Writing Proofs in Naproche

by Peter Koepke

April 2021

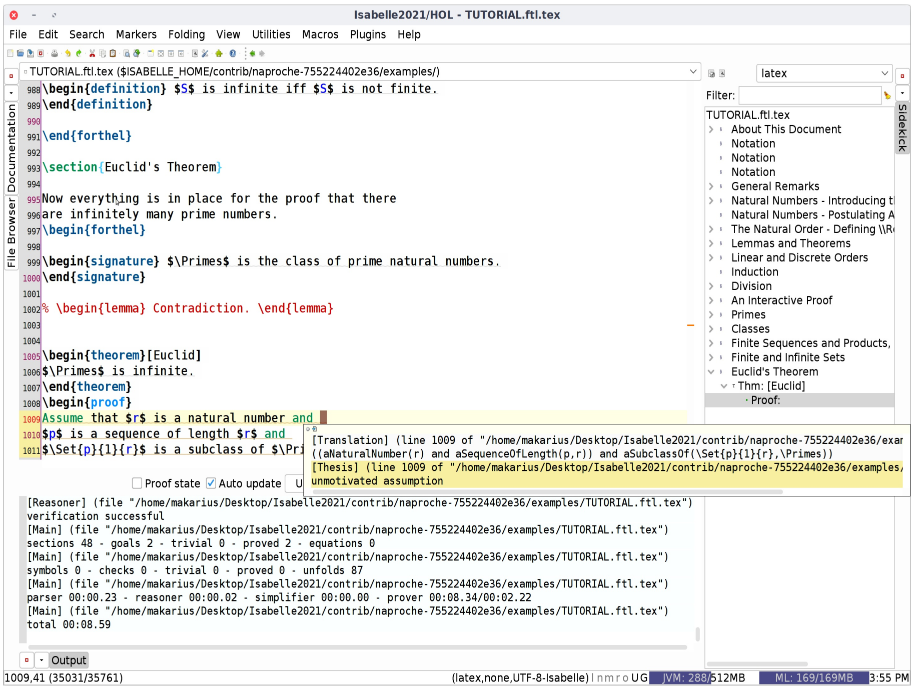
1 Introduction

The Naproche system (*N*atural *Pro*of *Che*cking) accepts input texts in the artifical language ForTheL (Formula Theory Language) which approximates the ordinary language of mathematics. The language of mathematics combines natural language with symbolic terms. ForTheL uses a simple English as its natural language. There are 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 by LaTeX.

This document is an introduction to the *N*aproche natural proof assistant. Towards the end it contains a proof of the infinitude of primes.

2 Running Naproche in Isabelle

Naproche is available as a component of the Isabelle prover environment. Isabelle can be downloaded and installed from `https://isabelle.in.tum.de/`. Editing files with a `.ftl` or `.ftl.tex` file extension in the Isabelle editor will activate continuous checking of the input buffer by Naproche.



```
\begin{definition} $S$ is infinite iff $S$ is not finite.
\end{definition}

\end{forthel}

\section{Euclid's Theorem}

Now everything is in place for the proof that there
are infinitely many prime numbers.
\begin{forthel}

\begin{signature} $\Primes$ is the class of prime natural numbers.
\end{signature}

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


\begin{theorem}[Euclid]
$\Primes$ is infinite.
\end{theorem}
\begin{proof}
Assume that $r$ is a natural number and
$p$ is a sequence of length $r$ and
$\Set{p}{1}{r}$ is a subclass of $\Pri
```

```
[Translation] (line 1009 of "/home/makarius/Desktop/Isabelle2021/contrib/naproche-755224402e36/exar
((aNaturalNumber(r) and aSequenceOfLength(p,r)) and aSubclassOf(\Set{p}{1}{r},\Primes))
[Thesis] (line 1009 of "/home/makarius/Desktop/Isabelle2021/contrib/naproche-755224402e36/examples/
unmotivated assumption
```

```
[Reasoner] (file "/home/makarius/Desktop/Isabelle2021/contrib/naproche-755224402e36/examples/TUTORIAL.ftl.tex")
verification successful
[Main] (file "/home/makarius/Desktop/Isabelle2021/contrib/naproche-755224402e36/examples/TUTORIAL.ftl.tex")
sections 48 - goals 2 - trivial 0 - proved 2 - equations 0
[Main] (file "/home/makarius/Desktop/Isabelle2021/contrib/naproche-755224402e36/examples/TUTORIAL.ftl.tex")
symbols 0 - checks 0 - trivial 0 - proved 0 - unfolds 87
[Main] (file "/home/makarius/Desktop/Isabelle2021/contrib/naproche-755224402e36/examples/TUTORIAL.ftl.tex")
parser 00:00.23 - reasoner 00:00.02 - simplifier 00:00.00 - prover 00:08.34/00:02.22
[Main] (file "/home/makarius/Desktop/Isabelle2021/contrib/naproche-755224402e36/examples/TUTORIAL.ftl.tex")
total 00:08.59
```

Naproche feedback is available through the output buffer, hovering the mouse over the input, or colored text highlighting.

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 *N*aproche system strongly depends on hardware performance. It may be necessary to supply more proof details to get a text checked. Text may well take several minutes for checking.

3 The ForTheL Language and LaTeX

We work with the `.ftl.tex` version which allows immediate mathematical typesetting through LaTeX. Since only content in
`\begin{forthel} ... \end{forthel}`
environments is passed to $\mathbb{N}$aproche one can write arbitrary material outside those environments. Documents should thus be structured like

```
\begin{document}
...
[Arbitrary text.]
\begin{forthel}
...
[ForTheL commands and statements]
...
\begin{forthel}
...
\begin{forthel}
...
\end{forthel}
...
\end{document}
```

To experiment with $\mathbb{N}$aproche 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`.

ForTheL files with a proper LaTeXpreamble and a `\begin{document}` ... `\end{document}` structuring can be immediately compiled by TeX/LaTeX once 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 provides a gray background for the active ForTheL parts.

4 Natural Language Processing

The ForTheL input text is interpreted in the target logic and it 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

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

with a (hidden) variable x, predicates horse(), white(), and $\mathrm{property} - \mathrm{of}(,)$, and a constant Mary. $\mathbb{N}$aproche 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.

5 Axiomatic Approach

The $\mathbb{N}$aproche 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.

6 Defining Notions

In mathematical English, words often attain meanings different from their ordinary usage and the same word may have different meanings in different areas of mathematics. A "natural number" is a specific mathematical notion which is introduced and defined in the language of arithmetic. Its meaning may have some vague connection to the ordinary English word "natural", but this is not necessarily so. A "natural transformation" in category theory involves a completely different usage of the word "natural".

Similarly symbols may have different meanings in different contexts. "0" may denote the natural number 0, or a zero matrix, or a certain "neutral element" in all sorts of structures.

Naproche comes with the input language ForTheL whose "vocabulary" has to be extended by **Signature** and **Definition** commands to introduce notions and constants required in a formalization.

Natural numbers and a specific natural number 0 can be introduced by the signature commands:

Signature. *A natural number is a notion.*

Signature. 0 *is a natural number.*

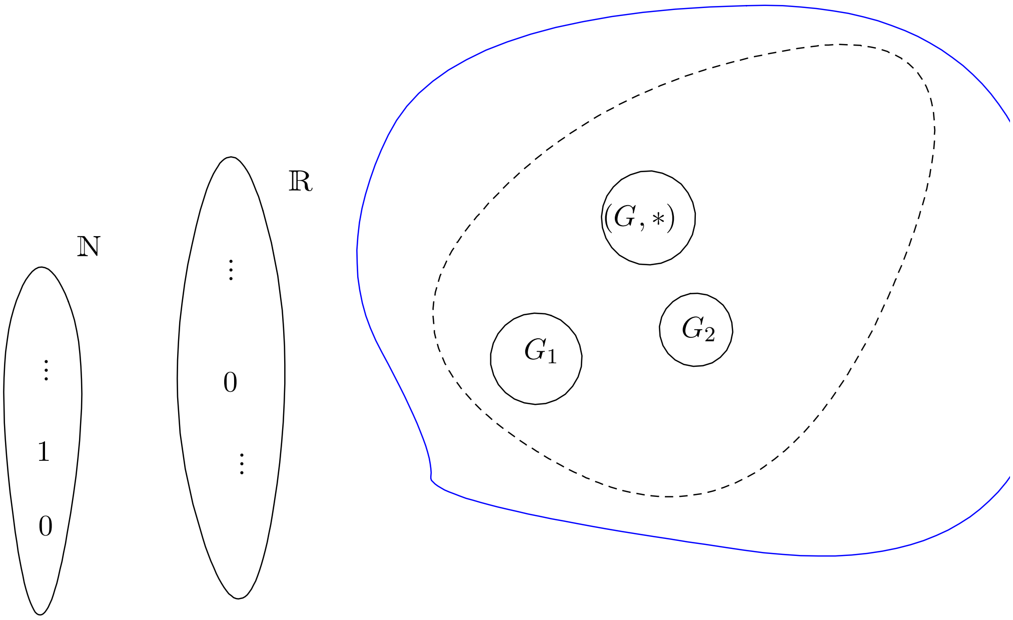
Remark 1. For matrices one would instead declare:

Signature. *A matrix is a notion.*

Signature. 0 *is a matrix.*

Since ForTheL terms and statements are intended to be uniquely readable identifiers like 0 should not be "overloaded". After these four signature commands together, a statement like $0=0$ would be rejected by Naproche as an ambiguity error.

The effects of the introduction of *natural numbers* and of 0 can be viewed as delineating a subdomain $\mathbb{N}$ in a universe of all mathematical objects and picking a specific object in the subdomain.



We thus imagine a universe of mathematical objects of which we cut out certain domains for closer investigation: Besides the standard structures like $\mathbb{N}$ and $\mathbb{R}$ the picture suggests several groups $G, G_1, G_2, \ldots$, which themselves could be collected together into, e.g., a category of groups (dotted line). And we could collect together all categories to study the general notion of a category (blue line).

The LaTeX source of the two signature commands are:

```
\begin{document}
\begin{forthel}

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

\begin{signature}
$0$ is a natural number.
\end{signature}

\end{forthel}
\end{document}
```

The internal effect of these commands can be inspected in the output window:

```
[Translation] (line 5 of "/home/koepke/TEST1/Temp/temp02.ftl.tex")
forall v0 ((HeadTerm :: aNaturalNumber(v0)) implies truth)
[Translation] (line 9 of "/home/koepke/TEST1/Temp/temp02.ftl.tex")
forall v0 ((HeadTerm :: v0 = 0) implies aNaturalNumber(v0))
```

So a unary internal property `aNaturalNumber(.)` and a constant symbol `0` have been introduced together with the type information that `0` is a natural number. This is expressed by the second translation; leaving away some system orientated information this is basically the logic formula:

$$\forall v_0 (v_0 = 0 \rightarrow \mathrm{aNaturalNumber}(v_0)).$$

6.1 Natural Language Aspects of Signature Commands

When Naproche encounters the two commands

> **Signature.** *A natural number is a notion.*
>
> **Signature.** 0 *is a natural number.*

the system first detects the kind of command given, and then looks for patterns of words and symbols in particular positions of the command.

The first command is of the form

> **Signature.** *A ... is a notion.*

This command expects a new pattern of words and symbols in the ... position, and a word is simply a sequence of alphabetical symbols. For the software, the particular choice of words is completely irrelevant as long that it leads to well-distinguished identifiers for further use. So could have used a "Hilbertian variant" like

> **Signature.** *A Bierseidel is a notion.*

> **Signature.** 0 *is a Bierseidel.*

Or wild letter combinations like AbcDE. The choice of words is a question of mathematical style and is an obligation of the user of the system.

There are many grammatical simplifications in the system: articles like "a" and "the" are sometimes (but not always, see above) ignored. Plural forms are treated as singular, etc. Words in patterns are internally transformed to all lowercase, and again the correct choice in the text is a question of grammatical correctness. To get a feel for such identifications, one should experiment with texts and look at their translations and behaviour.

Grammatical flexions of various kinds can be used via synonyms which are introduced by synonym commands to the parser. Writing

> [synonym number/numbers]

identifies the two words "number" and "numbers" so that the correct form singular or plural form of "natural number" can be inserted where needed. To use this for the pattern "natural number", the synonym command has to appear before the pattern introduction.

One can also declare aliases for all sorts of terms by commands of the form

> Let an ... stand for ... .

. We could, e.g., use "integer" instead of "natural number" as follows:

> [synonym number/numbers]

> **Signature.** *A natural number is a notion.*

> Let an integer stand for natural numbers.

> **Signature.** 0 *is a natural number.*

The symbolic pattern in the last command can be much more complex than just 0 . We could for example use the constant $\tilde{\sigma}$ (`$\tilde{\sigma}$`):

> **Signature.** $\tilde{\sigma}$ *is an integer.*

Note that this command for the introduction of a constant is of the form

> **Signature.** ... *is [an]* ....

This command is slightly different from

> **Signature.** *The* ... *is [an]* ....

If one writes

Signature. *Zero is an integer.*

then this introduces a symbolic constant that can only be written as "Zero" and not as "zero". On the other hand

Signature. *The zero is an integer.*

introduces a word (pattern) "zero" that can also be written as "Zero".

The treatment of natural language features in ForTheL and $\mathbb{N}$aproche is at the same time crude and subtle. It is modeled after the mathematical literature and should give good results on average. To get a better feeling, one should experiment with the system, or, even better, understand the behaviour by reading the code.

Exercise 1. Introduce constants ONE, TWO, THREE to the natural numbers, that have to be written exactly in this form. Alternatively introduce a word "one" which can also be capitalized at the beginning of a sentence.

Exercise 2. Introduce vector spaces by fixing a domain of scalars and a domain of vectors. Moreover introduce a scalar 0 and a zero vector $\vec{0}$. Provide aliases so that the $\vec{0}$ can also be called the "zero vector". How does $\mathbb{N}$aproche react if one wants to prove an equality or inequality between a scalar and a vector?

Exercise 3. Provide a language adequate for the Boolean algebra $\{\top, \bot\}$ of truth values $\top = \mathrm{TRUE}$ and $\bot = \mathrm{FALSE}$.

7 Equality and Inequality Statements

ForTheL provides the symbols $=$ (equality) and $\neq$ (inequality). This allows to form statements like $t_1 = t_2$ and $t_1 \neq t_2$, where t_1, t_2 are terms "known" to $\mathbb{N}$aproche. Let us start setting up the natural numbers with constants $0, 1, 2$:

Signature. *A natural number is a notion.*

Signature. 0 *is a natural number.*

Signature. 1 *is a natural number.*

Signature. 2 *is a natural number.*

Some basic knowledge about equalities is available in the proving mechanisms of $\mathbb{N}$aproche. So we formulate our first theorems and try:

Theorem 2. $0 = 0$.

***Theorem* 3.** $0 \neq 0$.

***Theorem* 4.** *not* $0 \neq 0$.

***Theorem* 5.** $0 = 1$.

***Theorem* 6.** $0 \neq 1$.

The theorems translate to:

$$\begin{gathered} 0=0 \\ \text{not}\, 0=0 \\ \text{not}\,\text{not}\, 0=0 \\ 0=1 \\ \text{not}\, 0=1 \end{gathered}$$

These claims are given to an automated theorem prover (ATP) as conjectures that have to be proven from current assumptions. One can see these "prover tasks" by putting the command `[dump on]` before the theorem to be proved. In case of Theorem 2 the output buffer contains:

```
[Translation] (line 27 of "/home/koepke/TEST1/Temp/temp.ftl.tex")
0 = 0
[Reasoner] (line 27 of "/home/koepke/TEST1/Temp/temp.ftl.tex")
goal:  0 = 0 .
[Main] (line 27 of "/home/koepke/TEST1/Temp/temp.ftl.tex")
fof(m_,hypothesis,$true).
fof(m_,hypothesis,aElement(sz0)).
fof(m_,hypothesis,aElement(szi)).
fof(m__,conjecture,(sz0 = sz0)).
```

So the parser which translates the input to formal logic translates to `0 = 0`.

The reasoner identifies prover tasks, i.e., goals to be proved: `0 = 0`.

Then the main $\mathbb{N}$aproche program sends the four bottom lines to the automated theorem prover in a special format; `fof` stands for "first-order formula"; lines with `hypothesis` have been assumed or proved earlier like `aElement(sz0)` from the signature introduction of the constant 0; note that identifiers are sometimes modified for programming reasons; lines with `conjecture` have to be proved by the ATP. The ATP can signal success or failure which will then be signaled to the user in various ways.

Theorem 2 is checked positively, because the reflexivity of = is an in-built assumption. Therefore Theorem 3 is checked negatively: the search for a proof of the theorem fails.

Actually, in case of the "false" Theorem 3, the reasoner makes two attempts which both fail:

```
fof(m_,hypothesis,$true).
fof(m_,hypothesis,aElement(sz0)).
fof(m_,hypothesis,aElement(szi)).
fof(m__,conjecture,( ~ (sz0 = sz0))).

[Main] (line 27 of "/home/koepke/TEST1/Temp/temp.ftl.tex")
fof(m_,hypothesis,$true).
fof(m_,hypothesis,aElement(sz0)).
fof(m_,hypothesis,aElement(szi)).
fof(m__,conjecture,( ~ (aElement(sz0) & (aElement(sz0) & (aElement(sz0) &
(aElement(sz0) & (sz0 = sz0))))))).
```

In the second round the reasoner tries, again without success, to "help" the prover by adding some information directly to the claim.

Theorem 2 must fail, because in Theorem 3 we successfully prove the opposite.

Remark 7. Naproche first parses a text and translates it into its internal logic. If this step is completed, it starts to logically check the (internal) text until the first failure. So if one wants to check more of the text, one has to exclude initial failures by inserting `\end{forthel}` and `\begin{forthel}` into the text.

Concerning Theorems 5 and 6, both have to fail, because the system has no information about the equality or inequality of 0 and 1. In certain circumstances they may agree, in others not. Note that an identifiers like 1 may suggest to human readers that this is the standard number 1 which in the standard model is $\neq 0$. But this implicit information is not available to Naproche, and indeed in axiomatizations of algebraic structures one often requires explicitely that $0 \neq 1$.

We can also form theorems with assumptions of the form "Assume that ..." where ... is a statement:

Theorem. *Assume that* $0 = 1$. *Then* $1 = 0$.

Theorem. *Assume that* $0 = 1$. *Assume that* $1 = 2$. *Then* $0 = 2$.

Exercise 4. State and prove a theorem that $\neq$ is symmetric.

7.1 Natural Language Aspects of Theorems

The Theorem command of ForTheL is invoked by the `theorem` environment of LaTeX:

```
\begin{theorem}
...
\end{theorem}
```

There are several linguistic degrees of freedom that capture common variants in the literature.

One can alternatively use the LaTeX `lemma`, `corollary`, `proposition` environments.

A theorem begins with zero or finitely many assumptions, which are of the form

```
([let us | we can] (assume | presume | suppose) [that] | let)
...
```

Alternatives are indicated by |, options by `[...]`; `...` stands for a statement.

After the assumptions the claim of the theorem consists of exactly one statement which is optionally prefixed by

```
[then | hence | thus | therefore | consequently]
```

Optional words can be viewed as filler words that make for better reading but are logically redundant. Before symbolic statements of the form $t_0 = t_1$ or $t_0 \neq t_1$ one may fill in the phrase "we have":

Theorem. *Assume that we have* $0 \neq 1$. *Then we have* $0 \neq 2$ *or we have* $1 \neq 2$.

Exercise 5. In the language of natural numbers introduced so far one can express that the constants $1, 2, ..., n$ are pairwise unequal by $\frac{n \cdot (n-1)}{2}$ assumptions of the form

Assume that $i \neq j$.

where $1 \leqslant i < j \leqslant n$. Can this also be expressed by less than $\frac{n \cdot (n-1)}{2}$ of those inequality assumptions?

ForTheL provides a shorthand for these inequalities: the statement

0,1,2,3 are pairwise nonequal.

translates to

```
[Translation] (line 29 of "/home/koepke/TEST1/Temp/temp02.ftl.tex")
(((((not 0 = 1 and not 0 = 2) and not 0 = 3) and not 1 = 2) and not 1 =
3) and not 2 = 3)
```

If we omit the word "pairwise" we get

0,1,2,3 are nonequal.

and

```
[Translation] (line 29 of "/home/koepke/TEST1/Temp/temp02.ftl.tex")
((not 0 = 1 and not 1 = 2) and not 2 = 3)
```

which means that (only) consecutive members of $0, 1, 2, 3$ are unequal.

7.2 Natural Language Connectives and Boolean Operations

We have proved that $0 = 0$ is true and that $0 \neq 0$ is false. We can take these statements as representatives for the truth values true and false. The alias mechanism can be applied to complete statements:

Let TRUE stand for $0 = 0$.
Let FALSE stand for $0 \neq 0$.

Statements φ and ψ can be combined to complex statements. In ForTheL, standard natural language connectives are available, with some natural language alternatives:

- not φ, alternatively: it is false that φ;
- φ or ψ;
- φ and ψ;
- if φ then ψ;
- φ iff ψ, alternatively: φ if and only if ψ.

Boolean operations

Applying these connectives to the truth values TRUE and FALSE corresponds to Boolean operations in a 2-element Boolean algebra:

Exercise 6. Form combinations of the statements TRUE and FALSE with the above connectives and and prove or disprove (i.e., prove the negation of) the combined statements. E.g., prove

Theorem. TRUE or FALSE.

or

Theorem. Not (TRUE or FALSE).

This exercise can be interpreted as showing that TRUE and FALSE form a Boolean algebra with the operations "and", "or", and "not".

8 Variables

Variables like $x, y, z, \ldots$ are characteristic for mathematical statements, and they are crucial for the strength of such statements: a variable stands for all possible elements of some notion so that statements with variables may include an infinity of cases.

In ForTheL, variables are *typed* as elements of notions. They can be introduced by "pretyping commands" like:

> Let x, y, z denote natural numbers.
> Let α, β, γ stand for integers.

This is a declaration, which introduces the identifiers "x", "y" and "z" as variables that range over natural numbers. If x is found in a later statement, the typing by natural numbers is automatically recalled and used for the processing of x.
After the above declarations we can formulate and prove theorems like

> **Theorem.** $x = x$.

> **Theorem.** *If* $x = y$ *then* $y = x$ *.*

> **Theorem.** *If* $x = y$ *and* $y = z$ *then* $x = y$ *.*

These theorems express that the relation = is reflexive, symmetric and transitive, i.e., that it is an equivalence relation.
The translations of these theorems show that the type declarations of the occuring variables, and how these are given to the external prover. In case of transitivity:

```
[Translation] (line 25 of "/home/koepke/TEST1/Temp/temp02.ftl.tex")
((aNaturalNumber(x) and aNaturalNumber(y)) and aNaturalNumber(z))
[Translation] (line 25 of "/home/koepke/TEST1/Temp/temp02.ftl.tex")
((x = y and y = z) implies x = z)
[Reasoner] (line 25 of "/home/koepke/TEST1/Temp/temp02.ftl.tex")
goal:  If x = y and y = z then x = z .
[Main] (line 25 of "/home/koepke/TEST1/Temp/temp02.ftl.tex")
fof(m_,hypothesis,$true).
fof(m_,hypothesis,( ! [W0] : (aNaturalNumber(W0) => (W0 = W0)))).
fof(m_,hypothesis,( ! [W0] : ( ! [W1] : ((aNaturalNumber(W0) &
aNaturalNumber(W1)) => ((W0 = W1) => (W1 = W0)))))).
fof(m__,hypothesis,((aNaturalNumber(xx) & aNaturalNumber(xy)) &
aNaturalNumber(xz))).
fof(m__,conjecture,(((xx = xy) & (xy = xz)) => (xx = xz))).
```

The translation also demonstrates that "free" variables in a theorem are interpreted as freely moving over their type. The already established theorem on the symmetry of = now appears as a hypothesis for the transitivity theorem:

```
fof(m_,hypothesis,( ! [W0] : ( ! [W1] : ((aNaturalNumber(W0) &
aNaturalNumber(W1)) => ((W0 = W1) => (W1 = W0)))))).
```

The exclamation marks "!" are the universal quantifiers of the TPTP language for automated theorem proving. A more intuitive reading is the universally quantified formula:

$$\forall w_0 \forall w_1((\mathrm{aNaturalNumber}(w_0) \wedge \mathrm{aNaturalNumber}(w_1)) \rightarrow (w_0 = w_1 \rightarrow w_1 = w_0)).$$

All variables need to be declared through pretyping (or other means of declaration). If a variable comes up without a declared type, the system will throw a parser error:

> **Theorem.** $u = u$.

leads to the error:

```
[Parser] (line 29 of "/home/koepke/TEST1/Temp/temp02.ftl.tex")
(line 29, column 2):
free undeclared variables: u
 in translation: u = u
```

Propositional logic

The truth value of a statement like $0=1$ is so far not decided, but nevertheless we consider it a properly formed mathematical statement and call it a *proposition.* Propositions can also be combined into larger propositions by the above connectives "and", "or", etc.

Let us form properties PHI and PSI:

Let PHI stand for $0=1$.

Let PSI stand for $0=2$.

Exercise 7. Study which complex statements are provable by Naproche. E.g.,

Theorem. If PHI then PHI.

Theorem. If PHI and if PHI then PSI then PSI.

The latter theorem is a basic rule of proving and is called *modus ponens*.

9 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. We shall later consider other domains as well, like sets and functions. The arithmetic functions and quantifiers will be restricted to the extension of the unary relation symbol for natural numbers. 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.

In the following we demonstrate Naproche 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;Bonn - Nordrhein-Westfalen
- 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.

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

[synonym number/-s]

Signature 8. *A natural number is a notion.*

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

Signature 9. 0 *is a natural number.*

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

Note that some natural language processing is taking place here: “nonzero” is introduced within the phrase “x is nonzero” in an adjective position. So “nonzero” can be used as an adjective which modifies “natural number”, like in:

Signature 10. 1 *is a nonzero natural number.*

10 Functions

We now introduce the usual arithmetic operations for natural numbers.

Signature 11. $m+n$ *is a natural number.*

Signature 12. $m*n$ *is a natural number.*

These commands introduce new function symbols to the language. We can find them in the first-order translation:

```
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))
```

So addition $+$ and multiplication $*$ are now function symbols of the internal logical language. In the Signature commands these are declared as symbolic patterns of the form `_ + _` and `_ * _`; these patterns are recognized since the parser is first looking for declared variables in the signature statement. These determine the “holes” in the sequence of words and symbols, and thus the newly agreed pattern. The number of holes determines the arity of the function. The types of the variables in the holes determines the argument type of the function, the “is a ...” determines the value type. Thus we get the function type of +:

$$+\colon \text{aNaturalNumber} \times \text{aNaturalNumber} \rightarrow \text{aNaturalNumber}$$

Note that 5 and 7 both have the premises
`(aNaturalNumber(m) and aNaturalNumber(n))`
for the two arguments of the operations. In the process of “ontological checking” these premises have to be proved before the operations can reasonably be applied within proofs.

11 Natural Numbers - Postulating Axioms

We now have to introduce axioms for our abstract first-order structure. Axiom are ForTheL statements written in axiom environments.

Axiom 13. $m+n=n+m$.

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

Axiom 15. $m+0=m=0+m$.

Axiom 16. $m*n=n*m$.

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

Axiom 18. *$m*1 = m = 1*m$.*

Axiom 19. *$m*0 = 0 = 0*m$.*

Axiom 20. *$m*(n+l) = (m*n)+(m*l)$ and $(n+l)*m = (n*m)+(l*m)$.*

Axiom 21. *If $l+m = l+n$ or $m+l = n+l$ then $m = n$.*

Axiom 22. *Assume that l is nonzero. If $l*m = l*n$ or $m*l = n*l$ then $m = n$.*

Axiom 23. *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.

12 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 24. *$m \leq n$ 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 25. *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 an abbreviation for another formula. The abbreviation will be expanded, possibly recursively, by the parser. The third definition defines the binary difference function $-$.

Remark 26. 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 definition of $-$ the condition for l should be satisfiable by a unique natural number. This is however not checked by $\mathbb{N}$aproche, 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 27. *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.

13 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 28. *$m \leq m$.*

Lemma 29. *If $m \leq n \leq m$ then $m = n$.*

Lemma 30. *If $m \leq n \leq l$ then $m \leq l$.*

14 Eprover in the Background

These lemmas were checked correct by $\mathbb{N}$aproche 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)).
```

15 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 31. *Contradiction.*

in various places of a text. If the lemma is validated by $\mathbb{N}$aproche 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.

16 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 32. *$m \leq n$ or $n < m$.*

Lemma 33. *Assume that $l < n$. Then $m + l < m + n$ and $l + m < n + m$.*

Lemma 34. *Assume that m is nonzero and $l < n$. Then $m * l < m * n$ and $l * m < n * m$.*

Axiom 35. *$n = 0$ or $n = 1$ or $1 < n$.*

Lemma 36. *If $m \neq 0$ then $n \leq n * m$.*

17 Induction

$\mathbb{N}$aproche has inherited an elegant treatment of induction from the SAD system. $\mathbb{N}$aproche 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 37. *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.

18 Division

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

Definition 38. *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.

19 An Interactive Proof

We shall later need a technical lemma on divisibility:

Lemma 39. *Let $l|m$ and $l|m + n$. Then $l|n$.*

On the computer I am using, $\mathbb{N}$aproche 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 $\mathbb{N}$aproche'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 40. *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 $\mathbb{N}$aproche encounters a statement immediately followed by an explicit proof then $\mathbb{N}$aproche 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 $\mathbb{N}$aproche 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.”.

$\mathbb{N}$aproche is able to prove the next lemma without an explicit proof in the text.

Lemma 41. *Let $m | n \neq 0$. Then $m \leq n$.*

20 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 42. *n is prime iff n is nontrivial and for every divisor m of n $m = 1$ or $m = n$.*

21 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 43. *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:

$$\forall v_0 (\forall v_1\, (v_1 \prec v_0 \rightarrow \phi(v_1)) \rightarrow \phi(v_0).$$

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.

22 Classes

"sets" and "classes" are built-in notions of $\mathbb{N}$aproche, 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 44. *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 $\mathbb{N}$aproche's terminology are "setsized"; both these adjective were identified earlier in the document. We extend the built-in ontology of $\mathbb{N}$aproche according to the following principles:

Axiom 45. *Every element of every class is small.*

Axiom 46. *Every set is a small class and every small class is a set.*

Classes can be naturally formed in ForTheL:

Definition 47. $\mathbb{N}$ *is the class of natural numbers.*

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

Definition 48. $\{m,...,n\} = \{$ *natural number* $i | m \leq i \leq n\}$.

23 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 $\langle \mathsf{Seq} | m | n \rangle$ 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, $\langle \mathsf{Seq} | m | n \rangle$ 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 49. $\langle \mathsf{Seq} | m | n \rangle$ *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 50. $\langle \mathsf{Seq} | m | n \rangle$ *is a set.*

The macro for the $\langle \mathsf{Seq} | m | n \rangle$- 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}
```

24 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 51. *Assume* F *is a function and* $x \in Dom(F)$. *Then* $F(x)$ *is small.*

Definition 52. *A sequence of length* n *is a function* F *such that* $Dom(F) = \langle \mathsf{Seq} | 1 | n \rangle$.

The members $F(i)$ of a sequence F are often written in an indexed notation $\langle \mathsf{val} | f | i \rangle$ where this notation is defined by a macro

⟨newcommand{\val}[2]{#1_{#2}}⟩.
The ForTheL semantics is defined by:
Let $\langle \mathsf{val} | F | i \rangle$ stand for $F(i)$.

Definition 53. *Let F be a sequence of length n. $\langle \mathsf{Set} | F | 1 | n \rangle = \{ \langle \mathsf{val} | F | i \rangle | i \in Dom(F) \}$.*

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 54. *Let F be a sequence of length n such that $\langle \mathsf{Set} | F | 1 | n \rangle \subseteq \mathbb{N}$. $\langle \mathsf{Prod} | F | 1 | n \rangle$ is a natural number.*

Axiom 55. (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 $\langle \mathsf{Prod} | F | 1 | n \rangle$ is nonzero and $F(i)$ divides $\langle \mathsf{Prod} | F | 1 | n \rangle$ 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)".

25 Finite and Infinite Sets

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

Definition 56. *S is finite iff $S = \langle \mathsf{Set} | F | 1 | n \rangle$ for some natural number n and some function F that is a sequence of length n.*

Definition 57. *S is infinite iff S is not finite.*

26 Euclid's Theorem

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

Signature 58. *$\langle \mathsf{Primes} \rangle$ is the class of prime natural numbers.*

Theorem 59. (Euclid) *$\langle \mathsf{Primes} \rangle$ is infinite.*

Proof. Assume that r is a natural number and p is a sequence of length r and $\langle \mathsf{Set} | p | 1 | r \rangle$ is a subclass of $\langle \mathsf{Primes} \rangle$.
(1) $\langle \mathsf{val} | p | i \rangle$ is a nonzero natural number for every i such that $1 \leq i \leq r$.
Consider $n = \langle \mathsf{Prod} | p | 1 | r \rangle + 1$. Take a prime divisor q of n.
Let us show that $q \neq \langle \mathsf{val} | p | i \rangle$ for all i such that $1 \leq i \leq r$.
Proof by contradiction. Assume that $q = \langle \mathsf{val} | p | i \rangle$ for some natural number i such that $1 \leq i \leq r$. q is a divisor of n and q is a divisor of $\langle \mathsf{Prod} | p | 1 | r \rangle$ (by Factorproperty, 1). Thus q divides 1. Contradiction. qed.
Hence $\langle \mathsf{Set} | p | 1 | r \rangle$ is not the class of prime natural numbers. □