An Brief Tutorial on Mathematical Formalizations in Naproche-SAD

by Peter Koepke

January 2020

Abstract

We give a brief tutorial introduction into the potential and problems of the natural language proof assistant Naproche-SAD. Naproche-SAD is a prototypical system for natural language formal mathematics, which allows some very naturalistic formalizations. On the other hand, using natural language in a strictly formal context we shall come across particular, partially unexpected problems that should even be subject of further research and development.

The tutorial examples are taken from a short ForTheL text which introduces the standard number systems $\mathbb{N} \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R}$ and some of their basic properties.

This tutorial is based on a file `Numbers01.ftl` in a folders which also contains the smaller files we are refering to.

1 Introduction

ForTheL is a Controlled natural language which approximates the ordinary natural language of mathematics, including formal phrases.

In the Naproche-SAD system, ForTheL is parsed into a structured first-order representation. ForTheL texts are checked for logical correctness using first-order ATPs.

The system aims at naturality of the input language and at naturality of proof texts. To obtain a natural level of mathematical discourse, strong ATPs are required to check a host of trivial mathematical requirements, which are usually left out in ordinary proofs. Using this approach, rather natural proofs can be conducted. Luckily one can cover a great deal of mathematics in rather simple grammars, as mathematics describes a "small world" situation of facts about mathematical object in a time-less self-contained environment. This allows to avoid temporal, modal or ambiguous language.

In the following example, the principle of complete induction on $\mathbb{N}$ is derived from the existence of infima in the real line $\mathbb{R}$ (see the file `Numbers01.ftl`).

```
Theorem. Assume E [[ NN and 1 << E and for all x << E x + 1 << E.
Then E = NN.
Proof.
Let us show that every element of NN is an element of E.
    Let n << NN.
    Assume the contrary.
    Define F = { j in NN | not j << E}.
    F is nonempty. F is bounded below.
  Take a greatest lower bound a of F.
    Let us show that a+1 is a lower bound of F.
        Let x << F. x - 1 << ZZ.
        Case x - 1 < 0. Then 0 < x < 1. Contradiction. end.
        Case x - 1 = 0. Then x = 1. not 1 << E. Contradiction. end.
        Case x - 1 > 0. Then x - 1 << NN. x - 1 << F.
            a <= x - 1.
            a + 1 <= (x - 1) + 1 = x.
            end.
    end.
    Then a+1 > a (by Next).
    Contradiction.
```

end.
qed.

Even more naturality can be achieved by putting a LaTeX-level on top of ForTheL. Actually future versions of ForTheL will be part of LaTeX and allow direct pretty-printing.

The example text above is the last theorem in a longer text on number systems, which is self-contained and contains definitions and properties used in the theorem.

In this tutorial we demonstrate how a first-order setup for a theory as above can be made, how mathematical statements can be formulated, and how documents and proofs can be structured.

1.1 Installing and using Naproche-SAD

Naproche-SAD is a command line tool which can be conveniently used in the Isabelle/jEdit proof integrated development environment (PIDE). The present version for Linux, Windows and macOS can be downloaded from `https://files.sketis.net/Isabelle_Naproche-20190611/` The installation only requires unpacking the compressed package. All required libraries like the theorem prover eprover are supplied.

The program is started be clicking the program `Isabelle_Naproche-20190611` in the `Isabelle_Naproche-20190611/` folder created by downloading and unpacking. Editing ascii-files with the file extension `ftl` will switch the editor into Naproche-SAD mode and start continuous checking of the input text with output supplied in an output window or through pop-up windows opened up by hovering the mouse over the text. Isabelle/jEdit is a powerful environment supporting proof development, which is too involved to be described here.

1.2 Disclaimer

Naproche-SAD is an experimental system under development. The general Naproche-SAD process has been devised by the SAD group in Kiev and Paris, the current implementation has been written by Andrei Paskevich (Thank you, Andrei!). The process is highly involved and coded into sophisticated Haskell. The Naproche group at Bonn is working its way into the system and its code. Large parts of the code have been rewritten without changing the overall process, and some additions have been made.

The code may and will contain a number of bugs, and we are still not able to oversee the whole of the SAD processes. Therefore the explanations in this tutorial have to be taken as provisional and potentially mistaken. We can give no guarantees.

The Naproche-SAD project primarily examines how to make formal mathematics input more natural or even very natural. The formal correctness of ForTheL texts is a longterm aim, but at the moment we cannot supply proof terms or proof certificates to guarantee correctness.

2 Setting up a first-order signature in ForTheL

2.1 ForTheL as a weakly typed language

As a CNL, ForTheL is part of natural language. Natural language can be viewed as weakly typed, i.e., using type notations without a rigid underlying type theory. ForTheL types are called "notions", which are the central organization principle of the ForTheL language.

Notions like `number` correspond to common nouns in ordinary language. The first-order translation in Naproche-SAD will turn `number` into a unary predicate symbol `aNumber(.)`. Notions can also depend on parameters. The notion `element of` depends on the term behind `element of`. Hence it is represented by a binary predicate symbol `aElementOf(.,.)`.

In ForTheL notions are introduced by `Signature` commands.

2.2 The real numbers

Open a new file `reals.ftl` and type

Signature. A real number is a notion.

Checking of the file will begin instantaneously as indicated in the Output window.

```
[Parser] (file "/home/koepke/Desktop/General_ForTheL/reals.ftl")
parsing successful
[Reasoner] (file "/home/koepke/Desktop/General_ForTheL/reals.ftl")
verification started
[Translation] (line 1 of "/home/koepke/Desktop/General_ForTheL/reals.ftl")
forall v0 ((HeadTerm :: aRealNumber(v0)) implies truth)
[Reasoner] (file "/home/koepke/Desktop/General_ForTheL/reals.ftl")
verification successful
[Main] (file "/home/koepke/Desktop/General_ForTheL/reals.ftl")
sections 1 - goals 0 - trivial 0 - proved 0 - equations 0
[Main] (file "/home/koepke/Desktop/General_ForTheL/reals.ftl")
symbols 0 - checks 0 - trivial 0 - proved 0 - unfolds 0
[Main] (file "/home/koepke/Desktop/General_ForTheL/reals.ftl")
parser 00:00.00 - reasoner 00:00.00 - simplifier 00:00.00 - prover 00:00.00/
00:00.00
[Main] (file "/home/koepke/Desktop/General_ForTheL/reals.ftl")
total 00:00.00
```

This means:

- The file is parsed into some first-order internal representation
- The "Reasoner" module initiates the line-by-line verification
- The first line is translated into first-order form

  ```
  forall v0 ((HeadTerm :: aRealNumber(v0)) implies truth)
  ```

- This shows the new unary predicate symbol `aRealNumber` and a tautological translation.
- This tautology is successfully checked.
- The output gives some statistical information about the checking process.

Several variants of the signature command are possible. The command could have a name for comment or later use, like:

Signature real_number. A real number is a notion.

The notion could have a different pattern of words, e.g., "real" instead of "real number", basically with the same effect:

Signature. A real is a notion.

And true to Hilbert "Bierseidels" etc. we could have used nonsense words like

Signature. A foobar is a notion.

2.3 Function and relation symbols

Functions on reals are also introduced by signature commands:

Signature. Let x be a real number. Let y be a real number. x + y is a real number.

To each real numbers x, y we have assigned a value $x + y$. The first order translation of the last statement reads:

```
forall v0 ((HeadTerm :: v0 = x+y) implies aRealNumber(v0))
```

We see a new infix function symbol + and a translation which registers the results of this function as being real numbers. The two assumptions: Let x be a real numbers and Let y be a real number restrict the applicability to the reals numbers, so that we have introduced a binary function symbol on the real numbers.

Several linguistic improvements are possible. The Let statements can be amalgamated to

Signature. Let x,y be real number. x + y is a real number.

One can also use the plural numbers *after* a linguistic declaration like

[synonym number/numbers]

which simply identifies the words number and numbers. This declaration has to be made before the introduction of the pattern "real number":

```
[synonym number/numbers]
Signature. A real number is a notion.
Signature. Let x,y be real numbers. x + y is a real number.
```

Another improvement is a "pretyping" of variables. After the notion of real number is available, one can agree that

Let x,y,z stand for real numbers.

Then the Let assumptions in the second signature statement can be omitted:

```
[synonym number/numbers]
Signature. A real number is a notion.
Let x,y,z denote real numbers.
Signature. x + y is a real number.
```

We can introduce the complete signature of the field of real numbers:

```
[synonym number/numbers]
Signature. A real number is a notion.
Let x,y,z denote real numbers.
Signature. x + y is a real number.
Signature. x * y is a real number.
Signature. 0 is a real number.
Signature. 1 is a real number.
```

Finally relations can be introduced:

Signature. x < y is an atom.

"atom" indicates that the symbolic phrase $x < y$ can be used as an atomic formula. This is apparent in the first order translation:

```
((HeadTerm :: x<y) implies truth)
```

Now we have the signature of an ordered field:

```
[synonym number/numbers]
Signature. A real number is a notion.
Let x,y,z denote real numbers.
Signature. x + y is a real number.
Signature. x * y is a real number.
Signature. 0 is a real number.
Signature. 1 is a real number.
Signature. x < y is an atom.
```

2.4 On the linguistics of ForTheL

Reading or parsing a natural language text is a complex task that humans are amazingly good at. The input data is arranged as a linear succession of tokens with only little internal structuring. Human readers discerns the hidden structuring using certain keywords, experience and semantical understanding. They also make use of grammatical features of language like the "numerus" of words (singular versus plural). Human readers are thus able to "disambiguate" statements which formally are ambiguous.

In its bare first-order approach, Naproche-SAD cannot use experience or understanding, but it can only look for keywords like "Signature", "is", or "notion" or key symbols like "." and ",". Also tokens of certain kinds may be pre-assigned to be variables or symbolic material. On the other hand the present ForTheL has a simplistic approach to language which supresses many details of natural language. One could describe ForTheL as an "engineered language" making use of crude simplifications and outright hacks. Probably much could be gained by expanding the linguistic quality of ForTheL and its parsing.

Technically, ForTheL is a language steering a parsing process aimed at producing intended first-order syntax. Translation results can be inspected in Isabelle by mouse hovering. It is highly recommended to check these translations for intended correctness, especially if weird effects show later in the text. ForTheL allows to write very formalistic statements close to first-order logic. The challenge of writing in ForTheL is however to write in normal mathematical language in a way that parses into the intended first-order meaning.

Let us illustrate some problems of ForTheL parsing with our little paragraph on the real numbers.

2.4.1 Notions

There is a lot of freedom in signature commands:

Signature. A real number is a notion.

Naproche-SAD would equally accept

Signature. A real number is the notion.

or

Signature. A real number is notion.

since articles "a/an/the" are often simply ignored by the system. They all create a unary predicate and translate to:

```
forall v0 ((HeadTerm :: aRealNumber(v0)) implies truth)
```

However, the article "a" in the beginning makes an essential difference:

Signature. Real number is a notion.

only creates a constant symbol `Realnumber` and translates to

```
forall v0 ((HeadTerm :: v0 = Realnumber) implies truth)
```

ForTheL is intended to correctly treat standard mathematical usage. Correctness is not guaranteed, and minor changes can be fatal.

2.4.2 Linguistic lists (coordinations)

Natural language often abreviates repetitive phrases.

Instead of

[synonym number/numbers]
Signature. A real number is a notion.

```
Let x denote a real number.
Let y denote a real number.
Signature. x + y is a real number.
```

we can contract the two "type" declarations into one:

```
[synonym number/numbers]
Signature. A real number is a notion.
Let x,y denote real numbers.
Signature. x + y is a real number.
```

It depends on the specific construct, whether such lists are allowed and lead to the desired first-order result. Here the translation is fine, when we check the typing of x and y by two propositions; the following text is checked to be correct:

```
[synonym number/numbers]
Signature. A real number is a notion.
Let x,y denote real numbers.
Proposition. x is a real number.
Proposition. y is a real number.
```

Listing with "and" instead of commas does not work:

```
[synonym number/numbers]
Signature. A real number is a notion.
Let x and y denote real numbers.
```

leads to the error:

```
illegal function alias: aRealNumber(h1)
```

Apparently, `x and y` is parsed into one complex term, which is not allowed as a pretyped *variable*.

In other places, e.g., in signature environments, even comma-separated lists of terms do not work:

```
[synonym number/numbers]
Signature. A real number is a notion.
Signature. a,b are real numbers.
```

The parser signals an `unexpected ,` .

More problematic are terms that are accepted in unexpected ways and lead to conflicts later. When a and b are coordinated by "and" instead of a comma,

```
[synonym number/numbers]
Signature. A real number is a notion.
Signature. a and b are real numbers.
```

is accepted "wrongly" as

```
forall v0 ((HeadTerm :: aAnd(v0)) implies aRealNumber(v0))
```

This indicates that `a and b` is interpreted as a pattern consisting of tokens `a` and `and` and a slot for one variable. So we have unintentionally defined a unary function that returns a real number.

2.4.3 Variables

The treatment of *variables* in formal logic is a delicate issue. Substitutions and unifications provide powerful mechanisms, but care is required to avoid variable conflicts. Here another difficulty arises, since variables may look like other any other word or token. So we have control whether tokens are read or registered as variables.

Any alphanumeric token that starts with a letter can be a variable: a,b,..., alpha, ..., foo, bar,..., a1, b3c. But these tokens may also be registered as a function, relation symbols.

It is worthwhile and irritating to experiment with texts like

```
[synonym number/numbers]
Signature. A real number is a notion.
Let gamma denote a real number.
Signature. alpha + beta is a real number.
Lemma. gamma + gamma is a real number.
```

versus

```
[synonym number/numbers]
Signature. A real number is a notion.
Let alpha, beta, gamma denote real numbers.
Signature. alpha + beta is a real number.
Lemma. gamma + gamma is a real number.
```

versus

```
[synonym number/numbers]
Signature. A real number is a notion.
Signature. alpha + alpha is a real number.
```

versus

```
[synonym number/numbers]
Signature. A real number is a notion.
Let alpha, beta, gamma denote real numbers.
Signature. alpha + alpha is a real number..
```

One needs to check for error messages, translations or effects in lemmas about the type of terms to understand the parsing.

2.4.4 Understanding the "understanding"

We may imagine Naproche-SAD to be somebody who is confronted with a (very) foreign language and has no general vocabulary available, except a small number of keywords like "signature", "is", and the like. How is that person to read a text like

```
[synonym number/numbers]
Signature. A real number is a notion.
Signature. alpha + alpha is a real number..
```

At the third line, the person will see that the signature command wants to introduce something new to the vocabulary. So the person will read until the keyword "is" is encountered and declare everything read so far as the new constant "alpha + alpha". If, however, "alpha" was declared to be a variable beforehand:

```
[synonym number/numbers]
Signature. A real number is a notion.
Let alpha denote a real number.
Signature. alpha + alpha is a real number..
```

then "alpha" is recognized (twice) in the last command. Now another problem arises since we do not want a variable to occur twice in the left hand side of a definition. The following correctly introduces the binary infix operation "+":

```
[synonym number/numbers]
```

```
Signature. A real number is a notion.
Let alpha, beta denote real numbers.
Signature. alpha + beta is a real number.
```

Observe that we may also use the same token in various functionalities (see `Numbers02.ftl`):

```
[synonym number/numbers]
Signature. A real number is a notion.
Let alpha, beta, gamma denote real numbers.
Signature Infix. alpha + beta is a real number.
Signature Prefix. + beta is a real number.
Signature Constant. + is a real number.
Lemma. + + is a real number.
Lemma. + (+ +) is a real number.
Lemma. (+ +) + (+ +) is a real number.
```

introduces "+" as binary infix, unary prefix and (0-ary) constant symbol. The lemmas show that all roles can co-exist in one statement if the parsing is unambiguous.

Ambiguous parsing leads to an error message with a list of alternative readings:

```
Lemma. + + + + + is a real number.
```

fails with

```
(line 10, column 8):
ambiguity error[aRealNumber(++(+++)).
,aRealNumber((+++)++).
,aRealNumber(++(+(++))).
,aRealNumber((++)+(++)).
,aRealNumber((+(++))++).
,aRealNumber(+(+(+(++)))).
]
```

Generally speaking a user of Naproche-SAD should only expect very limited linguistic capacities of the system and check first-order translations.

3 Statements

First-order symbols introduced can be combined to terms and atomic formulas. In our setup for the field of real numbers, we can form complex terms like (x + y) + z and x + (y + z) and put them into an atomic equality (x + y) + z = x + (y + z) as usual. Note that statements can only be used in certain environments like Axiom, Proposition or Proof. There are no "free" statements at arbitrary positions in a proper ForTheL text.

```
[synonym number/numbers]
Signature Real. A real number is a notion.
Let x,y,z denote real numbers.
Signature A1. x + y is a real number.
Signature M1. x * y is a real number.
Signature. x < y is an atom.
Axiom A2. x + y = y + x.
Axiom A3. (x + y) + z = x + (y + z).
```

Here we see two axioms A2 and A3, consisting of statements. After parsing, the corresponding first-order translations are added to a list of premises that can later be used in the logical checking of the text.

There are various ways to form complex statements out of atomic ones. Statements can be negated, composed by conjunctions and disjunctions, and quantified. ForTheL provides many natural language constructs.

Our examples so far were excerpts from a file `Numbers01.ftl` . We now take another excerpt on $\mathbb{R}$ as a linear order and bounds of subsets of $\mathbb{R}$. We comment this text in lines starting with a "#" (see `Numbers03.ftl`):

```
Let A,B stand for sets.
# Sets and elementhood are already predefined in Naproche-SAD.
Let x << A denote x is an element of A.
# One can use new notation like "x << A" for a statement like "x is an element of A".
Let x is in A denote x is an element of A.

Signature. The empty set is the set that has no elements.
# Signature commands can be extended by a "that" phrase
# by axiomatic assumptions on the newly introduced piece
# of language. Here something like "A has no elements" is stipulated.
# "has no" is a quantification phrase, whose interpretation can be clearly viewed
# in the first-order translation.
Let Emptyset denote the empty set.

Definition. A is nonempty iff A has an element.
# This is a definition which introduces the pattern "nonempty" and a defining equivalence.

Definition. A subset of B is a set A such that every element of A
is an element of B.
# The subset notion is a dependent notion, depending on the given set A.
# "every element of A" is a quantifier, typed with the notion of being an element of A.
Let A [[ B stand for A is a subset of B.
Let B ]] A stand for A is a subset of B.

Definition. A proper subset of B is a subset A of B such that there
is an element of B that is not in A.

Proposition. A [[ A.

Proposition. If A [[ B and B [[ A then A = B.
# Both propositions are checked automatically.

Definition. A \cup B = {x | x<<A \/ x<<B}.
# ForTheL also has inbuilt symbolic logical operations like the disjunction "\/".
```

The formal content of this text becomes more apparent in the output of the automatic checking:

```
[Parser] (file "/home/koepke/Desktop/General_ForTheL/reals.ftl")
parsing successful
[Reasoner] (file "/home/koepke/Desktop/General_ForTheL/reals.ftl")
verification started
[Translation] (line 6 of "/home/koepke/Desktop/General_ForTheL/reals.ftl")
forall v0 ((HeadTerm :: v0 = theEmptySet) implies (aSet(v0) and not exists v1
aElementOf(v1,v0)))
[Translation] (line 9 of "/home/koepke/Desktop/General_ForTheL/reals.ftl")
aSet(A)
[Translation] (line 9 of "/home/koepke/Desktop/General_ForTheL/reals.ftl")
((HeadTerm :: isNonempty(A)) iff exists v0 aElementOf(v0,A))
[Translation] (line 11 of "/home/koepke/Desktop/General_ForTheL/reals.ftl")
aSet(B)
```

```
[Translation] (line 11 of "/home/koepke/Desktop/General_ForTheL/reals.ftl")
forall v0 ((HeadTerm :: aSubsetOf(v0,B)) iff (aSet(v0) and forall v1
(aElementOf(v1,v0) implies aElementOf(v1,B))))
[Translation] (line 16 of "/home/koepke/Desktop/General_ForTheL/reals.ftl")
aSet(B)
[Translation] (line 16 of "/home/koepke/Desktop/General_ForTheL/reals.ftl")
forall v0 ((HeadTerm :: aProperSubsetOf(v0,B)) iff (aSubsetOf(v0,B) and exists
v1 (aElementOf(v1,B) and not aElementOf(v1,v0))))
[Translation] (line 19 of "/home/koepke/Desktop/General_ForTheL/reals.ftl")
aSet(A)
[Translation] (line 19 of "/home/koepke/Desktop/General_ForTheL/reals.ftl")
aSubsetOf(A,A)
[Reasoner] (line 19 of "/home/koepke/Desktop/General_ForTheL/reals.ftl")
goal:  A [[ A.
[Translation] (line 21 of "/home/koepke/Desktop/General_ForTheL/reals.ftl")
(aSet(A) and aSet(B))
[Translation] (line 21 of "/home/koepke/Desktop/General_ForTheL/reals.ftl")
((aSubsetOf(A,B) and aSubsetOf(B,A)) implies A = B)
[Reasoner] (line 21 of "/home/koepke/Desktop/General_ForTheL/reals.ftl")
goal:  If A [[ B and B [[ A then A = B.
[Translation] (line 23 of "/home/koepke/Desktop/General_ForTheL/reals.ftl")
(aSet(A) and aSet(B))
[Translation] (line 23 of "/home/koepke/Desktop/General_ForTheL/reals.ftl")
forall v0 ((HeadTerm :: v0 = A\cup B) iff (aSet(v0) and forall v1
(aElementOf(v1,v0) iff (Replacement :: (aElementOf(v1,A) or aElementOf(v1,
B))))))
[Reasoner] (file "/home/koepke/Desktop/General_ForTheL/reals.ftl")
verification successful
[Main] (file "/home/koepke/Desktop/General_ForTheL/reals.ftl")
sections 20 - goals 2 - trivial 0 - proved 2 - equations 0
[Main] (file "/home/koepke/Desktop/General_ForTheL/reals.ftl")
symbols 32 - checks 25 - trivial 25 - proved 0 - unfolds 3
[Main] (file "/home/koepke/Desktop/General_ForTheL/reals.ftl")
parser 00:00.04 - reasoner 00:00.00 - simplifier 00:00.00 - prover 00:00.20/
00:00.02
[Main] (file "/home/koepke/Desktop/General_ForTheL/reals.ftl")
total 00:00.25
```

This log shows the (correct) first-order translations and logs the successive proving tasks to be performed by the reasoner and the ATP.

3.1 Appendix: On the ForTheL statement grammar

The global statement grammar is described in Andrei Paskevich's *ForTheL Handbook* as follows

constStatement → [the] thesis
| [the] contrary
| [a | an] contradiction

Statements are composed with prepositions and conjunctions as follows:

statement → *headStatement* | *chainStatement*

headStatement → for *quantifiedNotion* { and *quantifiedNotion* } *statement*
| if *statement* then *statement*
| it is wrong that *statement*

chainStatement → *andChain* [and *headStatement*]
| *orChain* [or *headStatement*]
| (*andChain* | *orChain*) iff *statement*

andChain → *primaryStatement* { and *primaryStatement* }

orChain → *primaryStatement* { or *primaryStatement* }

The production rules of this formal grammar correspond to nested parsers in the Naproche-SAD code:

```
statement = headed <|> chained

headed = quStatem <|> ifThenStatem <|> wrongStatem
  where
    quStatem = liftM2 ($) quChain statement
    ifThenStatem = liftM2 Imp
      (wdToken "if" >> statement)
      (wdToken "then" >> statement)
    wrongStatem =
      mapM_ wdToken ["it", "is", "wrong", "that"] >> fmap Not statement

chained = andOr <|> neitherNor >>= chainEnd
  where
    andOr = atomic >>= \f -> opt f (andChain f <|> orChain f)
    andChain f =
      fmap (foldl And f) $ and >> atomic `sepBy` and
    orChain f = fmap (foldl Or f) $ or >> atomic `sepBy` or
    and = wdToken "and"
    or = wdToken "or"
    neitherNor = do
      wdToken "neither"; f <- atomic
      d "nor"
      fs <- atomic `sepBy` wdToken "nor"
      return $ foldl1 And $ map Not (f:fs)

chainEnd f = optLL1 f $ and_st <|> or_st <|> iff_st <|> where_st
  where
    and_st = fmap (And f) $ wdToken "and" >> headed
    or_st = fmap (Or f) $ wdToken "or" >> headed
    iff_st = fmap (Iff f) $ iff >> statement
    where_st = do
      wdTokenOf ["when", "where"]; y <- statement
      return $ foldr zAll (Imp y f) (declNames [] y)

atomic = (simple </> (wehve >> thesis))
  where
    wehve = optLL1 () $ wdToken "we" >> wdToken "have"

thesis = art >> (thes <|> contrary <|> contradiction)
  where
    thes = wdToken "thesis" >> return zThesis
    contrary = wdToken "contrary" >> return (Not zThesis)
    contradiction = wdToken "contradiction" >> return Bot
```

```
simple = ...
```

4 Another example

Sometimes mathematical results are expressed just in words. The main theorem in T. Hales' *A proof of the Kepler conjecture* reads:

> THEOREM 1.1 (The Kepler conjecture). No packing of congruent balls in Euclidean three space has density greater than that of the face-centered cubic packing.

In ForTheL one can axiomatically introduce the notions involved in the formulation of THEOREM 1.1 (see `Hales.ftl`)

```
[synonym number/-s]
Signature. A real number is a notion.
Let x,y stand for real numbers.

Signature. x is greater than y is an atom.

Signature. A packing of congruent balls
in Euclidean three space is a notion.

Signature. The face centered cubic packing is a packing
of congruent balls in Euclidean three space.

Let P denote a packing of congruent balls in
Euclidean three space.

Signature. The density of P is a real number.

Theorem The_Kepler_conjecture. No packing of congruent
balls in Euclidean three space has density greater than
the density of the face centered cubic packing.
```

This file is parsed successfully in Naproche-SAD but surely the verification fails:

```
[Parser] (file "/home/koepke/TEST/Hales.ftl")
parsing successful
[Reasoner] (file "/home/koepke/TEST/Hales.ftl")
verification started
[Translation] (line 2 of "/home/koepke/TEST/Hales.ftl")
forall v0 ((HeadTerm :: aRealNumber(v0)) implies truth)
[Translation] (line 5 of "/home/koepke/TEST/Hales.ftl")
(aRealNumber(x) and aRealNumber(y))
[Translation] (line 5 of "/home/koepke/TEST/Hales.ftl")
((HeadTerm :: isGreaterThan(x,y)) implies truth)
[Translation] (line 7 of "/home/koepke/TEST/Hales.ftl")
forall v0 ((HeadTerm :: aPackingOfCongruentBallsInEuclideanThreeSpace(v0))
implies truth)
[Translation] (line 10 of "/home/koepke/TEST/Hales.ftl")
forall v0 ((HeadTerm :: v0 = theFaceCenteredCubicPacking) implies
aPackingOfCongruentBallsInEuclideanThreeSpace(v0))
[Translation] (line 16 of "/home/koepke/TEST/Hales.ftl")
```

```
aPackingOfCongruentBallsInEuclideanThreeSpace(P)
[Translation] (line 16 of "/home/koepke/TEST/Hales.ftl")
forall v0 ((HeadTerm :: v0 = theDensityOf(P)) implies aRealNumber(v0))
[Translation] (line 18 of "/home/koepke/TEST/Hales.ftl")
forall v0 (aPackingOfCongruentBallsInEuclideanThreeSpace(v0) implies not
isGreaterThan(theDensityOf(v0),theDensityOf(theFaceCenteredCubicPacking)))
[Reasoner] (line 18 of "/home/koepke/TEST/Hales.ftl")
goal:  No packing of congruent balls in Euclidean three space has density
greater than the density of the face centered cubic packing.
[Reasoner] (line 18 of "/home/koepke/TEST/Hales.ftl")
goal failed
[Reasoner] (file "/home/koepke/TEST/Hales.ftl")
verification failed
[Main] (file "/home/koepke/TEST/Hales.ftl")
sections 7 - goals 1 - failed 1 - trivial 0 - proved 0 - equations 0
[Main] (file "/home/koepke/TEST/Hales.ftl")
symbols 5 - checks 5 - trivial 5 - proved 0 - unfolds 3
[Main] (file "/home/koepke/TEST/Hales.ftl")
parser 00:00.01 - reasoner 00:00.00 - simplifier 00:00.00 - prover 00:00.04/
00:00.00
[Main] (file "/home/koepke/TEST/Hales.ftl")
total 00:00.06
```

Note that the wording of the Theorem in ForTheL is very close to the original. Some differences are:

- whereas the original uses the anaphora "that" to avoid repetition of the word "density", we have to repeat it;
- "face centered" is written without the hyphen "-" since this would be mis-interpreted as a minus symbol.

This is a self-contained ForTheL text. Notions are introduced axiomatically as basically meaningless pattern. ForTheL supports the combination of such patterns into natural language sentences. Internally, statements are translated into first-order logic. The statement of the Theorem becomes:

```
[Translation] (line 18 of "/home/koepke/TEST/Hales.ftl")
forall v0 (aPackingOfCongruentBallsInEuclideanThreeSpace(v0) implies not
isGreaterThan(theDensityOf(v0),theDensityOf(theFaceCenteredCubicPacking)))
```

where `aPackingOfCongruentBallsInEuclideanThreeSpace` is a unary relation symbol, `isGreaterThan` is a binary relation symbol, `theDensityOf` is a unary function symbol, and `theFaceCenteredCubicPacking` is a constant symbol.

The translation is correct in first-order logic where we have the equivalence

$$\neg\exists v_0(P(v_0) \wedge G(v_0)) \Leftrightarrow \forall v_0(P(v_0) \rightarrow \neg G(v_0))$$

ForTheL uses many natural language techniques and also some clever tricks and conventions to simulate the reading process of an expert mathematician. ForTheL is a *pattern based language*, where new patterns can be introduced through language extensions (`Signature`) and definitions. Reading consists in identifying patterns and sending them to the corresponding first-order symbols, preserving logical structure.

The intuitive notion of $\geq$ on reals is introduced to the vocabulary of the language by:

```
Signature. x is greater than y is an atom.
```

From this the parser extracts or "learns" a relation pattern like `?,is,greater,than,?` and assigns to it a new binary relation symbol `isGreaterThan` that is put into the vocabulary. Subsequently the parser is regularly looking for this pattern and, if successful, will generate a formula of the form `isGreaterThan(_,_)` with entries which themselves may be results of such pattern recognitions.

Thinking of one's own reading process or that of children indicates, that we are facing an involved trial-and-error-process where the reader is reading a text word by word with perhaps several possible interpretations in mind and with some state of "knowledge" accumulated from earlier parts of the text. We can only expect this tree-like structure of alternative interpretations to collapse to a unique reading when the sentence is completed by a dot ".".

Parsing in the Naproche-SAD system can be considered as *Natural Language Processing* (NLP) adjusted to mathematical contexts. So we shall have patterns corresponding to nouns (notions), adjectives and verbs, and also symbolic mathematical patterns like `?,+,?` or `\sqrt{?}` (for $\sqrt{?}$). These patterns will be arranged in grammatical sentence constructs like in nounphrase / verbphrase sentences, or more involved sentences. Note however that nouns, adjectives, etc. are characterized here only by their grammatical function and not by some (English) dictionary. So we can modify Hilberts alleged quote:

> "Man muß jederzeit an Stelle von 'Punkte, Geraden, Ebenen', 'Tische, Stühle, Bierseidel' sagen können"

to the extremely formalistic:

> "Man muß jederzeit an Stelle von 'Punkte, Geraden, Ebenen', 'abcde, $\prec^{\sim}$, a packing of congruent balls in Euclidean three space' sagen können".

These are just arbitrary strings like points in a set, and we can just say whether two of these strings are equal or distinct. Although words like congruent, ball, Euclidean space evoke some mathematical meaning, this doesn't figure at all in the ForTheL text.

5 Ontological checking

A particular feature of Naproche-SAD is the *ontological* checking of a statement before checking its main mathematical content. Before proving an inequality like

$$\delta(x, r, \Lambda) \leq \pi / \sqrt{18} + C / r$$

on has to check the claim is well-formed within the locally established environment:

- if $\delta(v_0, v_1, v_2)$ is a density of a packing of balls within a sphere of radius v_1 around a midpoint v_0 then the well-formedness or ontological correctness of the left-hand side requires that $x \in \mathbb{R}^3$, $r \in \mathbb{R}$, $r > 0$, and Λ is such a packing.
- The preconditions or presuppositions for the application of δ can be accumulated from the introduction or definition of δ and from preconditions for further constituents of δ .
- The right-hand side requires among others that divisors are $\neq 0$. So the proof task $r \neq 0$ is given to the reasoner/prover within the local context. Going from left to right, one has already shown that $r > 0$ which by properties of $>$ implies $r \neq 0$.
- Once subterms are well-formed, one can go to the compound terms or statements. The relation $\leq$ requires that both sides are real numbers, which spawns other proof tasks.
- If all these subtasks are successful, the statement declared ontologically correct. Proving ontological correctness can amount to a fair number of such subtasks, which ideally should all be rather easy compared to the main statement.
- The main task might require a difficult proof, which may be interactive with intelligent human input.

Note that if the ontological check fails, this does not say that the global formula is false. Instead it says that the formula was formed in a way that at its place in the proof it is ill-formed.

Let us experiment with quotients of real numbers in the following excerpt from our file on numbers (see `Numbers04.ftl`):

```
[synonym number/-s]
Signature Real. A real number is a notion.
```

```
Let x,y,z denote real numbers.

Signature A1. x + y is a real number.
Signature M1. x * y is a real number.
Axiom A2. x + y = y + x.
Axiom A3. (x + y) + z = x + (y + z).
Signature A4. 0 is a real number such that
for every real number x x + 0 = x.

Axiom M2. x * y = y * x.
Axiom M3. (x * y) * z = x * (y * z).
Signature M4. 1 is a real number such that 1 !=0 and
for every real number x 1 * x = x.
Signature M5. Assume x != 0. 1/x is a real number
such that x * (1/x) = 1.

Axiom D. x * (y + z) = (x * y) + (x * z).

Proposition. If x != 0 and x * y = x then y = 1.
Proposition. If x != 0 and x * y = 1 then y = 1/x.
Proposition. If x != 0 then 1/(1/x) = x.
```

This text checks automatically in Naproche-SAD but one can break it by not supplying enough information to make 1/x well-defined. The checking fails with coloring the offensive statement violet. Hovering reveals: `unrecognized: 1/x` . One can get the same effect by commenting out some of the axioms until $x \neq 0$ is no longer provable.

As mentioned earlier, functions and relations require their arguments to be of certain types. If there are two types around like sets and numbers, statements can fail the type-checking (see `Numbers05.ftl`):

```
Let A,B stand for sets.

[synonym number/-s]
Signature Real. A real number is a notion.

Let x,y,z denote real numbers.

Signature A1. x + y is a real number.
Signature M1. x * y is a real number.
Axiom A2. x + y = y + x.
Axiom A3. (x + y) + z = x + (y + z).
Signature A4. 0 is a real number such that
for every real number x x + 0 = x.

Axiom M2. x * y = y * x.
Axiom M3. (x * y) * z = x * (y * z).
Signature M4. 1 is a real number such that 1 !=0 and
for every real number x 1 * x = x.

Lemma. A + B = 0.
```

Then applying addition to sets A, B results in `unrecognized: A+B`. Similarly

```
Lemma. x is an element of y.
```

leads to `unrecognized: aElementOf(x,y)`. In contrast,

```
Lemma. x is an element of A.
```

is ontologically correct since sets can have anything as elements, but the property cannot be proven since there is not further information on x and A.

6 Proofs

The main proof mode of Naproche-SAD is interactive: proving sophisticated and diverse mathematical results with input from a human user. We describe a few common proof techniques that are also available in Naproche-SAD and are employed in `Number01.ftl` .

6.1 Chains of equalities

Proposition 1_14_a. If x + y = x + z then y = z.
Proof. Assume x + y = x + z. Then
y = (-x+x) + y = -x + (x+y) = -x + (x+z) = (-x+x) + z = z.
qed.

Parsing transforms this into a conjunctive chain of single equalities:

```
((((y = ((-x)+x)+y and ((-x)+x)+y = (-x)+(x+y)) and (-x)+(x+y) = (-x)+(x+z))
and (-x)+(x+z) = ((-x)+x)+z) and ((-x)+x)+z = z)
```

Observe that the proof requires restating the premise of the implication: "Assume $x+y=x+z$." In human proofs, this assumption is often surpressed and only made implicitly. Currently, Naproche-SAD has no way to figure this out.

When proving a theorem (= proposition = lemma), Naproche-SAD carries along a current *thesis* to be proved. This thesis directs proof search. When the proof is completed, the thesis become "true". Hovering displays the current thesis. The assumption discussed above reduces the thesis `thesis: (x+y = x+z implies y = z)` to `new thesis: y = z` .

6.2 Proof by cases

From `Numbers01.ftl`:

Proposition. Assume 0 < x < y. Then 1/y < 1/x.
Proof by case analysis.
Case 1/x < 1/y. Then
1 = x * (1/x) = (1/x) * x < (1/x) * y = y * (1/x) < y * (1/y) = 1.
Contradiction. 1/y < 1/x.
end.
Case 1/x = 1/y. Then
1 = x * (1/x) < y * (1/y) = 1.
Contradiction. 1/y < 1/x.
end.
Case 1/y < 1/x. obvious.

1/x < 1/y or 1/x = 1/y or 1/y < 1/x.
qed.

The problem is split into three cases. Each "case" construct proves that the case assumption implies the thesis. Using *ex falso quodlibet* Contradiction implies the (any) thesis. Provided that the case assumptions exhaust all possibilities the proposition is proved. The system needs to be "reminded" of the trichotomie 1/x < 1/y or 1/x = 1/y or 1/y < 1/x. to complete the proof.

6.3 Proof by contradiction

From `Numbers01.ftl`:

```
Theorem. Assume A [[ NN and 1 << A and for all n << A n + 1 << A.
Then A = NN.
Proof.
Let us show that every element of NN is an element of A.
      Let n << NN.
      Assume the contrary.
      Define F = { j in NN | not j << A}.
      F is nonempty. F is bounded below.
  Take a greatest lower bound a of F.
      Let us show that a+1 is a lower bound of F.
            Let x << F. x - 1 << ZZ.
            Case x - 1 < 0. Then 0 < x < 1. Contradiction. end.
            Case x - 1 = 0. Then x = 1. not 1 << A. Contradiction. end.
            Case x - 1 > 0. Then x - 1 << NN. not x - 1 << A.
      Proof. Assume the contrary. x = (x-1) + 1. qed.
                  a <= x - 1.
                  a + 1 <= (x - 1) + 1 = x.
                  end.
      end.
      Then a+1 > a (by Next).
      Contradiction.
end.
qed.
```

The token contrary denotes the negation of the current thesis. Assuming the contrary inserts the negation of the current thesis into the local assumptions and make `contradiction` the thesis to be proved.

6.4 General remarks

Proving in ForTheL is a subtle, even fragile procedure depending on the invocation and success of eprover. "Non-linear" properties of the ATP influence the properties of Naproche-SAD. E.g., adding an assumption may confuse the ATP (because of a larger search space?) and may make a goal fail that was successfully provable with fewer assumptions.