted.

Indeed the "white horse" can be formalized in ForTheL. We use the ASCII format, since we are mainly developing a language.

```
Signature. A human being is a notion.
Signature. Mary is a human being.
Signature. An animal is a notion.
Signature. A horse is an animal.

Let A denote an animal. Let H denote a human being.
```

```
Signature. A is white is an atom.
Signature. A belongs to H is an atom.
```

From these premisses we can show the following triviality:

```
Theorem. Every white horse that belongs to Mary
is white.
```

Exercise. Inspect this natural language formalization: what is the meaning of "notion" and "atom". Why can't you replace "Every" by "Some" in the Theorem. What is the meaning of the error message for the theorem "Mary is white"? Change the example so that horses and Mary can all be potentially white.

Exercise. Formalize the classical syllogism

> All men are mortal.
> Socrates is a man.
> Therefore, Socrates is mortal.

in ℕaproche.

4.2 Axiomatic Approach

The ℕaproche system comes with a minimal set of in-built mathematical notions. Usually one has to explicitly extend the first-order language through Signature and Definition commands and through Axioms. Then Lemmas and Theorems can be postulated and proved with familiar proof structures. In the following this procedure is explained along a standard proof of the infinitude of prime numbers:

- set up a language and axioms for natural number arithmetic;
- define divisibility and prime natural numbers;
- introduce some set theory so that one can define finite sets, sequences and products.

Finally, a checked natural language proof of Euclid's theorem can be carried out in this axiomatic setup.

5 Importing General Preliminaries

The Naproche system comes along with a small inbuilt language which includes the notions of (mathematical) object, classes, sets, maps and functions:

> Elements of classes or sets are objects, maps and functions map objects to objects. Sets are classes which are objects. Similarly functions are maps which are objects.

Basic properties can be checked in the system. E.g.,

Theorem 6. Every element of every class is an object.

Proof. Let C be a class. Let x be an element of C. Then x is an object. □

Note that the inbuilt premisses are not available to the external prover but only to the inbuilt *reasoner* of Naproche. Since the reasoner is rather weak, we have to give an explicit proof: instantiate the universal quantifiers with "arbitrary but fixed" instances C and x and prove the claim in this context. Note that we use the LaTeX proof environment also as ForTheL proof environment.

The file `preliminaries.ftl.tex` proves important properties of the inbuilt notions, similar to the above example, and it postulates further axioms about them. More details can be found in the commentary parts of the file. The file also includes a list `vocabulary.ftl` of singular/plural pairs like the above `[synonym number/numbers]`, to be used for grammatical correctness. Moreover, some alternative phrases for certain mathematical phrases are imported with `macros.ftl`.

These preliminaries are imported by:

```
[read examples/preliminaries.ftl.tex]
```

This file also provides the $\subseteq$-relation between classes, so that we can prove:

Theorem 7. Let C be a class. Then $C \subseteq C$.

Exercise. Prove that $\subseteq$ is a transitive relation, and that the empty set is a subclass of every other class. One can also denote the empty set symbolically

by using the linguistic command:
Let $\emptyset$ denote the empty set.
Exercise. The preliminaries file also provides the notion $F(x)$ for elements $x \in \mathrm{dom}(F)$ where F is a function or map. Formalize the property that a function is injective and that the composition of two injective functions is injective.

6 On the Translation from Natural to First-Order Language

Above we have introduced the functions $+$ and $*$ of addition and multiplication (of natural numbers) and the constants 0 and 1. Domains are captured by unary relations. The type "natural number" of ordinary mathematical discourse is modeled by the internal unary relation symbol `aNaturalNumber`, and the arithmetic functions and quantifiers are restricted to the extension of the unary relation symbol. So the (weak) type system of ordinary mathematical language is modeled by a system of first-order predicates. These types do not follow any strict "type theory" with specific mathematical laws but they are powerful enough to organize the universe of mathematics.

6.1 First-Order Translation

The ForTheL code for introducing the type, or rather notion, of natural numbers, the constants 0 and 1 and the operations of $+$ and $*$ has the following first-order translation which can be found in the output window of jEdit or hovering the mouse over the sentence:

```
1. forall v0 ((HeadTerm :: aNaturalNumber(v0)) implies
     aObject(v0))
2. forall v0 ((HeadTerm :: v0 = 0) implies aNaturalNumber(v0))
3. forall v0 ((HeadTerm :: v0 = 1) implies
     (aNaturalNumber(v0) and not v0 = 0))
4. (aNaturalNumber(m) and aNaturalNumber(n))
5. forall v0 ((HeadTerm :: v0 = m+n) implies aNaturalNumber(v0))
6. (aNaturalNumber(m) and aNaturalNumber(n))
7. forall v0 ((HeadTerm :: v0 = m*n) implies aNaturalNumber(v0))
```

In these formulas we see the newly introduced first-order symbols:
`aNaturalNumber(v0), 0, 1, +, *`.

The first-order translations follow a certain idiom which is favourable for the overall processing. Formula 1 exhibits the new symbol marked by the tag `HeadTerm`. Similarly formula 2 emphasizes the symbol `0` which would

not have been the case in the equivalent `aNaturalNumber(0)`. Note that 5 and 7 both have the premises

`(aNaturalNumber(m) and aNaturalNumber(n))`

for the two arguments of the operations.

The correctness of mathematical texts includes the principle of *ontological correctness* which in this case means that such premises have to be proved before the operations can reasonably be applied within a proof. Naproche checks texts sentence by sentence. The first check is for the indicated ontological correctness of the sentence. Only if that is successful the logical correctness will be checked if the sentence is an assertion of a mathematical fact; Assumptions, or statements in signature or definition commands will be treated differently.

Exercise. Extract the first-order translation of one of your formalizations. You can put the command `[dump on]` into your ForTheL text to increase the amount of information in the output buffer of Isabelle.

6.2 Some ForTheL Commands and Keywords

Let us now go through the natural language phrases used to reach this translation. New first-order symbols are spawned by Signature commands. The new notion comes before the keyword "is" after which the new notion is classified as a new type ("is a notion") or as a member of of an existing type ("is a natural number").

The phrase before "is" is read as a new language pattern that the parser learns. A pattern has some word tokens, like "natural", "number", or some symbolic tokens, like "0", "1", "+", "∗". In between those tokens a pattern may have holes for the insertion of terms, which in the Signature command are indicated by previously introduced variables, like "m" or "n". These were introduced in the parser command "Let m, n denote natural numbers." Thereafter, m and n are variables which are "pretyped" to be natural numbers. With that,

Signature 8. $m + n$ is a natural number.

has the "double translation"

```
(aNaturalNumber(m) and aNaturalNumber(n))
forall v0 ((HeadTerm :: v0 = m+n) implies aNaturalNumber(v0))
```

where the first (or more) formulas are premises and the last contains the newly introduced symbol.

We can also qualify the typing on the right-hand side of the "is" keyword by first-order formulae. In our example, we have introduced a pattern for a first-order formula by the parser command "Let x is nonzero stand for $x \neq 0$." This formula is then applied as an adjective in the next Signature command

Signature 9. 1 is a nonzero natural number.

Note that some natural language processing is also taking place: "nonzero" is introduced within the phrase "x is nonzero" in an adjective position. So in the Signature command, "nonzero" can be used as an adjective which modifies "natural number". The first-order effect of this is a conjunction

```
3. forall v0 ((HeadTerm :: v0 = 1) implies
     (aNaturalNumber(v0) and not v0 = 0))
```

The equality "=" and inequality "≠" are predefined phrases with corresponding first-order symbols.

6.3 "Grammar"

Note that we have also used "linguistic" commands: the command `[synonym number/numbers]` identifies the tokens "number" and "numbers", providing the plural form. The command can be abbreviated to `[synonym number/-s]`. This is a simple linguistic "hack" which allows grammatically correct forms. But it also allows wrong ones, and it is up to the user to make the right choices.

7 Postulating Natural Number Axioms

We need to introduce axioms for our abstract first-order structure. Axioms are ForTheL statements written in axiom environments. For arithmetic we use self-explanatory symbolic formulas. There are many ways of axiomatizing the natural numbers in order to be able to prove our final goal: the infinitude of primes. Here we axiomatize the natural numbers as a sort of commutative "half-ring" with 1. We provide ourselves with another variable l, pretyped as a natural number.

Let m, n stand for natural numbers.

Axiom 10. $m + n = n + m$.

Let l stand for a natural number.

Axiom 11. $(m + n) + l = m + (n + l)$.

Axiom 12. $m + 0 = m = 0 + m$.

Axiom 13. $m * n = n * m$.

Axiom 14. $(m * n) * l = m * (n * l)$.

Axiom 15. $m * 1 = m = 1 * m$.

Axiom 16. $m * 0 = 0 = 0 * m$.

Axiom 17. $m*(n+l) = (m*n)+(m*l)$ and $(n+l)*m = (n*m)+(l*m)$.

Axiom 18. If $l + m = l + n$ or $m + l = n + l$ then $m = n$.

Axiom 19. Assume that l is nonzero. If $l * m = l * n$ or $m * l = n * l$ then $m = n$.

Axiom 20. If $m + n = 0$ then $m = 0$ and $n = 0$.

Axioms - like Signatures - are toplevel sections which consist of $n + 1$ statements. The first n are assumption statements ("Assume ...", "Let ...") under which the final statement is postulated. Note that pretypings of variables also act like assumptions.

8 The Natural Order - Defining Relations and Functions

Definitions extend the first-order language by defined symbols as in the following examples concerning the ordering of the natural numbers. A definition corresponds to a Signature command in which a symbol is introduced plus an Axiom containing the defining property.

Let m, n stand for natural numbers.

Definition 21. $m \leq n$ iff there exists a natural number l such that $m + l = n$.

Let $m < n$ stand for $m \leq n$ and $m \neq n$.

Definition 22. Assume that $n \leq m$. $m - n$ is a natural number l such that $n + l = m$.

The first definition defines the binary relation $\leq$ by an "iff" equivalence. This is followed by a purely syntactic definition of $<$. $m < n$ is simply an abbreviation for another formula. Such abbreviations and aliases are already

expanded during parsing, possibly recursively. The third definition defines the binary difference function $-$.

Exercise. Introduce aliases for numbers and arithmetic operations so that one can prove: "The sum of one and three is equal to five".

Exercise. Introduce the square m^2 of a natural number and prove the binomial formula for $(m+n)^2$.

8.1 Axiomatic Content in Definitions

Definitions of functions and constants usually contain implicit postulates corresponding to the existence and uniquess-properties of function values and constants. In the above definition of the difference $(-)$, the condition for l should be satisfiable by a unique natural number. This is however *not* checked by ℕaproche, so that the well-definedness of the function is the user's responsibility. If the function definition were non-unique we could have a contradictory system of assumptions. Consider, e.g., the wrong definition

Definition 23. Assume that $n \leq m$. $m-n$ is a natural number l such that $n = m$.

The first-order translation would be

```
(aNaturalNumber(m) and aNaturalNumber(n))
n\leq m
forall v0 ((HeadTerm :: v0 = m-n)
   iff (aNaturalNumber(v0) and n = m))
```

Every number fits the defining equivalence provided that $m = n$. But then $0 = 0 - 0 = 1$, contradiction.

With relation definitions, such problems do not arise.

9 Lemmas and Theorems

After setting up the axiomatics we proceed to claim and prove propositions. Claims together with the accumulated facts are given to the background ATP (= eprover). Many basic propositions can be proved by the ATP without further intervention. The following three lemmas show that $\leq$ is a partial order:

Let l, m, n stand for natural numbers.

Lemma 24. $m \leq m$.

Axiom 25. If $m \leq n \leq m$ then $m = n$.

Lemma 26. If $m \leq n \leq l$ then $m \leq l$.

9.1 Eprover in the Background

These lemmas are checked correct by Naproche without explicit proofs. We can look at the tasks given to the ATP by putting a [dump on] command in the beginning of the ForTheL parts of the document and looking for the dump of the provertask in the output window. The task is written in the first-order logic language TPTP which is a standard input language for ATPs. Observe that all previous Signature, Axiom and Definition commands can be found as premises of the conjecture.

The following is part of the trace of proving $m \leq m$. Actually the ontological correctness of the formula is checked first. In case of the $\leq$-relation, the assumptions or presuppositions were given by the pretypings of both arguments. So for the formula $m \leq m$ we have to show that m is a natural number. This prover task is expressed by the following TPTP text:

```
[Translation] (line 409 of ...
aNaturalNumber(m)
[Translation] (line 409 of ...
m\leq m
[Reasoner] (line 409 of ...
goal:  m \leq m .
[Main] (line 409 of ...
fof(m_,hypothesis,$true).
fof(m_,hypothesis,aNaturalNumber(sz0)).
fof(m_,hypothesis,(aNaturalNumber(sz1) & ( ~ (sz1 = sz0)))).
fof(m_,hypothesis,( ! [W0] : ( ! [W1] : ((aNaturalNumber(W0) & aNaturalNumber(W1))
  => aNaturalNumber(sdtpldt(W0,W1))))).
fof(m_,hypothesis,( ! [W0] : ( ! [W1] : ((aNaturalNumber(W0) & aNaturalNumber(W1))
  => aNaturalNumber(sdtasdt(W0,W1))))).
fof(m_,hypothesis,( ! [W0] : ( ! [W1] : ((aNaturalNumber(W0) & aNaturalNumber(W1))
 => (sdtpldt(W0,W1) = sdtpldt(W1,W0))))).
fof(m_,hypothesis,( ! [W0] : ( ! [W1] : ( ! [W2] : (((aNaturalNumber(W0) &
  aNaturalNumber(W1)) & aNaturalNumber(W2))
  => (sdtpldt(sdtpldt(W1,W2),W0) = sdtpldt(W1,sdtpldt(W2,W0)))))))).
fof(m_,hypothesis,( ! [W0] : (aNaturalNumber(W0)
  => ((sdtpldt(W0,sz0) = W0) & (W0 = sdtpldt(sz0,W0)))))).
fof(m_,hypothesis,( ! [W0] : ( ! [W1] : ((aNaturalNumber(W0) & aNaturalNumber(W1))
  => (sdtasdt(W0,W1) = sdtasdt(W1,W0))))).
fof(m_,hypothesis,( ! [W0] : ( ! [W1] : ( ! [W2] : (((aNaturalNumber(W0) &
  aNaturalNumber(W1)) & aNaturalNumber(W2))
  => (sdtasdt(sdtasdt(W1,W2),W0) = sdtasdt(W1,sdtasdt(W2,W0)))))))).
fof(m_,hypothesis,( ! [W0] : (aNaturalNumber(W0)
  => ((sdtasdt(W0,sz1) = W0) & (W0 = sdtasdt(sz1,W0)))))).
fof(m_,hypothesis,( ! [W0] : (aNaturalNumber(W0)
  => ((sdtasdt(W0,sz0) = sz0) & (sz0 = sdtasdt(sz0,W0)))))).
fof(m_,hypothesis,( ! [W0] : ( ! [W1] : ( ! [W2] : (((aNaturalNumber(W0)
  & aNaturalNumber(W1)) & aNaturalNumber(W2))
  => ((sdtasdt(W1,sdtpldt(W2,W0)) = sdtpldt(sdtasdt(W1,W2),sdtasdt(W1,W0)))
```

```
  & (sdtasdt(sdtpldt(W2,W0),W1) = sdtpldt(sdtasdt(W2,W1),sdtasdt(W0,W1)))))))).
fof(m_,hypothesis,( ! [W0] : ( ! [W1] : ( ! [W2] : (((aNaturalNumber(W0)
  & aNaturalNumber(W1)) & aNaturalNumber(W2))
  => (((sdtpldt(W0,W1) = sdtpldt(W0,W2))
  | (sdtpldt(W1,W0) = sdtpldt(W2,W0))) => (W1 = W2))))))).
fof(m_,hypothesis,( ! [W0] : (aNaturalNumber(W0) => (( ~ (W0 = sz0))
  => ( ! [W1] : ( ! [W2] : ((aNaturalNumber(W1) & aNaturalNumber(W2))
  => (((sdtasdt(W0,W1) = sdtasdt(W0,W2))
  | (sdtasdt(W1,W0) = sdtasdt(W2,W0))) => (W1 = W2)))))))).
fof(m_,hypothesis,( ! [W0] : ( ! [W1] : ((aNaturalNumber(W0) & aNaturalNumber(W1))
  => ((sdtpldt(W0,W1) = sz0) => ((W0 = sz0) & (W1 = sz0))))))).
fof(m_,hypothesis,( ! [W0] : ( ! [W1] : ((aNaturalNumber(W0) & aNaturalNumber(W1))
  => (sdtbszlzezqdt(W0,W1)
  <=> ( ? [W2] : (aNaturalNumber(W2) & (sdtpldt(W0,W2) = W1)))))))).
fof(m_,hypothesis,( ! [W0] : ( ! [W1] : ((aNaturalNumber(W0) & aNaturalNumber(W1))
  => (sdtbszlzezqdt(W1,W0) => ((aNaturalNumber(sdtmndt(W0,W1))
  & (sdtpldt(W1,sdtmndt(W0,W1)) = W0)) & ( ! [W2] : ((aNaturalNumber(W2)
  & (sdtpldt(W1,W2) = W0)) => (W2 = sdtmndt(W0,W1)))))))))).
fof(m__,hypothesis,aNaturalNumber(xm)).
```

9.2 Testing for Contradictions

It is quite common to accidentally introduce trivial inconsistencies in formalizations. Not just by function definitions, but also because some marginal cases outside the main argument have not been treated right. E.g., although the number 0 is quite uninteresting for the study of prime numbers, we still have to deal with 0-cases or explicitly request that terms are nonzero. If a text with non-trivial mathematical content checks unexpectedly fast then one should become suspicious, because eprover may exploit a contradiction to prove every proposition via *ex falso quodlibet*.

To find inconsistencies it is helpful to try to prove

Lemma 27. Contradiction.

in various places of a text. If the lemma is validated by ℕaproche then one has to investigate further. Contradiction- lemmas can be quickly deactivated by commenting out with `%`. They can also be used to force rechecking of the text: uncomment the lemma and then comment it again; this will lead to rechecking from the position of the lemma onwards.

10 Linear and Discrete Orders

We need more axiomatic assumptions for the ordering of the natural numbers. The axioms so far do not guarantee that the ordering is linear. So we continue:

Let l, m, n stand for natural numbers.

Axiom 28. $m \leq n$ or $n < m$.

Lemma 29. Assume that $l < n$. Then $m+l < m+n$ and $l+m < n+m$.

Lemma 30. Assume that m is nonzero and $l < n$. Then $m * l < m * n$ and $l * m < n * m$.

11 Induction

Naproche has inherited an elegant treatment of induction from the SAD system. Naproche has a special binary relation symbol $\prec$ for a universal inductive relation: if the property P is *inductive* along $\prec$, i.e., if at any point m property P is inherited at m provided all $\prec$-predecessors of m satisfy P, then P holds everywhere. More formally the general induction principle for the relation $\prec$ reads:

To prove $\forall k \phi(k)$, it suffices to prove inductivity along $\prec$:

$$\forall v_0(\forall v_1(v_1 \prec v_0 \to \phi(v_1)) \to \phi(v_0)).$$

This modification of proof tasks is immediately carried out by the parser when it comes across the keyword "proof by induction".

Induction along the relation $<$ on natural numbers is introduced axiomatically by embedding the $<$-relation into $\prec$:

Let m, n stand for natural numbers.

Axiom 31. If $n < m$ then $n \prec m$.

Let us consider a proof by induction that $\leq$ is a "discrete" order with nothing strictly between 0 and 1.

Lemma 32. For every n $n = 0$ or $1 \leq n$.

Proof by induction on n. □

The automatic translation of the statement of the lemma is

```
forall v0 (aNaturalNumber(v0) implies
((InductionHypothesis ::
forall v1 (aNaturalNumber(v1) implies (iLess(v1,v0) implies
(v1 = 0 or 1\leq v1))))
implies (v0 = 0 or 1\leq v0)))
```

The formula expresses that the property "$n = 0$ or $1 \leq n$" is *inductive*. Proving the universal formula "For every n $n = 0$ or $1 \leq n$" has been reduced to proving inductivity. In this case the inductivity task can be carried out automatically by eprover.

Remark: It seems difficult to prove this inductivity by a short intuitive proof. Apparently the above axiomatization is not standard and should be replaced by an approach where the natural numbers are generated from 0 by a +1-operation, expressed by using $n \prec n + 1$ as induction axiom.

With the previous lemma we can show:

Lemma 33. If $m \neq 0$ then $n \leq n * m$.

12 Division

Prime (natural) numbers are defined via divisibility.

Let l, m, n stand for natural numbers.

Definition 34. n divides m iff for some l $m = n * l$.

Let $x|y$ denote x divides y. Let a divisor of x denote a natural number that divides x.

Lemma 35. Assume $l|m|n$. Then $l|n$.

The definition is similar to the definition of $\leq$. Note, however, the possible syntactic variations: "there exists a natural number l such that $m = n * l$"; it is also possible to put the quantifier after the property: "n divides m iff $m = n * l$ for some l".

Natural language has many mechanisms for putting information into sentences in a compact, un-formalistic way. Un-formalistic means, e.g., that natural language does not normally use brackets (...) in speech. "The syntax and semantics of the ForTheL language" by Andrei Paskevich is still a good guide to most ForTheL constructs, although the language has been extended since then.

13 An Interactive Proof

The final contradiction in Euclid's proof will need:

Lemma 36. Let $l|m$ and $l|m + n$. Then $l|n$.

Naproche does not find a proof on its own: depending on some default timeouts the proof search is abandoned, and the goal $l|n$ fails. In Isabelle-Naproche this is signaled in the output window, and the failed goal is underlined in red.

So the user has to "interactively" supply a proof, which in a first approximation is a list of statements leading up to the claim, and which Naproche's ATP is able to prove successively. Proof statements can also introduce assumptions and new variables to the argument, and they can structure the proof.

Let l, m, n stand for natural numbers.

Lemma 37. Let $l|m$ and $l|m + n$. Then $l|n$.

Proof. Assume that l is nonzero. Take a natural number p such that $m = l * p$. Take a natural number q such that $m + n = l * q$.

Let us show that $p \leq q$. Proof by contradiction. Assume the contrary. Then $q < p$. $m + n = l * q < l * p = m$. Contradiction. qed.

Take $r = q - p$. We have $(l * p) + (l * r) = l * q = m + n = (l * p) + n$. Hence $n = l * r$. □

When Naproche encounters a statement immediately followed by an explicit proof then Naproche defers proving the statement and first goes through the proof. Since proofs may contain subproofs, this process may take place recursively.

Proofs of a "toplevel" Lemma or Theorem use the
`\begin{proof}...\end{proof}`
environment well-known from LaTeX. In our proof there is also a "lowlevel" proof of $p \leq q$ indicated by "Let us show that". We discuss some aspects of the proof:

- Most sentences in a proof are statements, or statements extended by constructs that organize the flow of the argument.
- "Assume that l is nonzero." is an assumption that introduces the premise "l is nonzero" to the argument. Instead of "Assume that" one could also use variants like [let us | we can] (assume | presume | suppose) [that].
- "Take p such that $m = l * p$." introduces a new variable p with a specific property to the argument. To verify this construct the prover

has to show the existence of some object satisfying the property. Again there are variants: [let us | we can] (choose | take | consider).

- "Let us show that $p \leq q$." claims that the statement $p \leq q$ holds and announces a subsequent proof. Alternatives: [let us | we can] (prove | show | demonstrate) (that).

- "Proof by contradiction" denotes the start of an indirect proof. It is recommended to explicitly mark indirect proofs. Note that in the example there is a "lowlevel" proof that uses a simple

  ```
  Proof [by ...](.) ... (qed. | end.)
  ```

  environment instead of the LaTeXproof environment.

- Other proof methods are "by cases" and "by induction".

- "Assume the contrary.": The contrary is the negation of the current thesis which in this case is the statement claimed just before. "thesis" denotes the current thesis, "contradiction" stands for "false".

- "Then $q < p$.": Words like "then", "hence", "thus", "therefore", "consequently" are filler words which are redundant for ℕaproche but may help human readers to understand the text.

- "$m + n = l * q < l * p = m$": binary relations like "=" or "<" can be chained. The statement means the conjunction of the single relations. These will be checked from left to right.

- "Contradiction. qed.": The indirect proof has reached the desired contradiction, and the (local) proof environment is closed by "qed.".

ℕaproche is able to prove the next lemma without an explicit proof in the text.

Lemma 38. Let $m|n \neq 0$. Then $m \leq n$.

14 Primes

Prime numbers are defined as usual. Indeed we define the adjective "prime" which will enable us to write "prime natural number" or "prime divisor".

Let m, n stand for natural numbers. Let x is nontrivial stand for $x \neq 0$ and $x \neq 1$.

Definition 39. n is prime iff n is nontrivial and for every divisor m of n $m = 1$ or $m = n$.

The following lemma obviously holds by induction: either k is prime itself, or k has a divisor strictly between 1 and k; by induction that divisor has a prime divisor which is also a prime divisor of k.

Lemma 40. Every nontrivial m has a prime divisor.

Proof by induction on m. □

"Proof by induction" transforms the thesis into the inductivity theses:

```
thesis: forall v0 ((aNaturalNumber(v0) and (not v0 = 0 and not v0 = 1))
implies ((InductionHypothesis :: forall v1 ((aNaturalNumber(v1) and
(not v1 = 0 and not v1 = 1)) implies (iLess(v1,v0)
implies exists v2 ((aNaturalNumber(v2) and doDivides(v2,v1))
and isPrime(v2))))) implies exists v1
((aNaturalNumber(v1) and doDivides(v1,v0)) and isPrime(v1))))
```

which can be discharged automatically by eprover.

15 Sets and Classes

Some basic properties of sets and classes are built into ℕaproche or are explicitly postulated in the `preliminaries.ftl.tex` file. We shall use these to express the infinitude of the collection of prime natural numbers.

Arbitrary collections of objects can be defined naturally in ForTheL:

Definition 41. ℕ is the collection of all natural numbers.

This translates as

```
forall v0 ((HeadTerm :: v0 = \mathbb{N}) iff
(aClass(v0) and forall v1
(aElementOf(v1,v0) iff (aNaturalNumber(v1) and aObject(v1)))))
```

The `aObject(v1)` is part of the collection construct and ensures that only objects will be elements of the collection. Such collections are automatically registered as *classes* (`aClass(v0)`). There are a few grammatical extensions

and variants of this construct: in particular one could form the collection of all [terms] such that [condition], where [condition] is an arbitrary ForTheL statement.

Classes can also be defined by abstraction terms, i.e., we could have defined alternatively:

Definition 42. $\mathbb{N} = \{n | n \text{ is a natural number}\}$.

To work freely with the class of natural numbers one wants it be a *mathematical object* itself, i.e., a *set*. The axioms so far do not imply this, as one can check by trying to prove "$\mathbb{N}$ is a set." So we postulate this as an axiom which is (equivalent to) the well-known axiom of infinity in set theory:

Axiom 43. $\mathbb{N}$ is a set.

We define the set of prime numbers so that we can prove its infinity.

Definition 44. $\mathbb{P} = \{n | n \text{ is a prime natural number}\}$.

The axiom of separation from `preliminaries.ftl.tex` yields:

Lemma 45. $\mathbb{P}$ is a set.

since $\mathbb{P}$ is a subclass of the *set* $\mathbb{N}$.

Exercise. Form the collection of all even numbers or of all square numbers and prove that these are sets.

Exercise. Prove that the class $\{x | x \text{ is a set and } x \notin x\}$ is *not* a set.

16 Finite Sequences and Products

We shall use natural numbers and bounded intervals of natural numbers to "count" the elements of sets.

Let m, n stand for natural numbers.

Definition 46. $\{m, \ldots, n\} = \{i \in \mathbb{N} | m \leq i \leq n\}$.

This translates to

```
forall v0 ((HeadTerm :: v0 = \Seq{m}{n}) iff ((aClass(v0) and
(aSet(\mathbb{N}) implies aSet(v0))) and
forall v1 (aElementOf(v1,v0) iff ((aElementOf(v1,\mathbb{N})
and (m\leq v1 and v1\leq n)) and aObject(v1)))))
```

Note that the second line of the translation contains an instance of the axiom of separation: if the class that bounds the variable i is a set then the resulting class is a set. Thus:

Lemma 47. $\{m, \ldots, n\}$ is a set.

The notation $\{m, \ldots, n\}$ indicates the notational and typographical potential for ℕaproche texts if one uses appropriate LaTeXmacros.

$\{m, \ldots, n\}$ is printed using the macro

```
\newcommand{\Seq}[2]{\{#1,\dots,#2\}}
```

The macro is used in the source code of the above definition:

```
\begin{definition} $\Seq{m}{n} = \{ i \in \mathbb{N} |
m \leq i \leq n \}$.
\end{definition}
```

ℕaproche reads this as the introduction of a binary function pattern `\Seq{m}{n}` as one can see from the first-order translation.

In this way, intuitive and customary ...-notations which may seem vague can be given an unambiguous meaning and can be used in ℕaproche formalizations.

These techniques are used in further definitions and notations for finite sequences.

Definition 48. A sequence of length n is a function F such that $\mathrm{dom}(F) = \{1, \ldots, n\}$.

The members $F(i)$ of a sequence F are often written in an indexed notation F_i. This is just an alias at the parsing level:

Let F_i stand for $F(i)$.

Definition 49. Let F be a sequence of length n. $\{F_1, \ldots, F_n\} = \{F_i | i \in \mathrm{dom}(F)\}$.

This definition uses the LaTeX-macro

```
\newcommand{\Finset}[3]{\{#1_{#2},\dots,#1_{#3}\}}
```

Dot notation is also used for iterations of all sorts. For Euclid's theorem we shall want to consider products of finitely many prime numbers. So we postulate axiomatically:

Signature 50. Let F be a sequence of length n such that $\{F_1, \ldots, F_n\} \subseteq \mathbb{N}$. $F_1 \cdots F_n$ is a natural number.

Axiom 51. Let F be a sequence of length n such that $F(i)$ is a nonzero natural number for every $i \in \mathrm{dom}(F)$. Then $F_1 \cdots F_n$ is nonzero and $F(i)$ divides $F_1 \cdots F_n$ for every $i \in \mathrm{dom}(F)$.

Note that we can name toplevel sections by single words like "Factorproperty" or numbers. These can be referenced later in the form "(by Factorproperty)".

17 Finite and Infinite Sets

Finite sequences readily allow a formalization of finiteness for arbitrary sets and classes.

Let S denote a class.

Definition 52. S is finite iff $S = \{F_1, \ldots, F_n\}$ for some natural number n and some function F that is a sequence of length n.

Definition 53. S is infinite iff S is not finite.

18 Euclid's Theorem

Now everything is in place for the proof that there are infinitely many prime numbers.

Signature 54. $\mathbb{P}$ is the class of prime natural numbers.

Theorem 55 (Euclid). $\mathbb{P}$ is infinite.

Proof. Assume that r is a natural number and p is a sequence of length r and $\{p_1, \ldots, p_r\}$ is a subclass of $\mathbb{P}$.

(1) p_i is a nonzero natural number for every $i \in \mathrm{dom}(p)$.

Consider $n = p_1 \cdots p_r + 1$. $p_1 \cdots p_r$ is nonzero. Hence n is nontrivial. Take a prime divisor q of n.

Let us show that $q \neq p_i$ for all natural numbers i such that $1 \leq i \leq r$.

Proof by contradiction. Assume the contrary. Take a natural number i such that $1 \leq i \leq r$ and $q = p_i$. q is a divisor of n and q is a divisor of

$p_1 \cdots p_r$ (by Factorproperty,1). Thus q divides 1. Contradiction. qed.

Hence $\{p_1, \ldots, p_r\}$ is not the class of prime natural numbers. $\square$