An Introduction to ℕaproche, Along a Formalization of Euclid's Proof of the Infinitude of Primes

Peter Koepke
University of Bonn

February 20, 2021

1 About This Document

1.1 Running ℕaproche in Isabelle

This pdf-document can be read as an introduction to the ℕaproche natural proof assistant. It contains a proof of the infinitude of primes, typeset on a light-grey background, and explains various features of the ForTheL input language to ℕaproche with examples from that proof.

Simulataneously, the LaTeX source `Introduction.ftl.tex` of this document is a ForTheL text which proof-checks successfully in the ℕaproche prover. The source text can be loaded into the Isabelle Proof Interactive Development Environment (Isabelle-PIDE), in which the text can be edited and continuously checked for correctness. Isabelle 2021 contains ℕaproche as a component which is activated when files with the extension .ftl.tex are edited.

Note that the checking process consists of proving a large number of explicit and implicit first-order properties of the ForTheL text by the automatic theorem prover E. The power of E and of the overall ℕaproche system strongly depends on hardware performance. It may be necessary to supply more proof details to get a text checked. On an older laptop with an Intel Pentium N3710 processor ℕaproche takes around 75 seconds to check this text.

1.2 The ForTheL Language and LaTeX

Since only content in

`\begin{forthel} ... \end{forthel}`
environments is passed to Naproche one can write arbitrary material outside those environments (like these explanatory comments). To experiment with Naproche one can edit the forthel environments. For this it may be convenient to deactivate most of these environments by replacing the outer `\begin` and `\end` by `begin` and `end`.

By default, Naproche continuously checks the active buffer for correctness as a ForTheL text (parsing) and for logical correctness (proof checking). Checking results are displayed in the jedit Output window. Further feedback is given by coloured highlighting of the buffer and by ballons when hovering over sentences with the mouse pointer.

ForTheL files with a proper LaTeXpreamble and a begin document / end document structuring can be immediately compiled by TeX/LaTeXonce the forthel environment is defined. This has, e.g. been defined in the style file `naproche` which we are using for this document. This package also provides the gray background for ForTheL.

1.3 An Example

We demonstrate this principle with a small technical section mainly concerned with notation. The `[...]` commands in this section give some parser commands to Naproche which we shall discuss later.

The source code of the following section is:

```
\section{Notation}

\begin{forthel}


%\begin{lemma} Contradiction. \end{lemma}

[printprover on]

[synonym number/-s]
[synonym divide/-s]
[synonym set/-s] [synonym element/-s]
[synonym belong/-s] [synonym subset/-s]

\end{forthel}
```

Here, `\section{Notation}` is an ordinary LaTeXsection command. The

rest of the snippet is in a `begin{forthel} ... end{forthel}` environment which is marked in the LaTeXoutput by a gray background:

2 Notation

[printprover on][dump on]
[synonym number/-s] [synonym divide/-s] [synonym set/-s] [synonym element/-s] [synonym belong/-s] [synonym subset/-s]

3 General Remarks

The Naproche system (Natural Proof Checking) is a proof assistant that accepts input texts in a controlled natural language for mathematics, and checks them for logical correctness. Naproche accepts two dialects of the ForTheL language (Formula Theory Language): a classic one close to the language of the original SAD system, indicated by the .ftl file ending, and a LaTeX-oriented version with .ftl.tex file ending. In this note we use the latter version, which directly allows mathematical typesetting with LaTeX.

Text processing by Naproche translates accepted texts in a natural mathematical language into a formal logic format which is then handed over to further processing in the formal logic. In the present Naproche system, the logic format is classical first-order predicate logic and the processing is mainly by repeated sledgehammer-like proving using the eprover automatic theorem prover (ATP). Other logic formats like Lean or possibly Isabelle's Isar language are under consideration.

3.1 Natural Language Processing

The ForTheL input text is thus interpreted in the target logic and also proposes proof methods to be used by the reasoner and the 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.

3.2 Axiomatic Approach

The ℕaproche system comes with a minimal set of in-built mathematical notions. Usually on 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.

4 Natural Numbers - Introducing the Language of Arithmetic

A first-order language is determined by its (symbols for) relations, functions, and constants. We want to introduce the functions $+$ and $*$ of addition and multiplication (of natural numbers) and the constants 0 and 1. Since we shall later consider other domains as well, like sets and functions, we capture those domains by relations. The type "natural number" of ordinary mathematical discourse will be modeled by an internal unary relation symbol `aNaturalNumber`, and the arithmetic functions and quantifiers will be 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 a strict "type theory" with specific mathematical laws but they are still powerful enough to organize the universe of mathematics.

Here is ForTheL code for introducing the type, or rather notion, of natural numbers, the constants 0 and 1 and the operations of $+$ and $*$.

In order to be able to form classes and sets of natural numbers we agree that natural numbers are small objects.

Let x is small stand for x is setsized.

Signature 1. A natural number is a small object.

Let i, k, l, m, n, p, q, r denote natural numbers.

Lemma 2. i is small.

Signature 3. 0 is a natural number.

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

Signature 4. 1 is a nonzero natural number.

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

Signature 6. $m * n$ is a natural number.

4.1 First-Order Translation

The first-order translation of sentences in the ForTheL code can be found in the output window of jedit or hovering the mouse over the sentence:

```
1. forall v0 ((HeadTerm :: aNaturalNumber(v0)) implies
     (truth and isSetsized(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 is 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. These premises have to be proved before the operations can reasonably be applied within proofs.

4.2 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 i, k, l, m, n, p, q, r denote natural numbers." Thereafter, m and n are variables which are "pretyped" to be natural numbers. With that,

Signature 7. $m + n$ 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 8. 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.

4.3 "Grammar"

Note that we have also used the "linguistic" commands of the notation section: 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.

5 Natural Numbers - Postulating Axioms

We now have to introduce axioms for our abstract first-order structure. Axiom 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.

Axiom 9. $m + n = n + m$.

Axiom 10. $(m + n) + l = m + (n + l)$.

Axiom 11. $m + 0 = m = 0 + m$.

Axiom 12. $m * n = n * m$.

Axiom 13. $(m * n) * l = m * (n * l)$.

Axiom 14. $m * 1 = m = 1 * m$.

Axiom 15. $m * 0 = 0 = 0 * m$.

Axiom 16. $m*(n+l) = (m*n)+(m*l)$ and $(n+l)*m = (n*m)+(l*m)$.

Axiom 17. If $l + m = l + n$ or $m + l = n + l$ then $m = n$.

Axiom 18. Assume that l is nonzero. If $l * m = l * n$ or $m * l = n * l$ then $m = n$.

Axiom 19. 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 previous pretypings of variables also act like assumptions.

6 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.

Definition 20. $m \leq n$ 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 21. 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. The abbreviation is already expanded, possibly recursively, by the parser. The third definition defines the binary difference function $-$.

6.1 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 case of the function definition 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 would have a contradictory system of assumptions. Consider, e.g., the wrong definition

Definition 22. 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.

For relation definitions, these problems do not arise.

7 Lemmas and Theorems

After setting up the axiomatics we proceed to claim and prove properties. Claims together with the accumulated axioms will be 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:

Lemma 23. $m \leq m$.

Lemma 24. If $m \leq n \leq m$ then $m = n$.

Lemma 25. If $m \leq n \leq l$ then $m \leq l$.

7.1 Eprover in the Background

These lemmas were checked correct by ℕaproche without explicit proofs. We can look at the task 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 environments can be found as premises of the conjecture $m \leq m$.

```
[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)).
```

7.2 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.

To find inconsistencies it is helpful to try to prove

Lemma 26. Contradiction.

in various places of a text. If the lemma is validated by ℕaproche then one has to investigate further. In the source text of this document one finds an Contradiction-Lemma which is commented out by `%`. This can be quickly activated for a contradiction check. It can also be used to force rechecking of the text: uncomment the lemma and then comment it again; this will lead to rechecking at least from the position of the lemma onwards.

8 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. Also we want a "discrete" order with nothing strictly between 0 and 1. So we continue:

Axiom 27. $m \leq n$ or $n < m$.

Lemma 28. Assume that $l < n$. Then $m+l < m+n$ and $l+m < n+m$.

Lemma 29. Assume that m is nonzero and $l < n$. Then $m*l < m*n$ and $l * m < n * m$.

Axiom 30. $n = 0$ or $n = 1$ or $1 < n$.

Lemma 31. If $m \neq 0$ then $n \leq n * m$.

9 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 at any point m property P is inherited at m provided all $\prec$-predecessors of m satisfy P, then P holds everywhere.

So to prove that P holds universally, it suffices to prove the "inheritance" along $\prec$. This modification of proof tasks is already carried out by the parser when it comes across the keyword "proof by induction". This will be demonstrated in a later proof in this Introduction.

Axiom 32. If $n < m$ then $n \prec m$.

From this axiom one can derive Peano axioms for the natural numbers. On the other hand, with some simple axioms about the successor operation $+1$ and with $\prec$ one could have derived all the above structural axioms.

10 Division

We now get to notions that are crucial for the study of divisors and prime numbers:

Definition 33. 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.

The definition is similar to the definition of $\leq$. Note, however, the possible syntactic variations: "there exists a natural number l such that $m + l = n$", "for some l $m = n * l$". 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 generally allow brackets (...) in speech. ForTheL implements several of these natural language mechanisms. Although the language has evolved, "The syntax and semantics of the ForTheL language" by Andrei Paskevich is still a good guide to most constructs of the language.

11 An Interactive Proof

We shall later need a technical lemma on divisibility:

Lemma 34. Let $l|m$ and $l|m + n$. Then $l|n$.

On the computer I am using, 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 gets an underlining in red.

So the user has to "interactively" supply a proof, which in a first approximation is a list of statements which leads 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.

Lemma 35. Let $l|m$ and $l|m + n$. Then $l|n$.

Proof. Assume that l is nonzero. Take p such that $m = l * p$. Take 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". Let us discuss some aspects of the proof:

- Most sentences in a proof are statements, or statements extended by certain 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 proof. Note that in the example this 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 that proof environment is closed by "qed.".

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

Lemma 36. Let $m|n \neq 0$. Then $m \leq n$.

12 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 x is nontrivial stand for $x \neq 0$ and $x \neq 1$.

Definition 37. n is prime iff n is nontrivial and for every divisor m of n $m = 1$ or $m = n$.

12.1 Proof by Induction

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 38. Every nontrivial k has a prime divisor.

Proof. Proof by induction. □

The phrase "proof by induction" invokes a general induction principle for the relation $\prec$. To prove $\forall k \phi(k)$, it suffices to prove:

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

So "proof by induction" transforms the thesis into a new thesis:

```
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))))
```

Note that internally, `iLess` represents the relation $\prec$. Since we had postulated axiomatically that $<$ is a subrelation of $\prec$, the induction principle for $\prec$ implies a standard induction principle for the natural numbers.

13 Classes

"sets" and "classes" are built-in notions of Naproche, as well as "element of ...". "element of y" determines the type of elements-of-y. Such "of"-types lead to several linguistic modifications: one can quantify over elements-of-y like (for all — for some — no) (element of y); y has some element; etc.

Similarly, the subclass relation is given by the dependent type of subclasses-of-T.

Let S, T stand for classes. Let x belongs to S stand for x is an element of S.

Definition 39. A subclass of S is a class T such that every element of T belongs to S.

Let $T \subseteq S$ stand for T is a subclass of S.

To avoid the classical antinomies of set theory, classes can only have "small" elements which in Naproche's terminology are "setsized"; both these adjective were identified earlier in the document. We extend the built-in ontology of Naproche according to the following principles:

Axiom 40. Every element of every class is small.

Axiom 41. Every set is a small class and every small class is a set.

Classes can be naturally formed in ForTheL:

Definition 42. $\mathbb{N}$ is the class of natural numbers.

The verbose form is equivalent to the use of abstraction terms:

Definition 43. $\{m, \dots, n\} = \{$ natural number $i | m \leq i \leq n\}$.

14 Finite Sequences and Products, using Intuitive "..."-Notation

This section demonstrates how some notation that is considered vague can be interpreted as formally exact. Mathematics often uses ...-notations to indicate arbitrary size finite or even infinite mathematical objects.

From a LaTeXstandpoint, a notation like $\{m, \dots, n\}$ can canonically be seen as the printout of a corresponding macro in the LaTeXsource. Naproche on the other hand accepts standard LaTeXmacros as patterns, so that the macro can be a properly introduced Naproche pattern with a first-order

definition. In this way, intuitive and customary notation can be used also as Naproche input.

In the present text, $\{m, \ldots, n\}$ is printed from a macro defined by:
`\newcommand{\Seq}[2]{\{#1,\dots,#2\}}`.

This notation or macro can be given a precise semantics by ForTheL definitions.

Definition 44. $\{m, \ldots, n\}$ is the class of natural numbers i such that $m \leq i \leq n$.

So far there are basically no axioms for set formation, so we postulate:

Axiom 45. $\{m, \ldots, n\}$ is a set.

The macro for the $\{m, \ldots, n\}$- notation is visible in the LaTeXcode:

```
\begin{definition} $\Seq{m}{n}$ is the class of
natural numbers $i$ such that $m \leq i \leq n$.
\end{definition}
```

14.1 Functions

The notion of "function" and some related notations like the domain of a function F or the application $F(x)$ of a function to an argument are provided by Naproche. We add an axiom about smallness of values:

Axiom 46. Assume F is a function and $x \in Dom(F)$. Then $F(x)$ is small.

Definition 47. A sequence of length n is a function F such that $Dom(F) = \{1, \ldots, n\}$.

The members $F(i)$ of a sequence F are often written in an indexed notation f_i where this notation is defined by a macro
`\newcommand{\val}[2]{#1_{#2}}`.
The ForTheL semantics is defined by:

Let F_i stand for $F(i)$.

Definition 48. Let F be a sequence of length n. $\{F_1, \ldots, F_n\} = \{F_i | i \in Dom(F)\}$.

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 49. 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 50. (Factorproperty) Let F be a sequence of length n such that $F(i)$ is a nonzero natural number for every $i \in Dom(F)$. Then $F_1 \cdots F_n$ is nonzero and $F(i)$ divides $F_1 \cdots F_n$ for every $i \in Dom(F)$.

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

15 Finite and Infinite Sets

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

Definition 51. 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 52. S is infinite iff S is not finite.

16 Euclid's Theorem

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

Signature 53. $\mathbb{P}$ is the class of prime natural numbers.

Theorem 54. (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 such that $1 \leq i \leq r$.

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

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

Proof by contradiction. Assume that $q = p_i$ for some natural number i such that $1 \leq i \leq r$. 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. □