A Short Introduction To Naproche v0.1

Daniel Kuehlwein

July 8, 2008

1 Introduction

This document gives a quick overview of Naproche as of June 30th 2008. We define the input language, and how Naproche deals with the different constructs. Then, we explain the output and feedback system of Naproche. After the theory, we take a closer look at an example.

Please keep in mind that this is a work in progress, and that this document only explains what has already been implemented. For more information visit:

http://www.math.uni-bonn.de/people/naproche/

1.1 General Outline Of Naproche

Naproche - NAtural language PRoof CHEcker - aims at narrowing the gap between natural language mathematics and automated theorem provers. Using computer linguistic and mathematical means, we want to translate natural language mathematical texts into a format which is readable for automated theorem provers (ATP). Combining Texmacs, a Latex WYSIWYG editor, Naproche and an ATP gives a program which is able to check natural language mathematical texts for correctness.

2 Input, Implementation And Output

This sections first introduces two of the formats used in current Naproche. We then define the part of the english language which is currently allowed as input. After that, we explain how the input is handled by Naproche, and, finally, we talk about the output Naproche produces, and how to understand and use it.

2.1 Formats

We explain two of the formats used in Naproche in more detail. Please note that there are more formats involved in Naproche, namely PRSs and DOBSODs. These are, however, beyond the scope of this article.

2.1.1 The PROLOG Format

While Naproche aims at being able to process natural language mathematics, we are not quite there yet. At the moment, every input has to be written directly in the sourcecode in a format we call, as the programming laguage we use,

PROLOG. Sentences are prolog lists, and mathematical symbols are written in their unicode representation. We give a few examples of natural language mathematic expressions and their corresponding PROLOG input. These examples do not provide a complete translation of natural language into PROLOG, but hopefully give you the general idea.

Natural language	Assume, that $\neg\exists y$ such that $y \in \emptyset$.
PROLOG	[assume, that,math("\u00ac \u2203 y (y \u2208 \u2205)")]
Natural language	Define $Trans(x)$ if, and only if $\forall u, v((u \in v) \wedge (v \in x)) \rightarrow (u \in x)$.
PROLOG	[define, math("Trans(x)"),iff,math("\u2200 u \u2200 v (((u \u2208 v) \u2227 (v \u2208 x)) \u2192 (u \u2208 x))")]
Natural language	Then $\exists x, x \in \emptyset$.
PROLOG	[then, math("\u2203 x x \u2208 \u2205")]

2.1.2 The TPTP Format

Originally, each automated theorem prover had its own input format. In the past few years, with the rise of CASC, and, with it, the Thousand Problems for Theorem Provers, the TPTP format emerged. The TPTP format is a way of writing first order logic which is understood by almost every first order ATP. In Naproche, we translate natural language to ATP queries. We write the queries in TPTP, as this allowes us to try different ATP systems. Therefore, a basic understanding of the TPTP format is helpful. Again, we will only scratch the surface.If you want more information about TPTP, take a look at www.tptp.org

A first order formula in TPTP has the following Syntax:

$$fof(name, role, formula).$$

name is the name of the formula, *role* states whether the formula is an axiom or a conjecture and $formula$ is the formula we are talking about. In section 3 we will talk about the name and the role in more detail.

Our examples from before look as follows in TPTP

Natural language	Assume, that $\neg\exists y$ such that $y \in \emptyset$.
PROLOG	[assume, that,math("\u00ac \u2203 y (y \u2208 \u2205)")]
TPTP	fof(1, axiom, (?[Vy]:(in(Vy,vemptyset)))).
Natural language	Define $Trans(x)$ if, and only if $\forall u, v((u \in v) \wedge (v \in x)) \rightarrow (u \in x)$.
PROLOG	[define, math("Trans(x)"),iff,math("\u2200 u \u2200 v (((u \u2208 v) \u2227 (v \u2208 x)) \u2192 (u \u2208 x))")]
TPTP	fof(1, axiom, ![Vx]:((trans(Vx))<=> (![Vu]:(![Vv]: (((in(Vu,Vv)) & (in(Vv,Vx)))=> (in(Vu,Vx))))))).
Natural language	Then $\exists x, x \in \emptyset$.
PROLOG	[then, math("\u2203 x x \u2208 \u2205")]
TPTP	fof(1, conjecture, ?[Vx]:(in(Vx,vemptyset))).

2.2 The Input

Automatic understanding of natural language is the topic of computer linguistics. If computers could understand natural language, translations from one language into another would not be a problem anymore. If you ever used any automatic translation program you probably noticed that we are not quite there yet. One way around this problem is to simply restrict the allowed input. Naproche also facilitates this approach.

In the following, *statement* stand for a mathematical formula. We now define the allowed input constructs:

2.2.1 Statements

A statement sentence consists of a, possibly empty, statement_prefix and the actual statement. So we have the structure:

$$[statement\_prefix, statement]$$

Here, statement_prefix can be any of the following: [then],[hence],[recall, that], [but], [in, particular],[observe, that],[together, we, have],[so].

e.g. [then, math("\u2203 x x \u2208 \u2205")] is a valid statement.

2.2.2 Negation

A negation is quite similar to a normal statement:

$$[not, statement]$$

e.g. [not, math("x=x")] is a valid negation.

2.2.3 Definitions

A definition has two possible structures:

$$[define, left\_statement, iff, right\_statement]$$

and

$$[define, left\_statement, if, and, only, if, right\_statement]$$

e.g. [define, math("Trans(x)"),iff,math("\u2200 u \u2200 v (((u \u2208 v) \u2227 (v \u2208 x)) \u2192 (u \u2208 x))")] is a valid definition.

2.2.4 Assumptions

Assumptions are a common mathematical construction. However, there is one problem. In natural language texts, they often end impicitly. In order to make it easier for us, we introduce a trigger to mark the end of an assumption. Therefore, assumptions usually come in pairs. (There is one exception, which involved a [qed] sentence. See structural markers for more details).

To open an assumption we use the structure:

$$[assumption\_trigger, statement]$$

where assumption_trigger is any of the following: [assume, that], [consider], [let], [assume, for, a, contradiction, that].

To close an assumption we use:

$$[thus, statement]$$

The assumption is valid for every sentence between the [assumption_trigger] and the [thus] sentence.

e.g. [assume, that, math("\u00ac Trans(\u2205)")],[then, math("\u2203 x x \u2208 \u2205")],[contradiction],[thus, math("Trans(\u2205)")].

2.2.5 Implication

Implications are triggered by *implies*:

$$[statement, implies, statement]$$

e.g. [math("x=y"), implies, math("y=x")].

2.2.6 For all

Forall statements using natural language quantification have the structure:

$$[for, all, statement\_list, (,), statement]$$

Here statement_list is a list of statement. We expect a list here to make quantification over several variable at once possible.

e.g. [for, all, math("x"), (,) math("x=x")].

2.2.7 Contradiction

"Proof by Contradiction" is a another common construct in mathematical proofs. A contradiction is indicated by:

$$[contradiction]$$

2.2.8 Structural markers

Mathematical texts also use markers like *Theorem* or *Lemma* to structure them. In Naproche we allow the following constructions:

$$[[theorem], goal, [proof], body, [qed]]$$

$$[[lemma], goal, [proof], body, [qed]]$$

Here *goal* and *body* are list of statements. Note that *[qed]* closes any assumptions which are still open in the end of the proof.

e.g. [[theorem],[math("x=x")],[proof],[math("x=x")],[qed]] is a valid theorem.

2.3 The Implementation

In section 2.2, we specified the input of Naproche. In this section, we explain what is happening internally, when parsing a sentence.

When processing a text, Naproche keeps a list of *Premises*. At any given moment, *Premises* contains all statements which we consider true at that moment. Naproche takes care of keeping the *Premises* updated, and of checking statements using an ATP. Note that at the start of Naproche, *Premises* is an empty list. In the following, we describe the Naproche algorithm for each sentence type.

2.3.1 Statements

Statements are the most basic sentences. Upon parsing a statement sentence, Naproche tries to prove the statement from the current *Premises*.

To be more precise, Naproche first translates the *Premises* and the statement into TPTP and writes the result in *Output/InputId*, where Id is a unique string, corresponding to the sentence. It then calls *SystemsOnTPTP*, a program written by *Geoff Sutcliffe*, to start the ATP. The result of the ATP is saved in *Output/OutputId*. Naproche then creates a new line in *final_output* containing the Id, the formula to be proved in TPTP and the result. (For more details see section 2.4 and 3). Finally the statement is added to the list of *Premises*.

e.g. Assume that *Premises* is empty. The sentence

```
[hence, math("x=x)"]
```

first creates a new input file, say *Output/input0*, which contains

```
fof(1, conjecture, =(vx,vx)).
```

then it runs the ATP, and stores the result in an output file, *Output/output0*:

```
Output/input0 Theorem 0.0 0.0  0 1 12
Output/input0 Refutation 0.0 0.0
```

as the ATP says *Theorem*, we add the following line to *final_output*:

```
0     [ fof(1, conjecture, =(vx,vx))]     Theorem
```

2.3.2 Negation

Recall, that a negation has the form $[not, statement]$. Naproche tries to prove the negated statement from the *Premises*.. Then it adds the negated statement to *Premises*.

e.g.

```
[not, math("x=y")]
```

After trying to prove $\neg x = y$, Naproche adds $\neg(x = y)$ to the *Premises*.

2.3.3 Definitions

Definitions are treated as equivalences. A definition sentence, $[define, left\_statement, iff, right\_statement]$, adds $left\_statement \leftrightarrow right\_statement$ to the *Premises*.

e.g.

```
[define, math("Trans(x)"),iff,math("\u2200 u \u2200 v
  ( ((u \u2208 v) \u2227 (v \u2208 x)) \u2192 (u \u2208 x))")],
```

Adds $Trans(x) \leftrightarrow \forall u, v((u \in v) \wedge (v \in x) \rightarrow (u \in x)$ to the *Premises*.

2.3.4 Assumptions

Upon opening an assumption, e.g. $[let, statement]$, statement is added to the *Premises*. When we encounter $[thus]$ or $[qed]$, the *Premises*, which were valid before opening the assumption, are restored. Then, for each statement between the opening and the closing of the assumption, we add $\forall$[free variables in the assumption], $assumption\_statement \rightarrow statement$ to the *Premises*. If $[thus]$ was used to close the assumption, Naproche then tries to prove the statement following $[thus]$ from the updated *Premises*.

There is one special case, namely, when a contradiction was proved while the assumption was open. In that case, every statement between the opening and the closing is discarded. We restore the old *Premises* and add $\neg assumption\_statement$.

e.g. Assume that *Premises* is empty. Consider the following discourse.

```
[[let, math("x=y")],
 [hence, math("y=x"],
 [thus, math("\u2200 x \u2200 y ( (x = y) \u2192 (y = x))")]]
```

Parsing the sentence `[let, math("x=y")]` adds $x = y$ to *Premises*. `[hence, math("y=x"]` makes Naproche try and prove $y = x$ from the *Premises*. `[thus, math("\u2200 x \u2200 y ( (x = y) \u2192 (y = x))")]` does two things. First, it closes the assumption and resets the *Premises* to the empty list. Then, Naproche calculates the free variables of the assumption statement, x and y, and adds $\forall x, y(x = y \rightarrow y = x)$ to the *Premises*. Second, it tries to prove the statement following `[thus]` from the *Premises*. This, of course, succeeds as they are the same.

2.3.5 Implication

When encountering an implication $[statementA, implies, statementB]$, first, $statementA$ is added to the *Premises*. Then, Naproche tries to prove $statementB$ from the updated *Premises*. Afterwards, we restore the old *Premises*, and add $statementA \rightarrow statementB$.

e.g.

```
[math("x=y"), implies, math("y=x")]
```

Naproche first tries to prove $y = x$ from $x = y$, and then adds $x = y \rightarrow y = x$ to *Premises*.

2.3.6 For all

When parsing a natural language quantification
$[for, all, statement\_list, (,), statement]$, Naproche tries to prove
$\forall statement\_list, statement$ from the *Premises*. Then
$\forall statement\_list, statement$ is added.

e.g.

```
[for, all, math("x"), (,), math("x=x")]
```

We first try to prove $\forall x, x = x$ and then add $\forall x, x = x$ to *Premises*.

2.3.7 Contradiction

The sentence $[contradiction]$ makes Naproche try, and prove $\$false$. It then adds $\$false$ to the *Premises*. For the use of $[contradiction]$ see 2.3.4.

2.3.8 Structural markers

Both $[[theorem], goal, [proof], body, [qed]]$ and
$[lemma], goal, [proof], body, [qed]]$ are treated the same. Therefore, we only consider the $[theorem]$ case. When Naproche parses $[theorem]$, it creates a temporary variable and saves all the statements in *goal* in it. It then parses *body*. Afterwards, we try to prove *goal* from all the *Premises*, which were availabe before $[theorem]$, and the statements from *body*. Finally, we discard *body* and add *goal* to the *Premises*.

e.g. Assume the *Premises* is empty.

```
[[theorem],
 [math("y=y"],
 [proof],
 [math("x=x")],
 [qed]].
```

We first try to prove the *body*: $x = x$, and afterwards add $x = x$ to the *Premises*. Then, the *goal*,$y = y$, has to be proved from *Premises*. After parsing these sentences, *Premises* contains only $y = y$.

2.4 The Output

Naproche provides you with several means to keep track of what it actually does when parsing a mathematical text. After running Naproche, you will find a file named *final_output* in your Naproche folder, as well as lots of new files, named *inputID* and *outputID*, in your *Output* folder. Here, *Id* is either a number, or a string followed by a number.

2.4.1 final_output

final_output is basically a table with three columns. Each row corresponds to one query to the ATP. The first column notes the Id of the sentence which is processed. The second column has the statement which Naproche tries to prove, in TPTP format. And the third column holds the result of the query.

e.g. The input, which is a proof for $Ord(\emptyset)$,

```
[assume, that,math("\u00ac \u2203 y ( y \u2208 \u2205 )")],
[assume, that, for,all,math("x"), (,)
,math("\u00ac ( x \u2208 x )")],
[define, math("Trans(x)"),iff,math("\u2200 u \u2200 v (
((u \u2208 v) \u2227 (v \u2208 x)) \u2192 (u \u2208 x))")],
[define,math("Ord(x)"),iff,math("Trans(x) \u2227 (\u2200 y (
(y \u2208 x) \u2192 Trans(y) ) )")],

[theorem],
[math("Ord(\u2205)")],
[proof],

[consider, math("u \u2208 v"), and ,math("v \u2208 \u2205")],
[assume, that, math("\u00ac Trans(\u2205)")],
[then, math("\u2203 x x \u2208 \u2205")],
[contradiction],
[thus, math("Trans(\u2205)")],

[assume, that, math("\u00ac Trans(v)")],
[then, math("\u2203 x x \u2208 \u2205")],
[contradiction],
[thus, math("Trans(v)")],

[thus, math("Ord(\u2205)")],

[qed]
```

creates the following *final_result*:

```
 9     [fof(1, conjecture, ?[Vx]:(in(Vx,vemptyset)))]     Theorem
10     [fof(1, conjecture, $false)]     Theorem
11     [fof(1, conjecture, trans(vemptyset))]     Theorem
13     [fof(1, conjecture, ?[Vx]:(in(Vx,vemptyset)))]     Theorem
14     [fof(1, conjecture, $false)]     Theorem
15     [fof(1, conjecture, trans(vv))]     Theorem
theorem_4     [fof(1, conjecture, ord(vemptyset))]     Theorem
```

Here, each call was successful, as can be seen by the word *Theorem* in the third colum. If you compare the calls with the input and the description in 2.3, you can get a good idea of how Naproche works.

2.4.2 The Output folder

final_output already gives you a good overview of what is happening. The *Output* folders contains even more information. For each ATP query, two files get created in *Output*, *InputId* and *OutputId*. Here, *Id* is the unique identification number of the input sentence.

e.g. When you take a look at the example in the last section, you will find that the first row in *final_result* is

```
 9     [fof(1, conjecture, ?[Vx]:(in(Vx,vemptyset)))]     Theorem
```

The *Output* folders contains the corresponding input and output files: *input9* and *output9*. The input file contains:

```
fof(1, axiom, ~(?[Vy]:(in(Vy,vemptyset)))).
fof(2, axiom, ![Vx]:(~(in(Vx,Vx)))).
fof(3, axiom, ![Vx]:((trans(Vx))<=>(![Vu]:(![Vv]:
(((in(Vu,Vv))&(in(Vv,Vx)))=>(in(Vu,Vx)))))).
fof(4, axiom, ![Vx]:((ord(Vx))<=>((trans(Vx))&(![Vy]:
((in(Vy,Vx))=>(trans(Vy)))))).
fof(5, axiom, in(vv,vemptyset)).
fof(6, axiom, in(vu,vv)).
fof(7, axiom, ~(trans(vemptyset))).
fof(1, conjecture, ?[Vx]:(in(Vx,vemptyset))).
```

Our query is: Try to proof conjecture 1 using only the axioms 1-7. Have a look at the input, if you are not sure where these axioms stem from.

The output file contains:

```
Output/input9 Theorem 0.0 0.0  0 1 12
Output/input9 Refutation 0.0 0.0
```

Which tells us, that this is a theorem, meaning that the ATP could prove conjecture 1 from the axiom 1-7.

3 An Example

This section takes a closer look at an example. First, we explain how to use Naproche in its current form. Then, we give the example in normal english, followed by the example in PROLOG format. We then present *final_output*. Finally, we go through the example, one sentence at a time, looking at the valid *Premises* at any given point, and compare the theory with the actual results.

3.1 Running Naproche

First make sure that you have TPTP World and SWI Prolog installed. Have a look at the README file in the Naproche folder for installation instructions. After that, open the terminal, go to your main Naproche directory, and start prolog using the 'pl' command. After that, consult load.pl, as well as, runner.pl via *[load].* and *[runner].* You should get the following output:

```
 Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.6.55)
Copyright (c) 1990-2008 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- [load].
%  library(pldoc) compiled into pldoc 0.14 sec, 407,876 bytes
```

```
%  library(plunit) compiled into plunit 0.02 sec, 78,548 bytes
%  library(sgml) compiled into sgml 0.00 sec, 19,840 bytes
%  library(porter_stem) compiled into porter_stem 0.00 sec,
 2,288 bytes
%   library(occurs) compiled into occurs 0.01 sec,
6,844 bytes
%  library(apply_macros) compiled into apply_macros 0.01 sec,
16,268 bytes
%  naproche(gulp4swi) compiled into gulp4 0.01 sec,
19,752 bytes
%  naproche(utils) compiled into utils 0.00 sec, 9,248 bytes
%  naproche(production_utils) compiled 0.01 sec, 5,820 bytes
%  naproche(development_utils) compiled into
development_utlis 0.00 sec, 4,760 bytes
%  naproche(xml) compiled into xml 0.01 sec, 8,660 bytes
%  naproche(texmacs) compiled into texmacs 0.00 sec, 6,664 bytes
%  naproche(prs) compiled into prs 0.02 sec, 50,044 bytes
%   naproche(translation_tptp) compiled into
translation_tptp 0.01 sec, 7,880 bytes
%  naproche(prs_export) compiled into prs_export
0.01 sec, 23,500 bytes
%   premises compiled into premises 0.02 sec, 22,296 bytes
%   fof_check compiled into fof_check 0.01 sec, 16,608 bytes
%  naproche(checker) compiled into check_prs 0.04 sec,
54,884 bytes
%  naproche(expr_grammar) compiled into expr_grammar
0.01 sec, 16,668 bytes
%  naproche(grammar) compiled 0.02 sec, 23,536 bytes
%  naproche(math_lexicon) compiled into math_lexicon
0.00 sec, 10,120 bytes
%  naproche(lexicon) compiled into lexicon 0.00 sec, 5,096 bytes
%  test/production_utils.plt compiled 0.00 sec, 10,332 bytes
%  test/development_utils.plt compiled 0.01 sec, 1,656 bytes
%  test/xml.plt compiled 0.00 sec, 6,364 bytes
%  test/prs/prs.plt compiled 0.01 sec, 30,416 bytes
%  test/prs/prs_export.plt compiled 0.00 sec, 11,112 bytes
%  test/logic/translation_tptp.plt compiled 0.01 sec,
11,996 bytes
%  test/logic/checker.plt compiled 0.03 sec, 47,908 bytes
%  test/logic/premises.plt compiled 0.04 sec, 58,140 bytes
%  test/logic/fof_check.plt compiled 0.01 sec, 11,768 bytes
%  test/expr_grammar.plt compiled 0.02 sec, 37,636 bytes
% Started Prolog Documentation server at port 8000
% You may access the server at http://localhost:8000/
% load compiled 0.78 sec, 1,778,512 bytes
true.

?- [runner].
Warning: /home/rekzah/Programming/Naproche/naproche/
  runner.pl:350:
```

```
        Redefined static procedure logger/1
% runner compiled 0.03 sec, 27,844 bytes
true.

?-
```

If you want, you can type *run_tests.* to check whether everything is working, but for now we assume that that is the case. Next, we start our example by typing *try_check(ord).*. You should get the following output:

```
?- try_check(ord).
true.

?-
```

Note, that between typing the command and getting the result (true) up to 5 seconds will pass. Most of this time is needed for the actual checking of the proof. We hope that we can reduce it by using other checkers, and more efficient coding. You can use the *try_nocheck(ord).* command to see how long it takes without checking, and the *try(ord,X).* command if you want to just build the PRS.

3.2 The Ordinals Example In Normal English

The following text originates from set theory, a mathematical discipline. Set theory concerns itself with the very basics of mathematics. Together with logic, it can be seen at the foundation of nowadays mathematics. The text proves that there is a difference between classes and sets. In particular, that the class of all ordinals is not a set. The only assumptions we use are the existence of an emtpy set, and the foundation axiom. To make the natural language version easier to compare with the PROLOG version, each sentence is written in a separate line.

Assume, that $\neg\exists y$ such that $y \in \emptyset$.
Assume, that for all x holds $\neg x \in x$.
Define $Trans(x)$ if, and only if $\forall u, v((u \in v) \wedge (v \in x)) \rightarrow (u \in x)$.
Define $Ord(x)$ if, and only if $Trans(x) \wedge (\forall y(y \in x) \rightarrow Trans(y)$.

Theorem.
$Ord(\emptyset)$.

Proof.
Consider $u \in v$ and $v \in \emptyset$.
Assume that $\neg Trans(\emptyset)$.
Then $\exists x, x \in \emptyset$.
Contradiction.
Thus $Trans(\emptyset)$.
Assume that $\neg Trans(v)$.
Then $\exists x, x \in \emptyset$.
Contradiction.
Thus $Trans(v)$.
Thus $Ord(\emptyset)$.

Qed.

Theorem.
For all $x, y\ x \in y \wedge Ord(y)$ implies $Ord(x)$.

Proof.
Consider $x \in y$ and $Ord(y)$.
Then $\forall x((x \in y) \rightarrow Trans(x))$.
Hence $Trans(x)$.
Assume that $u \in x$.
Then $u \in y$.
Hence $Trans(u)$.
Thus $Ord(x)$.
Qed.

Theorem.
For all $x, \neq (\forall u(u \in x) \rightarrow Ord(u))$.

Proof.
Assume for a contradiction that there is an x such that $\forall u((u \in x) \rightarrow Ord(u))$.

Lemma.
$Ord(x)$
Proof.
Let $u \in v$ and $v \in x$.
Then $Ord(v)$.
Hence $Ord(u)$.
So, $u \in x$.
Thus $Trans(x)$.
Let $v \in x$.
Then $Ord(v)$.
So $Trans(v)$.
Thus $Ord(x)$.
Qed.

Then $x \in x$.
Contradiction.
Qed.

3.3 The Ordinals Example In Naproche

As our plugin for Texmacs is not working at the moment, we have to hardcode the example in our source files. In the Naproche folder, you find a file called *runner.pl*. This file contains tests and examples. The translation of the example into our current input format, PROLOG, can be found between the lines 120 and 197. We also present it here:

```
 p(ord,Sentence) :-
Sentence = [
```

```
[assume, that,math("\u00ac \u2203 y ( y \u2208 \u2205 )")],
[assume, that, for,all,math("x"), (,)
  ,math("\u00ac ( x \u2208 x )")],
[define, math("Trans(x)"),iff,math("\u2200 u \u2200 v (
  ((u \u2208 v) \u2227 (v \u2208 x)) \u2192 (u \u2208 x))")],
[define,math("Ord(x)"),iff,math("Trans(x) \u2227 (\u2200 y (
(y \u2208 x) \u2192 Trans(y) ) )")],

[theorem],
[math("Ord(\u2205)")],
[proof],
[consider, math("u \u2208 v"), and ,math("v \u2208 \u2205")],

[assume, that, math("\u00ac Trans(\u2205)")],
[then, math("\u2203 x x \u2208 \u2205")],
[contradiction],
[thus, math("Trans(\u2205)")],

[assume, that, math("\u00ac Trans(v)")],
[then, math("\u2203 x x \u2208 \u2205")],
[contradiction],
[thus, math("Trans(v)")],

[thus, math("Ord(\u2205)")],

[qed],

[theorem],
[for ,all , math("x"), (,), for, all, math("y"), (,),
math("( x \u2208 y \u2227 Ord(y) )"), implies, math("Ord(x)")],
[proof],

[consider, math("x \u2208 y"), and, math("Ord(y)")],
[then, math("\u2200 x ( (x \u2208 y) \u2192 Trans(x) )")],
[hence, math("Trans(x)")],
[assume, that, math("u \u2208 x")],
[hence, math("u \u2208 y")],
[then, math("Trans(u)")],
[thus, math("Ord(x)")],

[qed],

[theorem],
[for, all, math("x"), (,) , math("\u00ac (\u2200 u (
  (u \u2208 x) \u2194 Ord(u)) )")],
[proof],
[assume, for, a, contradiction, that, there, is, an, math("x"),
   such, that, math("\u2200 u (
   (u \u2208 x) \u2194 Ord(u) )")],
```

```
[lemma],
[math("Ord(x)")],
[proof],
[let, math("u \u2208 v"), and, math("v \u2208 x")],
[then, math("Ord(v)")],
[hence, math("Ord(u)")],
[so, math("u \u2208 x")],
[thus, math("Trans(x)")],
[let, math("v \u2208 x")],
[then, math("Ord(v)")],
[so, math("Trans(v)")],
[thus, math("Ord(x)")],
[qed],

[then, math("x \u2208 x")],
[contradiction],

[qed]

].
```

The programming language we use is prolog. Our input is a list of lists, where every 'internal' list represents a sentence in the normal proof. We use Unicode to encode mathematical symbols such as $\forall, \exists$ etc. That aside, you will find that our input is almost the same as the natural english proof, which we presented in the last section.

3.4 *final_output* after running Naproche

After running Naproche on the ordinals example *final_output* should look as follows:

```
9     [fof(1, conjecture, ?[Vx]:(in(Vx,vemptyset)))]     Theorem
10     [fof(1, conjecture, $false)]     Theorem
11     [fof(1, conjecture, trans(vemptyset))]     Theorem
13     [fof(1, conjecture, ?[Vx]:(in(Vx,vemptyset)))]     Theorem
14     [fof(1, conjecture, $false)]     Theorem
15     [fof(1, conjecture, trans(vv))]     Theorem
theorem_4     [fof(1, conjecture, ord(vemptyset))]     Theorem
22     [fof(1, conjecture, ![Vx]:
  ((in(Vx,vy))=>(trans(Vx))))]     Theorem
23     [fof(1, conjecture, trans(vx))]     Theorem
25     [fof(1, conjecture, in(vu,vy))]     Theorem
26     [fof(1, conjecture, trans(vu))]     Theorem
27     [fof(1, conjecture, ord(vx))]     Theorem
theorem_18     [fof(1, conjecture, ![Vx]:(![Vy]:(![Vy,Vx]:
  (((in(Vx,Vy))&(ord(Vy)))=>(ord(Vx))))))]     Theorem
37     [fof(1, conjecture, ord(vv))]     Theorem
38     [fof(1, conjecture, ord(vu))]     Theorem
39     [fof(1, conjecture, in(vu,vx))]     Theorem
```

```
40     [fof(1, conjecture, trans(vx))]     Theorem
42     [fof(1, conjecture, ord(vv))]     Theorem
43     [fof(1, conjecture, trans(vv))]     Theorem
lemma_33     [fof(1, conjecture, ord(vx))]     Theorem
46     [fof(1, conjecture, in(vx,vx))]     Theorem
47     [fof(1, conjecture, $false)]     Theorem
theorem_29     [fof(1, conjecture,
![Vx]:(~(![Vu]:((in(Vu,Vx))<=>(ord(Vu))))))]     Theorem
```

Each line corresponds to one call of the ATP, in our case OTTER. In this case, we could prove everything, as can be seen by the word *Theorem* at the end of each line.

3.5 Understanding Naproche

Now, we look at one sentence at a time. We explain what is going on, define the *Premises*, and compare the theory with the output we get in *final_output* and the *Output* folder.

3.5.1 Preliminaries

The first four lines of our example contain basic assumptions and definitions, under which we are working.

```
[assume, that,math("\u00ac \u2203 y ( y \u2208 \u2205 )")],
[assume, that, for,all,math("x"), (,)
  ,math("\u00ac ( x \u2208 x )")],
[define, math("Trans(x)"),iff,math("\u2200 u \u2200 v (
  ((u \u2208 v) \u2227 (v \u2208 x)) \u2192 (u \u2208 x))")],
[define,math("Ord(x)"),iff,math("Trans(x) \u2227 (\u2200 y (
(y \u2208 x) \u2192 Trans(y) ) )")],
```

Each opening of an assumptions adds a premise, and also each definition adds a premise. (See 2.3 for more details.) As there are only assumptions and definitions, there is nothing to prove. We list the active *Premises* after processing these four lines and proceed:

active Premises
$\neg\exists y, y \in \emptyset$
$\forall x, \neg(x \in x)$
$Trans(x) \leftrightarrow \forall u, v((u \in v) \wedge (v \in x)) \rightarrow (u \in x)$
$Ord(x) \leftrightarrow Trans(x) \wedge (\forall y(y \in x) \rightarrow Trans(y)$

3.5.2 The First Theorem: $Ord(\emptyset)$

The first thing we prove is $Ord(\emptyset)$:

```
[theorem],
[math("Ord(\u2205)")],
[proof],
[consider, math("u \u2208 v"), and ,math("v \u2208 \u2205")],
```

```
[assume, that, math("\u00ac Trans(\u2205)")],
[then, math("\u2203 x x \u2208 \u2205")],
[contradiction],
[thus, math("Trans(\u2205)")],

[assume, that, math("\u00ac Trans(v)")],
[then, math("\u2203 x x \u2208 \u2205")],
[contradiction],
[thus, math("Trans(v)")],

[thus, math("Ord(\u2205)")],

[qed],
```

Upon parsing the sentence `[theorem]` , Naproche looks for the keywords `[proof]` and `[qed]` . All sentences between `[theorem]` and `[proof]` become the *goal*, all sentences between `[proof]` and `[qed]` are the *body*. Naproche tries to prove the *goal* from the *body*.

In this case, *goal* only contains one statement: $Ord(\emptyset)$. This is parsed and stored for later. Naproche proceeds by checking the *body*.

The first two statements of the *body* are two more assumptions. After parsing those, *Premises* contains:

Premises
$\neg\exists y, y \in \emptyset$
$\forall x, \neg(x \in x)$
$Trans(x) \leftrightarrow \forall u, v((u \in v) \wedge (v \in x)) \rightarrow (u \in x)$
$Ord(x) \leftrightarrow Trans(x) \wedge (\forall y(y \in x) \rightarrow Trans(y)$
$(u \in v) \wedge (v \in \emptyset)$
$\neg Trans(\emptyset)$

The next sentence is `[then, math("\u2203 x x \u2208 \u2205")]` . This is the first statement. Time for some real work. Recall, that a statement means that we have to prove it from the current *Premises*. If you have a look at *final_output*, you find that the first line of that file corresponds to our sentence:

Natural language
Then $\exists x, x \in \emptyset$
PROLOG
`[then, math("\u2203 x x \u2208 \u2205")]`
final_output
9 [fof(1, conjecture, ?[Vx]:(in(Vx,vemptyset)))] Theorem

Furthermore, the file *Output/input9* gives you a way of checking which *Premises* Naproche considers active at that moment. Again, the input corresponds to the *Premises* listed above.

```
fof(1, axiom, ~(?[Vy]:(in(Vy,vemptyset)))).
fof(2, axiom, ![Vx]:(~(in(Vx,Vx)))).
```

```
fof(3, axiom, ![Vx]:((trans(Vx))<=>(![Vu]:(![Vv]:
  (((in(Vu,Vv))&(in(Vv,Vx)))=>(in(Vu,Vx)))))).
fof(4, axiom, ![Vx]:((ord(Vx))<=>((trans(Vx))&(![Vy]:
  ((in(Vy,Vx))=>(trans(Vy))))))).
fof(5, axiom, in(vv,vemptyset)).
fof(6, axiom, in(vu,vv)).
fof(7, axiom, ~(trans(vemptyset))).
fof(1, conjecture, ?[Vx]:(in(Vx,vemptyset))).
```

For the sake of completeness, we also list *Output/output9*:

```
Output/input9 Theorem 0.0 0.1  0 1 12
Output/input9 Refutation 0.0 0.1
```

This gives the same result as *final_output*: This statement can be proven from the *Premises*, and therefore is a *Theorem*.

After checking this sentence, the statement is added to the *Premises*. Therefore, after parsing `[then, math("\u2203 x x \u2208 \u2205")]`, our *Premises* are:

Premises
$\neg\exists y, y \in \emptyset$
$\forall x, \neg(x \in x)$
$Trans(x) \leftrightarrow \forall u, v((u \in v) \wedge (v \in x)) \rightarrow (u \in x)$
$Ord(x) \leftrightarrow Trans(x) \wedge (\forall y(y \in x) \rightarrow Trans(y)$
$(u \in v) \wedge (v \in \emptyset)$
$\neg Trans(\emptyset)$
$\exists x, x \in \emptyset$

Next comes `[contradiction]` . First, Naproche tries to prove *$false* from the *Premises*. As $\neg\exists y, y \in \emptyset$ and $\exists x, x \in \emptyset$ are both available *Premises*, it should be no surprise that this check succeeds. *final_output*s second line corresponds with this check:

```
 10     [fof(1, conjecture, $false)]     Theorem
```

Output/input10 contains our *Premises* with the conjecture $false.

```
fof(1, axiom, ~(?[Vy]:(in(Vy,vemptyset)))).
fof(2, axiom, ![Vx]:(~(in(Vx,Vx)))).
fof(3, axiom, ![Vx]:((trans(Vx))<=>(![Vu]:(![Vv]:
  (((in(Vu,Vv))&(in(Vv,Vx)))=>(in(Vu,Vx)))))).
fof(4, axiom, ![Vx]:((ord(Vx))<=>((trans(Vx))&(![Vy]:
  ((in(Vy,Vx))=>(trans(Vy))))))).
fof(5, axiom, in(vv,vemptyset)).
fof(6, axiom, in(vu,vv)).
fof(7, axiom, ~(trans(vemptyset))).
fof(8, axiom, ?[Vx]:(in(Vx,vemptyset))).
fof(1, conjecture, $false).
```

and *Output/output10* says *Theorem*, as expected:

```
Output/input10 Theorem 0.0 0.0  0 1 12
Output/input10 Refutation 0.0 0.0
```

Finally, *$false* is added to the *Premises*. *Premises* now contains:

Premises
$\neg\exists y, y \in \emptyset$
$\forall x, \neg(x \in x)$
$Trans(x) \leftrightarrow \forall u, v((u \in v) \wedge (v \in x)) \rightarrow (u \in x)$
$Ord(x) \leftrightarrow Trans(x) \wedge (\forall y(y \in x) \rightarrow Trans(y)$
$(u \in v) \wedge (v \in \emptyset)$
$\neg Trans(\emptyset)$
$\exists x, x \in \emptyset$
$\$false$

The next sentence is `[thus, math("Trans(\u2205)")]` . Recall, that *thus* closes an assumption. As we have a contradiction, all statements made in the scope of the assumptions are discarded, and the negated assumptions statement is added to the *Premises*. After this, the active *Premises* are:

Premises
$\neg\exists y, y \in \emptyset$
$\forall x, \neg(x \in x)$
$Trans(x) \leftrightarrow \forall u, v((u \in v) \wedge (v \in x)) \rightarrow (u \in x)$
$Ord(x) \leftrightarrow Trans(x) \wedge (\forall y(y \in x) \rightarrow Trans(y)$
$(u \in v) \wedge (v \in \emptyset)$
$\neg\neg Trans(\emptyset)$

From these updated *Premises*, we now try to prove the statement `[math("Trans(\u2205)")]` . Because $\neg\neg Trans(\emptyset)$ is an active premise, this also succeeds.

The third line in *final_output* corresponds to this query:

```
11      [fof(1, conjecture, trans(vemptyset))]      Theorem
```

Output/input11 contains the exact query, the axioms corresponding to our *Premises*, the conjecture to the statement.

```
fof(1, axiom, ~(?[Vy]:(in(Vy,vemptyset)))).
fof(2, axiom, ![Vx]:(~(in(Vx,Vx)))).
fof(3, axiom, ![Vx]:((trans(Vx))<=>(![Vu]:(![Vv]:
  (((in(Vu,Vv))&(in(Vv,Vx)))=>(in(Vu,Vx))))))).
fof(4, axiom, ![Vx]:((ord(Vx))<=>((trans(Vx))&(![Vy]:
  ((in(Vy,Vx))=>(trans(Vy))))))).
fof(5, axiom, in(vv,vemptyset)).
fof(6, axiom, in(vu,vv)).
fof(7, axiom, ~(~(trans(vemptyset)))).
fof(1, conjecture, trans(vemptyset)).
```

Finally, the statement is added to *Premises*:

Premises
$\neg\exists y, y \in \emptyset$
$\forall x, \neg(x \in x)$
$Trans(x) \leftrightarrow \forall u, v((u \in v) \wedge (v \in x)) \rightarrow (u \in x)$
$Ord(x) \leftrightarrow Trans(x) \wedge (\forall y(y \in x) \rightarrow Trans(y)$
$(u \in v) \wedge (v \in \emptyset)$
$\neg\neg Trans(\emptyset)$
$Trans(\emptyset)$

The next four lines of the *body* of the theorem are dealt with equivalently. We present the active *Premises* after parsing them:

Premises
$\neg\exists y, y \in \emptyset$
$\forall x, \neg(x \in x)$
$Trans(x) \leftrightarrow \forall u, v((u \in v) \wedge (v \in x)) \rightarrow (u \in x)$
$Ord(x) \leftrightarrow Trans(x) \wedge (\forall y(y \in x) \rightarrow Trans(y)$
$(u \in v) \wedge (v \in \emptyset)$
$\neg\neg Trans(\emptyset)$
$Trans(\emptyset)$
$\neg\neg Trans(v)$
$Trans(v)$

We now close another assumption: `[thus, math("Ord(\u2205)")]` . This time, however, we don't have a contradiction. Therefore, Naproche does the following. First, it determins the free variables of the assumption statement: u and v. For each statement in the scope of the assumption, namely $\neg\neg Trans(\emptyset)$, $Trans(\emptyset)$, $\neg\neg Trans(v)$ and $Trans(v)$, Naproche adds $\forall u, v$ *assumption* $\rightarrow$ *statement* to *Premises*. The active *Premises* are then:

Premises
$\neg\exists y, y \in \emptyset$
$\forall x, \neg(x \in x)$
$Trans(x) \leftrightarrow \forall u, v((u \in v) \wedge (v \in x)) \rightarrow (u \in x)$
$Ord(x) \leftrightarrow Trans(x) \wedge (\forall y(y \in x) \rightarrow Trans(y)$
$\forall u, v((u \in v) \wedge (v \in \emptyset)) \rightarrow \neg\neg Trans(\emptyset)$
$\forall u, v((u \in v) \wedge (v \in \emptyset)) \rightarrow Trans(\emptyset)$
$\forall u, v((u \in v) \wedge (v \in \emptyset)) \rightarrow \neg\neg Trans(v)$
$\forall u, v((u \in v) \wedge (v \in \emptyset)) \rightarrow Trans(v)$

Normally, Naproche would now try and prove the statement `[thus, math("Ord(\u2205)")]` . But in this case, a special feature comes into play. If the last statement of the proof is the same as the *goal* of theorem, we skip proving it, and, instead, try to prove the theorem. This is reflected in *final_output* and the *Output* folder.

3.5.3 The Rest Of The Text

We hope that, after giving you a detailed step by step approach in the last section, you can easily understand the remainder of the input text. Remember to use *final_output* and *Output* to check the valid *Premises* and the ATP queries.

If you do have any questions, or found a bug, please contact us. You find the contact details, as well as news and updates to Naproche at

$$http://www.math.uni-bonn.de/people/naproche/$$