Diproche - Automatic Proof Checking for Didactical Applications

Problem statement

Mathematics needs practice. Therefore, there are exercises.

Problem: Feedback comes long after exercises have been finished (typically about a week later) and can in particular not enter the process of solving and presenting a solution.

Approach: Partially automatize the feedback.

Problem statement

Mathematics needs practice. Therefore, there are exercises.

Problem: Feedback comes long after exercises have been finished (typically about a week later) and can in particular not enter the process of solving and presenting a solution.

Approach: Partially automatize the feedback.

Problem statement

Mathematics needs practice. Therefore, there are exercises.

Problem: Feedback comes long after exercises have been finished (typically about a week later) and can in particular not enter the process of solving and presenting a solution.

Approach: Partially automatize the feedback.

Problem statement

Mathematics needs practice. Therefore, there are exercises.

Problem: Feedback comes long after exercises have been finished (typically about a week later) and can in particular not enter the process of solving and presenting a solution.

Approach: Partially automatize the feedback.

From the QED Manifesto:

"The third motivation for the QED project is education. (...) The development of mathematical ability is notoriously dependent upon "doing" rather than upon "being told" or "remembering". The QED system will provide, via such techniques as interactive proof checking algorithms (...), an opportunity for the one-on-one presenting, checking, and debugging of mathematical technique, which it is so expensive to provide by the method of one trained mathematician in dialogue with one student. QED can provide an engaging and non-threatening framework for the carrying out of proofs by students, (...)."

From the QED Manifesto:

"The third motivation for the QED project is education. (...) The development of mathematical ability is notoriously dependent upon "doing" rather than upon "being told" or "remembering". The QED system will provide, via such techniques as interactive proof checking algorithms (...), an opportunity for the one-on-one presenting, checking, and debugging of mathematical technique, which it is so expensive to provide by the method of one trained mathematician in dialogue with one student. QED can provide an engaging and non-threatening framework for the carrying out of proofs by students, (...)."

From the QED Manifesto:

"The third motivation for the QED project is education. (...) The development of mathematical ability is notoriously dependent upon "doing" rather than upon "being told" or "remembering". The QED system will provide, via such techniques as interactive proof checking algorithms (...), an opportunity for the one-on-one presenting, checking, and debugging of mathematical technique, which it is so expensive to provide by the method of one trained mathematician in dialogue with one student. QED can provide an engaging and non-threatening framework for the carrying out of proofs by students, (...)."

From the QED Manifesto:

"The third motivation for the QED project is education. (...) The development of mathematical ability is notoriously dependent upon "doing" rather than upon "being told" or "remembering". The QED system will provide, via such techniques as interactive proof checking algorithms (...), an opportunity for the one-on-one presenting, checking, and debugging of mathematical technique, which it is so expensive to provide by the method of one trained mathematician in dialogue with one student. QED can provide an engaging and non-threatening framework for the carrying out of proofs by students, (...)."

From the QED Manifesto:

"The third motivation for the QED project is education. (...) The development of mathematical ability is notoriously dependent upon "doing" rather than upon "being told" or "remembering". The QED system will provide, via such techniques as interactive proof checking algorithms (...), an opportunity for the one-on-one presenting, checking, and debugging of mathematical technique, which it is so expensive to provide by the method of one trained mathematician in dialogue with one student. QED can provide an engaging and non-threatening framework for the carrying out of proofs by students, (...)."

Some sample proofs

A mathematical argument can proceed by a step-by-step deduction, but it doesn't have to:

Example 1: On Monday, 8.00, Gandalf starts in the Shire and travels to Rivendell, where he has a date with Galadriel at 19.00. On Tuesday morning, he starts in Rivendell at 8.00 and travels back to the Shire along the same route. Show that there is a time of the day at which he was in the same spot on Monday and Tuesday.

Solution: Imagine two Gandalfs travelling on the same day. Clearly, they must meet.

Some sample proofs

A mathematical argument can proceed by a step-by-step deduction, but it doesn't have to:

Example 1: On Monday, 8.00, Gandalf starts in the Shire and travels to Rivendell, where he has a date with Galadriel at 19.00. On Tuesday morning, he starts in Rivendell at 8.00 and travels back to the Shire along the same route. Show that there is a time of the day at which he was in the same spot on Monday and Tuesday.

Solution: Imagine two Gandalfs travelling on the same day. Clearly, they must meet.

Some sample proofs

A mathematical argument can proceed by a step-by-step deduction, but it doesn't have to:

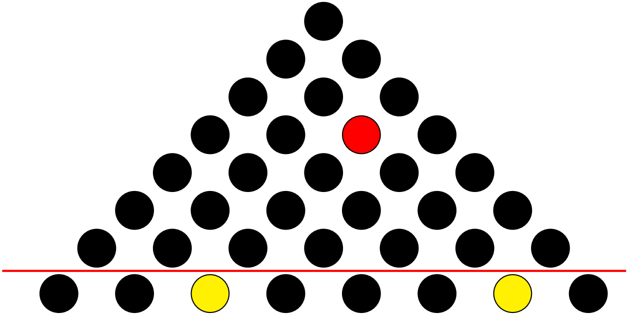
Example 1: On Monday, 8.00, Gandalf starts in the Shire and travels to Rivendell, where he has a date with Galadriel at 19.00. On Tuesday morning, he starts in Rivendell at 8.00 and travels back to the Shire along the same route. Show that there is a time of the day at which he was in the same spot on Monday and Tuesday.

Solution: Imagine two Gandalfs travelling on the same day. Clearly, they must meet.

Example 2: Picture Proof for $\Sigma_{i=1}^{n} i = \binom{n+1}{2}$.

From https://mathoverflow.net/questions/8846/proofs-without-words.

Example 2: Picture Proof for $\sum_{i=1}^{n} i = \binom{n+1}{2}$.



From `https://mathoverflow.net/questions/8846/proofs-without-words`.

Example 3: Do objects with a larger weight have a greater fall velocity? Imagine two objects with weight a and $b > a$. Join them with a rope. The compound object should (i) fall quicker because it has weight $a + b$ and (ii) fall slower because the objects with weight b is 'slowed down' by the object with weight a.

An essential part of mathematical understanding and proving consists in conducting thought experiments, looking a the problem from a new perspective etc.

This is an important part of mathematics - in particular of mathematical creativity - that has little to do with logical deduction.

On the other hand: Step-by-step deduction.

Example 4: Show that, for all sets A, B, C, we have $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

Proof: "$\subseteq$": Let $x \in A \cap (B \cup C)$. Then we have $x \in A$ and $x \in B \cup C$. Hence, we have $x \in B$ or $x \in C$. If $x \in B$, then $x \in A \cap B$ and hence $x \in (A \cap B) \cup (A \cap C)$. If $x \in C$, then $x \in A \cap C$ and hence $x \in (A \cap B) \cup (A \cap C)$. In any case, we have $x \in (A \cap B) \cup (A \cap C)$.

"$\supseteq$": ...

An essential part of mathematical understanding and proving consists in conducting thought experiments, looking a the problem from a new perspective etc.
This is an important part of mathematics - in particular of mathematical creativity - that has little to do with logical deduction.

On the other hand: Step-by-step deduction.

Example 4: Show that, for all sets A, B, C, we have $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

Proof: "$\subseteq$": Let $x \in A \cap (B \cup C)$. Then we have $x \in A$ and $x \in B \cup C$. Hence, we have $x \in B$ or $x \in C$. If $x \in B$, then $x \in A \cap B$ and hence $x \in (A \cap B) \cup (A \cap C)$. If $x \in C$, then $x \in A \cap C$ and hence $x \in (A \cap B) \cup (A \cap C)$. In any case, we have $x \in (A \cap B) \cup (A \cap C)$.

"$\supseteq$": ...

An essential part of mathematical understanding and proving consists in conducting thought experiments, looking a the problem from a new perspective etc.
This is an important part of mathematics - in particular of mathematical creativity - that has little to do with logical deduction.

On the other hand: Step-by-step deduction.

Example 4: Show that, for all sets A, B, C, we have $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

Proof: "$\subseteq$": Let $x \in A \cap (B \cup C)$. Then we have $x \in A$ and $x \in B \cup C$. Hence, we have $x \in B$ or $x \in C$. If $x \in B$, then $x \in A \cap B$ and hence $x \in (A \cap B) \cup (A \cap C)$. If $x \in C$, then $x \in A \cap C$ and hence $x \in (A \cap B) \cup (A \cap C)$. In any case, we have $x \in (A \cap B) \cup (A \cap C)$.

"$\supseteq$": ...

An essential part of mathematical understanding and proving consists in conducting thought experiments, looking a the problem from a new perspective etc.
This is an important part of mathematics - in particular of mathematical creativity - that has little to do with logical deduction.

On the other hand: Step-by-step deduction.

Example 4: Show that, for all sets A, B, C, we have $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

Proof: "$\subseteq$": Let $x \in A \cap (B \cup C)$. Then we have $x \in A$ and $x \in B \cup C$. Hence, we have $x \in B$ or $x \in C$. If $x \in B$, then $x \in A \cap B$ and hence $x \in (A \cap B) \cup (A \cap C)$. If $x \in C$, then $x \in A \cap C$ and hence $x \in (A \cap B) \cup (A \cap C)$. In any case, we have $x \in (A \cap B) \cup (A \cap C)$.

"$\supseteq$": ...

An essential part of mathematical understanding and proving consists in conducting thought experiments, looking a the problem from a new perspective etc.
This is an important part of mathematics - in particular of mathematical creativity - that has little to do with logical deduction.

On the other hand: Step-by-step deduction.

Example 4: Show that, for all sets A, B, C, we have $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

Proof: "$\subseteq$": Let $x \in A \cap (B \cup C)$. Then we have $x \in A$ and $x \in B \cup C$. Hence, we have $x \in B$ or $x \in C$. If $x \in B$, then $x \in A \cap B$ and hence $x \in (A \cap B) \cup (A \cap C)$. If $x \in C$, then $x \in A \cap C$ and hence $x \in (A \cap B) \cup (A \cap C)$. In any case, we have $x \in (A \cap B) \cup (A \cap C)$.

"$\supseteq$": ...

An essential part of mathematical understanding and proving consists in conducting thought experiments, looking a the problem from a new perspective etc.
This is an important part of mathematics - in particular of mathematical creativity - that has little to do with logical deduction.

On the other hand: Step-by-step deduction.

Example 4: Show that, for all sets A, B, C, we have $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

Proof: "$\subseteq$": Let $x \in A \cap (B \cup C)$. Then we have $x \in A$ and $x \in B \cup C$. Hence, we have $x \in B$ or $x \in C$. If $x \in B$, then $x \in A \cap B$ and hence $x \in (A \cap B) \cup (A \cap C)$. If $x \in C$, then $x \in A \cap C$ and hence $x \in (A \cap B) \cup (A \cap C)$. In any case, we have $x \in (A \cap B) \cup (A \cap C)$.
"$\supseteq$": ...

Checking creative though experiments is out of scope.

However, step-by-step dedductions of this kind are a central part of the mathematical armamentarium. If automatization can yield some progress here, it is certainly worth it.

Checking creative though experiments is out of scope.
However, step-by-step dedductions of this kind are a central part of the mathematical armamentarium. If automatization can yield some progress here, it is certainly worth it.

Goal

A system that takes proof attempts in a controlled form that is still natural for beginner's students, checks it and provides helpful feedback.

Feedback:
(1) Proofreading of natural language, type checking ("Suppose we have $7 + \emptyset$"), logical consistency, achievement check for proof goal.
(2) Hints for proof search: (1) general ("to show $A \to B$, try to assume A and to deduce B"), (2) specific (proof attempt is completed automatically (if possible) and then an intermediate step is proposed) (3) heuristical (hints explictly provided by the person who posed the exercise).
(3) Diagnosis of possible fallacies ("Anti-ATP").

Goal

A system that takes proof attempts in a controlled form that is still natural for beginner's students, checks it and provides helpful feedback.

Feedback:
(1) Proofreading of natural language, type checking ("Suppose we have $7 + \emptyset$"), logical consistency, achievement check for proof goal.
(2) Hints for proof search: (1) general ("to show $A \to B$, try to assume A and to deduce B"), (2) specific (proof attempt is completed automatically (if possible) and then an intermediate step is proposed) (3) heuristical (hints explictly provided by the person who posed the exercise).
(3) Diagnosis of possible fallacies ("Anti-ATP").

Goal

A system that takes proof attempts in a controlled form that is still natural for beginner's students, checks it and provides helpful feedback.

Feedback:
(1) Proofreading of natural language, type checking ("Suppose we have $7 + \emptyset$"), logical consistency, achievement check for proof goal.
(2) Hints for proof search: (1) general ("to show $A \to B$, try to assume A and to deduce B"), (2) specific (proof attempt is completed automatically (if possible) and then an intermediate step is proposed) (3) heuristical (hints explictly provided by the person who posed the exercise).
(3) Diagnosis of possible fallacies ("Anti-ATP").

Goal

A system that takes proof attempts in a controlled form that is still natural for beginner's students, checks it and provides helpful feedback.

Feedback:
(1) Proofreading of natural language, type checking ("Suppose we have $7 + \emptyset$"), logical consistency, achievement check for proof goal.
(2) Hints for proof search: (1) general ("to show $A \to B$, try to assume A and to deduce B"), (2) specific (proof attempt is completed automatically (if possible) and then an intermediate step is proposed) (3) heuristical (hints explictly provided by the person who posed the exercise).
(3) Diagnosis of possible fallacies ("Anti-ATP").

The model: Naproche

Naproche (NAtural PROof CHEcking) is a joint project of the logic group in Bonn and linguists from Duisburg-Essen (Bernhard Schröder), initiated by Peter Koepke. The system was essentially developed by Marcos Cramer in his dissertation. Further important contributors are Bernhard Fisseni, Daniel Kühlwein, Nikolay Kolev and others.

It is the goal of Naproche to develop an automated system for the verification of natural language mathematical proofs.

For example, the system can read a slightly adapted version of the first chapter of Edmund Landau's 'Foundations of Analysis' (and has considerably developed recently).

The model: Naproche

Naproche (NAtural PROof CHEcking) is a joint project of the logic group in Bonn and linguists from Duisburg-Essen (Bernhard Schröder), initiated by Peter Koepke. The system was essentially developed by Marcos Cramer in his dissertation. Further important contributors are Bernhard Fisseni, Daniel Kühlwein, Nikolay Kolev and others.

It is the goal of Naproche to develop an automated system for the verification of natural language mathematical proofs.

For example, the system can read a slightly adapted version of the first chapter of Edmund Landau's 'Foundations of Analysis' (and has considerably developed recently).

The model: Naproche

Naproche (NAtural PROof CHEcking) is a joint project of the logic group in Bonn and linguists from Duisburg-Essen (Bernhard Schröder), initiated by Peter Koepke. The system was essentially developed by Marcos Cramer in his dissertation. Further important contributors are Bernhard Fisseni, Daniel Kühlwein, Nikolay Kolev and others.

It is the goal of Naproche to develop an automated system for the verification of natural language mathematical proofs.

For example, the system can read a slightly adapted version of the first chapter of Edmund Landau's 'Foundations of Analysis' (and has considerably developed recently).

Theorem 9: Fix x, y. Then precisely one of the following cases holds: Case 1: $x = y$. Case 2: There is a u such that $x = y + u$. Case 3: There is a v such that $y = x + v$.
Proof: Fix x, y. By theorem 7, case 1 and case 2 are inconsistent and case 1 and case 3 are inconsistent. Suppose case 2 and case 3 hold. Then $x = y + u = (x + v) + u = x + (v + u) = (v + u) + x$. Contradiction by theorem 7. Thus case 2 and case 3 are inconsistent. Thus for all x, y, at most one of case 1, case 2 and case 3 holds. (...)

Theorem 9: For given x and y, exactly one of the following must be the case: 1) x=y 2) There exists a u such that $y = x + v$ 3) There exists v such that $y = x + v$
Proof: A) By Theorem 7, cases 1) and 2) are incompatible. Similarly, 1) and 3) are incompatible. The incompatibility of 2) and 3) also follows from Theorem 7: for otherwise, we would have
$x = y + u = (x + v) + u = x + (v + u) = (v + u) + x$. Therefore we can have at most one of the cases 1), 2) and 3). (...)

Naproche and Didactics

In spite of its strength, Naproche is not suitable for a didactical application in German universities, due to several reasons:

- The input language is English
- 'too smart': will accept steps that a beginner student should elaborate. (In many cases, it will accept the empty string as a sufficient proof for an exercise problem.)
- No possibility to control the admissible deduction rules; but teaching should start with demanding very basic steps and then gradually allow for greater leaps.
- 'too nice': It does not attempt to enforce a strict style of presentation. A lot of bad writing (such as adding irrelevant stuff after reaching the proof goal) will go through.
- No differentiated feedback: A sentence is either verified or it is not.
- No didactical extra functions, such as hints, a problem database etc.
- Not taylored to the language and prerequisites of beginner's exercises; e.g., no use of $\rightarrow$ for 'I am now showing one direction of an equivalence', no module for term manipulations or calculations such as $0 \leq (a-b)^2 = (a-b)(a-b) = (a^2 - ab - ba + b^2) = (a^2 - 2ab + b^2)$.

Naproche and Didactics

In spite of its strength, Naproche is not suitable for a didactical application in German universities, due to several reasons:

- The input language is English
- 'too smart': will accept steps that a beginner student should elaborate. (In many cases, it will accept the empty string as a sufficient proof for an exercise problem.)
- No possibility to control the admissible deduction rules; but teaching should start with demanding very basic steps and then gradually allow for greater leaps.
- 'too nice': It does not attempt to enforce a strict style of presentation. A lot of bad writing (such as adding irrelevant stuff after reaching the proof goal) will go through.
- No differentiated feedback: A sentence is either verified or it is not.
- No didactical extra functions, such as hints, a problem database etc.
- Not taylored to the language and prerequisites of beginner's exercises; e.g., no use of $\rightarrow$ for 'I am now showing one direction of an equivalence', no module for term manipulations or calculations such as $0 \leq (a-b)^2 = (a-b)(a-b) = (a^2 - ab - ba + b^2) = (a^2 - 2ab + b^2)$.

Naproche and Didactics

In spite of its strength, Naproche is not suitable for a didactical application in German universities, due to several reasons:

- The input language is English
- 'too smart': will accept steps that a beginner student should elaborate. (In many cases, it will accept the empty string as a sufficient proof for an exercise problem.)
- No possibility to control the admissible deduction rules; but teaching should start with demanding very basic steps and then gradually allow for greater leaps.
- 'too nice': It does not attempt to enforce a strict style of presentation. A lot of bad writing (such as adding irrelevant stuff after reaching the proof goal) will go through.
- No differentiated feedback: A sentence is either verified or it is not.
- No didactical extra functions, such as hints, a problem database etc.
- Not taylored to the language and prerequisites of beginner's exercises; e.g., no use of $\rightarrow$ for 'I am now showing one direction of an equivalence', no module for term manipulations or calculations such as $0 \leq (a-b)^2 = (a-b)(a-b) = (a^2 - ab - ba + b^2) = (a^2 - 2ab + b^2)$.

Naproche and Didactics

In spite of its strength, Naproche is not suitable for a didactical application in German universities, due to several reasons:

- The input language is English
- 'too smart': will accept steps that a beginner student should elaborate. (In many cases, it will accept the empty string as a sufficient proof for an exercise problem.)
- No possibility to control the admissible deduction rules; but teaching should start with demanding very basic steps and then gradually allow for greater leaps.
- 'too nice': It does not attempt to enforce a strict style of presentation. A lot of bad writing (such as adding irrelevant stuff after reaching the proof goal) will go through.
- No differentiated feedback: A sentence is either verified or it is not.
- No didactical extra functions, such as hints, a problem database etc.
- Not taylored to the language and prerequisites of beginner's exercises; e.g., no use of $\rightarrow$ for 'I am now showing one direction of an equivalence', no module for term manipulations or calculations such as $0 \leq (a-b)^2 = (a-b)(a-b) = (a^2 - ab - ba + b^2) = (a^2 - 2ab + b^2)$.

Naproche and Didactics

In spite of its strength, Naproche is not suitable for a didactical application in German universities, due to several reasons:

- The input language is English
- 'too smart': will accept steps that a beginner student should elaborate. (In many cases, it will accept the empty string as a sufficient proof for an exercise problem.)
- No possibility to control the admissible deduction rules; but teaching should start with demanding very basic steps and then gradually allow for greater leaps.
- 'too nice': It does not attempt to enforce a strict style of presentation. A lot of bad writing (such as adding irrelevant stuff after reaching the proof goal) will go through.
- No differentiated feedback: A sentence is either verified or it is not.
- No didactical extra functions, such as hints, a problem database etc.
- Not taylored to the language and prerequisites of beginner's exercises; e.g., no use of $\rightarrow$ for 'I am now showing one direction of an equivalence', no module for term manipulations or calculations such as $0 \leq (a-b)^2 = (a-b)(a-b) = (a^2 - ab - ba + b^2) = (a^2 - 2ab + b^2)$.

Naproche and Didactics

In spite of its strength, Naproche is not suitable for a didactical application in German universities, due to several reasons:

- The input language is English
- 'too smart': will accept steps that a beginner student should elaborate. (In many cases, it will accept the empty string as a sufficient proof for an exercise problem.)
- No possibility to control the admissible deduction rules; but teaching should start with demanding very basic steps and then gradually allow for greater leaps.
- 'too nice': It does not attempt to enforce a strict style of presentation. A lot of bad writing (such as adding irrelevant stuff after reaching the proof goal) will go through.
- No differentiated feedback: A sentence is either verified or it is not.
- No didactical extra functions, such as hints, a problem database etc.
- Not taylored to the language and prerequisites of beginner's exercises; e.g., no use of $\rightarrow$ for 'I am now showing one direction of an equivalence', no module for term manipulations or calculations such as $0 \leq (a-b)^2 = (a-b)(a-b) = (a^2 - ab - ba + b^2) = (a^2 - 2ab + b^2)$.

Naproche and Didactics

In spite of its strength, Naproche is not suitable for a didactical application in German universities, due to several reasons:

- The input language is English
- 'too smart': will accept steps that a beginner student should elaborate. (In many cases, it will accept the empty string as a sufficient proof for an exercise problem.)
- No possibility to control the admissible deduction rules; but teaching should start with demanding very basic steps and then gradually allow for greater leaps.
- 'too nice': It does not attempt to enforce a strict style of presentation. A lot of bad writing (such as adding irrelevant stuff after reaching the proof goal) will go through.
- No differentiated feedback: A sentence is either verified or it is not.
- No didactical extra functions, such as hints, a problem database etc.
- Not taylored to the language and prerequisites of beginner's exercises; e.g., no use of $\rightarrow$ for 'I am now showing one direction of an equivalence', no module for term manipulations or calculations such as $0 \leq (a-b)^2 = (a-b)(a-b) = (a^2 - ab - ba + b^2) = (a^2 - 2ab + b^2)$.

Naproche and Didactics

In spite of its strength, Naproche is not suitable for a didactical application in German universities, due to several reasons:

- The input language is English
- 'too smart': will accept steps that a beginner student should elaborate. (In many cases, it will accept the empty string as a sufficient proof for an exercise problem.)
- No possibility to control the admissible deduction rules; but teaching should start with demanding very basic steps and then gradually allow for greater leaps.
- 'too nice': It does not attempt to enforce a strict style of presentation. A lot of bad writing (such as adding irrelevant stuff after reaching the proof goal) will go through.
- No differentiated feedback: A sentence is either verified or it is not.
- No didactical extra functions, such as hints, a problem database etc.
- Not taylored to the language and prerequisites of beginner's exercises; e.g., no use of $\rightarrow$ for 'I am now showing one direction of an equivalence', no module for term manipulations or calculations such as $0 \leq (a-b)^2 = (a-b)(a-b) = (a^2 - ab - ba + b^2) = (a^2 - 2ab + b^2)$.

The goal is thus go come up a didactical “offshoot” of Naproche - namely “Diproche”. We will reuse the basic idea of the Naproche architecture, but no part of the code. Diproche is built up from scratch.

The goal is thus go come up a didactical “offshoot” of Naproche - namely “Diproche”. We will reuse the basic idea of the Naproche architecture, but no part of the code. Diproche is built up from scratch.

The goal is thus go come up a didactical “offshoot” of Naproche - namely “Diproche”. We will reuse the basic idea of the Naproche architecture, but no part of the code. Diproche is built up from scratch.

The architecture of the checking routine

Components:

- Preprocessing of the input string
- Annotation
- Determination of text structure
- Generation of ATP-tasks
- ATP

The architecture of the checking routine

Components:

- Preprocessing of the input string
- Annotation
- Determination of text structure
- Generation of ATP-tasks
- ATP

The architecture of the checking routine

Components:

- Preprocessing of the input string
- Annotation
- Determination of text structure
- Generation of ATP-tasks
- ATP

The architecture of the checking routine

Components:

- Preprocessing of the input string
- Annotation
- Determination of text structure
- Generation of ATP-tasks
- ATP

The architecture of the checking routine

Components:

- Preprocessing of the input string
- Annotation
- Determination of text structure
- Generation of ATP-tasks
- ATP

Example

Input: "Es gelte a. Ferner gelte $(a \rightarrow b)$. Dann folgt b."

list format: [[es,gelte,a],[ferner,gelte,[a,− >,b]],[dann,folgt,b]]

Annotated Format:
[[1,[],[],ann,bam,[]],[2,[a],[],ang,[],a],[3,[a,b],[],ang,[],[a,− >,b]],
[4,[b],[],beh,[],b],[5,[],[],ann,bem,[]]]

text structure graph: [[2,4],[3,4]] - encodes which assumptions are available at which text portions.

ATP-task (for "line" 4): [[a,[a,− >,b]],[b]]

ATP: Contains the (Prolog) rule "accept pairs [Vss,Y], where [X,− >,Y] and X belong to Vss".

Example

Input: "Es gelte a. Ferner gelte $(a \to b)$. Dann folgt b."

list format: [[es,gelte,a],[ferner,gelte,[a,$-$ $>$,b]],[dann,folgt,b]]

Annotated Format:
[[1,[],[],ann,bam,[]],[2,[a],[],ang,[],a],[3,[a,b],[],ang,[],[a,$-$ $>$,b]],
[4,[b],[],beh,[],b],[5,[],[],ann,bem,[]]]

text structure graph: [[2,4],[3,4]] - encodes which assumptions are available at which text portions.

ATP-task (for "line" 4): [[a,[a,$-$ $>$,b]],[b]]

ATP: Contains the (Prolog) rule "accept pairs [Vss,Y], where [X,$-$ $>$,Y] and X belong to Vss".

Example

Input: "Es gelte a. Ferner gelte $(a \rightarrow b)$. Dann folgt b."

list format: [[es,gelte,a],[ferner,gelte,[a,$->$,b]],[dann,folgt,b]]

Annotated Format:
[[1,[],[],ann,bam,[]],[2,[a],[],ang,[],a],[3,[a,b],[],ang,[],[a,$->$,b]],
[4,[b],[],beh,[],b],[5,[],[],ann,bem,[]]]

text structure graph: [[2,4],[3,4]] - encodes which assumptions are available at which text portions.

ATP-task (for "line" 4): [[a,[a,$->$,b]],[b]]

ATP: Contains the (Prolog) rule "accept pairs [Vss,Y], where [X,$->$,Y] and X belong to Vss".

Example

Input: "Es gelte a. Ferner gelte $(a \to b)$. Dann folgt b."

list format: [[es,gelte,a],[ferner,gelte,[a,$-$ $>$,b]],[dann,folgt,b]]

Annotated Format:
[[1,[],[],ann,bam,[]],[2,[a],[],ang,[],a],[3,[a,b],[],ang,[],[a,$-$ $>$,b]],
[4,[b],[],beh,[],b],[5,[],[],ann,bem,[]]]

text structure graph: [[2,4],[3,4]] - encodes which assumptions are available at which text portions.

ATP-task (for "line" 4): [[a,[a,$-$ $>$,b]],[b]]

ATP: Contains the (Prolog) rule "accept pairs [Vss,Y], where [X,$-$ $>$,Y] and X belong to Vss".

Example

Input: "Es gelte a. Ferner gelte ($a \to b$). Dann folgt b."

list format: [[es,gelte,a],[ferner,gelte,[a,$->$,b]],[dann,folgt,b]]

Annotated Format:
[[1,[],[],ann,bam,[]],[2,[a],[],ang,[],a],[3,[a,b],[],ang,[],[a,$->$,b]],
[4,[b],[],beh,[],b],[5,[],[],ann,bem,[]]]

text structure graph: [[2,4],[3,4]] - encodes which assumptions are available at which text portions.

ATP-task (for "line" 4): [[a,[a,$->$,b]],[b]]

ATP: Contains the (Prolog) rule "accept pairs [Vss,Y], where [X,$->$,Y] and X belong to Vss".

Example

Input: "Es gelte a. Ferner gelte $(a \rightarrow b)$. Dann folgt b."

list format: [[es,gelte,a],[ferner,gelte,[a,$->$,b]],[dann,folgt,b]]

Annotated Format:
[[1,[],[],ann,bam,[]],[2,[a],[],ang,[],a],[3,[a,b],[],ang,[],[a,$->$,b]],
[4,[b],[],beh,[],b],[5,[],[],ann,bem,[]]]

text structure graph: [[2,4],[3,4]] - encodes which assumptions are available at which text portions.

ATP-task (for "line" 4): [[a,[a,$->$,b]],[b]]

ATP: Contains the (Prolog) rule "accept pairs [Vss,Y], where [X,$->$,Y] and X belong to Vss".

The controlled ATP

The ATP is specifically designed to only accept proof steps that should be accepted as sufficient in the context of beginner's exercises.

It is thus intentionally weak: It only accepts a step when it matches one of the (many) rules in its database, i.e. there is no iteration in the proof search.

In addition, one can specify 'difficult degrees', i.e. subsets of prover rules, for specific exercises. In this way, the ATP can become more liberal as the student advances.

(For example, when proving the de Morgan rule, one should not have the de Morgan rule available; but for later exercises, one certainly should.)

The controlled ATP

The ATP is specifically designed to only accept proof steps that should be accepted as sufficient in the context of beginner's exercises.

It is thus intentionally weak: It only accepts a step when it matches one of the (many) rules in its database, i.e. there is no iteration in the proof search.

In addition, one can specify 'difficult degrees', i.e. subsets of prover rules, for specific exercises. In this way, the ATP can become more liberal as the student advances.

(For example, when proving the de Morgan rule, one should not have the de Morgan rule available; but for later exercises, one certainly should.)

The controlled ATP

The ATP is specifically designed to only accept proof steps that should be accepted as sufficient in the context of beginner's exercises.

It is thus intentionally weak: It only accepts a step when it matches one of the (many) rules in its database, i.e. there is no iteration in the proof search.

In addition, one can specify 'difficult degrees', i.e. subsets of prover rules, for specific exercises. In this way, the ATP can become more liberal as the student advances.

(For example, when proving the de Morgan rule, one should not have the de Morgan rule available; but for later exercises, one certainly should.)

The controlled ATP

The ATP is specifically designed to only accept proof steps that should be accepted as sufficient in the context of beginner's exercises.

It is thus intentionally weak: It only accepts a step when it matches one of the (many) rules in its database, i.e. there is no iteration in the proof search.

In addition, one can specify 'difficult degrees', i.e. subsets of prover rules, for specific exercises. In this way, the ATP can become more liberal as the student advances.

(For example, when proving the de Morgan rule, one should not have the de Morgan rule available; but for later exercises, one certainly should.)

The controlled ATP

The ATP is specifically designed to only accept proof steps that should be accepted as sufficient in the context of beginner's exercises.

It is thus intentionally weak: It only accepts a step when it matches one of the (many) rules in its database, i.e. there is no iteration in the proof search.

In addition, one can specify 'difficult degrees', i.e. subsets of prover rules, for specific exercises. In this way, the ATP can become more liberal as the student advances.

(For example, when proving the de Morgan rule, one should not have the de Morgan rule available; but for later exercises, one certainly should.)

The controlled ATP

The ATP is specifically designed to only accept proof steps that should be accepted as sufficient in the context of beginner's exercises.

It is thus intentionally weak: It only accepts a step when it matches one of the (many) rules in its database, i.e. there is no iteration in the proof search.

In addition, one can specify 'difficult degrees', i.e. subsets of prover rules, for specific exercises. In this way, the ATP can become more liberal as the student advances.

(For example, when proving the de Morgan rule, one should not have the de Morgan rule available; but for later exercises, one certainly should.)

Assumptions: “Es gelte/Angenommen, es gilt/Nehmen wir an, dass/...”

Claims: “Dann folgt/Es gilt/Damit haben wir/...”

Annotations: "Beweis:", "=>", "<=", "durch widerspruch", "wir zeigen, dass...", "qed", "Fall 1", paragraphs

An assumption is valid (only) in the paragraph in which it is introduced.
Exception: Assumptions made in the first paragraph after a proof start marker (“Beweis”) are valid until the (sub-)proof is declared to be finished by a proof ending marker (“qed”).

Assumptions: “Es gelte/Angenommen, es gilt/Nehmen wir an, dass/...”

Claims: “Dann folgt/Es gilt/Damit haben wir/...”

Annotations: "Beweis:", "=>", "<=", "durch widerspruch", "wir zeigen, dass...", "qed", "Fall 1", paragraphs

An assumption is valid (only) in the paragraph in which it is introduced.
Exception: Assumptions made in the first paragraph after a proof start marker (“Beweis”) are valid until the (sub-)proof is declared to be finished by a proof ending marker (“qed”).

The Diproche Language

Assumptions: “Es gelte/Angenommen, es gilt/Nehmen wir an, dass/...”

Claims: “Dann folgt/Es gilt/Damit haben wir/...”

Annotations: "Beweis:", "=>", "<=", "durch widerspruch", "wir zeigen, dass...", "qed", "Fall 1", paragraphs

An assumption is valid (only) in the paragraph in which it is introduced.
Exception: Assumptions made in the first paragraph after a proof start marker (“Beweis”) are valid until the (sub-)proof is declared to be finished by a proof ending marker (“qed”).

Assumptions: "Es gelte/Angenommen, es gilt/Nehmen wir an, dass/..."

Claims: "Dann folgt/Es gilt/Damit haben wir/..."

Annotations: "Beweis:", "=>", "<=", "durch widerspruch", "wir zeigen, dass...", "qed", "Fall 1", paragraphs

An assumption is valid (only) in the paragraph in which it is introduced.
Exception: Assumptions made in the first paragraph after a proof start marker ("Beweis") are valid until the (sub-)proof is declared to be finished by a proof ending marker ("qed").

The Diproche Language

Assumptions: "Es gelte/Angenommen, es gilt/Nehmen wir an, dass/..."

Claims: "Dann folgt/Es gilt/Damit haben wir/..."

Annotations: "Beweis:", "=>", "<=", "durch widerspruch", "wir zeigen, dass...", "qed", "Fall 1", paragraphs

An assumption is valid (only) in the paragraph in which it is introduced.
Exception: Assumptions made in the first paragraph after a proof start marker ("Beweis") are valid until the (sub-)proof is declared to be finished by a proof ending marker ("qed").

Wir zeigen $((a \vee b) \leftrightarrow ((a \rightarrow b) \rightarrow b))$.
$=>$ Angenommen, es gilt $(a \vee b)$. Angenommen ferner, es gilt $(a \rightarrow b)$. Falls a gilt, so gilt b. Falls b gilt, so gilt ebenfalls b. Also gilt b.
Damit folgt $((a \rightarrow b) \rightarrow b)$.
qed.
$<=$
Angenommen, es gilt $((a \rightarrow b) \rightarrow b)$. Nehmen wir an, es gilt $\neg a$.
Dann gilt auch $(a \rightarrow b)$. Damit folgt b.
Also folgt $(\neg a \rightarrow b)$. Damit folgt $(a \vee b)$.
qed.
Also gilt auch $(((a \rightarrow b) \rightarrow b) \rightarrow (avb))$.
Damit folgt nun endlich $((a \vee b) \leftrightarrow ((a \rightarrow b) \rightarrow b))$.
Qed.

Sample text: propositional logic

Wir zeigen $((a \vee b) \leftrightarrow ((a \rightarrow b) \rightarrow b))$.
=> Angenommen, es gilt $(a \vee b)$. Angenommen ferner, es gilt $(a \rightarrow b)$. Falls a gilt, so gilt b. Falls b gilt, so gilt ebenfalls b. Also gilt b.
Damit folgt $((a \rightarrow b) \rightarrow b)$.
qed.
<=
Angenommen, es gilt $((a \rightarrow b) \rightarrow b)$. Nehmen wir an, es gilt $\neg a$.
Dann gilt auch $(a \rightarrow b)$. Damit folgt b.
Also folgt $(\neg a \rightarrow b)$. Damit folgt $(a \vee b)$.
qed.
Also gilt auch $(((a \rightarrow b) \rightarrow b) \rightarrow (avb))$.
Damit folgt nun endlich $((a \vee b) \leftrightarrow ((a \rightarrow b) \rightarrow b))$.
Qed.

Sample text: propositional logic

Wir zeigen $((a \vee b) \leftrightarrow ((a \rightarrow b) \rightarrow b))$.
=> Angenommen, es gilt $(a \vee b)$. Angenommen ferner, es gilt $(a \rightarrow b)$. Falls a gilt, so gilt b. Falls b gilt, so gilt ebenfalls b. Also gilt b.
Damit folgt $((a \rightarrow b) \rightarrow b)$.
qed.
<=
Angenommen, es gilt $((a \rightarrow b) \rightarrow b)$. Nehmen wir an, es gilt $\neg a$.
Dann gilt auch $(a \rightarrow b)$. Damit folgt b.
Also folgt $(\neg a \rightarrow b)$. Damit folgt $(a \vee b)$.
qed.
Also gilt auch $(((a \rightarrow b) \rightarrow b) \rightarrow (avb))$.
Damit folgt nun endlich $((a \vee b) \leftrightarrow ((a \rightarrow b) \rightarrow b))$.
Qed.

Sample text: Boolean set theory

Es seien A, B, C und D Mengen. Es gelte $(A \cap B) = \emptyset$. Ferner gelte $(C \subseteq A)$ und $(D \subseteq B)$. Wir zeigen $(C \cap D) = \emptyset$.

Beweis.

Es gelte $x \in (C \cap D)$. Dann folgt $(x \in C)$. Also folgt $(x \in A)$. Ferner gilt $(x \in D)$. Damit gilt auch $(x \in B)$. Damit haben wir $(x \in (A \cap B))$.
Also gilt $(x \in \emptyset)$. Widerspruch.

Also gilt $(C \cap D) = \emptyset$.

qed.

Sample text: Boolean set theory

Es seien A, B, C und D Mengen. Es gelte $(A \cap B) = \emptyset$. Ferner gelte $(C \subseteq A)$ und $(D \subseteq B)$. Wir zeigen $(C \cap D) = \emptyset$.

Beweis.

Es gelte $x \in (C \cap D)$. Dann folgt $(x \in C)$. Also folgt $(x \in A)$. Ferner gilt $(x \in D)$. Damit gilt auch $(x \in B)$. Damit haben wir $(x \in (A \cap B))$.
Also gilt $(x \in \emptyset)$. Widerspruch.

Also gilt $(C \cap D) = \emptyset$.

qed.

Sample text: Boolean set theory

Es seien A, B, C und D Mengen. Es gelte $(A \cap B) = \emptyset$. Ferner gelte $(C \subseteq A)$ und $(D \subseteq B)$. Wir zeigen $(C \cap D) = \emptyset$.

Beweis.

Es gelte $x \in (C \cap D)$. Dann folgt $(x \in C)$. Also folgt $(x \in A)$. Ferner gilt $(x \in D)$. Damit gilt auch $(x \in B)$. Damit haben wir $(x \in (A \cap B))$.
Also gilt $(x \in \emptyset)$. Widerspruch.

Also gilt $(C \cap D) = \emptyset$.

qed.

Sample text: even/odd

Es sei n eine natuerliche Zahl. Wir zeigen $n(n+1)$ ist gerade.

Beweis.

Angenommen, n ist gerade. Dann ist auch $n(n+1)$ gerade.

Nehmen wir nun an, n ist ungerade. Dann ist $(n+1)$ gerade. Damit ist $n(n+1)$ ebenfalls gerade.

Also ist $n(n+1)$ gerade.

qed.

Sample text: even/odd

Es sei n eine natuerliche Zahl. Wir zeigen $n(n+1)$ ist gerade.

Beweis.

Angenommen, n ist gerade. Dann ist auch $n(n+1)$ gerade.

Nehmen wir nun an, n ist ungerade. Dann ist $(n+1)$ gerade. Damit ist $n(n+1)$ ebenfalls gerade.

Also ist $n(n+1)$ gerade.

qed.

Sample text: even/odd

Es sei n eine natuerliche Zahl. Wir zeigen $n(n+1)$ ist gerade.

Beweis.

Angenommen, n ist gerade. Dann ist auch $n(n+1)$ gerade.

Nehmen wir nun an, n ist ungerade. Dann ist $(n+1)$ gerade. Damit ist $n(n+1)$ ebenfalls gerade.

Also ist $n(n+1)$ gerade.

qed.

Sample text: even/odd

Es sei n eine natuerliche Zahl. Wir zeigen $n(n+1)$ ist gerade.

Beweis.

Angenommen, n ist gerade. Dann ist auch $n(n+1)$ gerade.

Nehmen wir nun an, n ist ungerade. Dann ist $(n+1)$ gerade. Damit ist $n(n+1)$ ebenfalls gerade.

Also ist $n(n+1)$ gerade.

qed.

Sample text: even/odd

Es sei n eine natuerliche Zahl. Wir zeigen $n(n+1)$ ist gerade.

Beweis.

Angenommen, n ist gerade. Dann ist auch $n(n+1)$ gerade.

Nehmen wir nun an, n ist ungerade. Dann ist $(n+1)$ gerade. Damit ist $n(n+1)$ ebenfalls gerade.

Also ist $n(n+1)$ gerade.

qed.

Sample text: even/odd

Es sei n eine natuerliche Zahl. Wir zeigen $n(n+1)$ ist gerade.

Beweis.

Angenommen, n ist gerade. Dann ist auch $n(n+1)$ gerade.

Nehmen wir nun an, n ist ungerade. Dann ist $(n+1)$ gerade. Damit ist $n(n+1)$ ebenfalls gerade.

Also ist $n(n+1)$ gerade.

qed.

Sample text: even/odd

Es sei n eine natuerliche Zahl. Wir zeigen $n(n+1)$ ist gerade.

Beweis.

Angenommen, n ist gerade. Dann ist auch $n(n+1)$ gerade.

Nehmen wir nun an, n ist ungerade. Dann ist $(n+1)$ gerade. Damit ist $n(n+1)$ ebenfalls gerade.

Also ist $n(n+1)$ gerade.

qed.

Sample text: Induction, Gauß sum

Es gilt $\Sigma_{i=1}^{1} i = 1 = \frac{1\cdot(1+1)}{2}$.
Es gelte $\Sigma_{i=1}^{n} i = \frac{n(n+1)}{2}$.
Dann folgt $\Sigma_{i=1}^{n+1} i = \Sigma_{i=1}^{n} i + (n+1) = \frac{n(n+1)}{2} + (n+1) = \frac{(n+1)(n+2)}{2} = \frac{(n+1)((n+1)+1)}{2}$.
Also gilt $\Sigma_{i=1}^{n+1} i = \frac{(n+1)((n+1)+1)}{2}$.
Also haben wir $\Sigma_{i=1}^{n} i = \frac{n(n+1)}{2} \to \Sigma_{i=1}^{n+1} i = \frac{(n+1)((n+1)+1)}{2}$.
Damit folgt $\forall n \Sigma_{i=1}^{n} i = \frac{n(n+1)}{2} \to \Sigma_{i=1}^{n+1} i = \frac{(n+1)((n+1)+1)}{2}$.
Induktiv folgt nun $\forall n \Sigma_{i=1}^{n} i = \frac{n(n+1)}{2}$.
Qed.

Sample text: Induction, Gauß sum

Es gilt $\Sigma_{i=1}^{1} i = 1 = \frac{1\cdot(1+1)}{2}$.
Es gelte $\Sigma_{i=1}^{n} i = \frac{n(n+1)}{2}$.
Dann folgt $\Sigma_{i=1}^{n+1} i = \Sigma_{i=1}^{n} i + (n+1) = \frac{n(n+1)}{2} + (n+1) = \frac{(n+1)(n+2)}{2} = \frac{(n+1)((n+1)+1)}{2}$.
Also gilt $\Sigma_{i=1}^{n+1} i = \frac{(n+1)((n+1)+1)}{2}$.
Also haben wir $\Sigma_{i=1}^{n} i = \frac{n(n+1)}{2} \to \Sigma_{i=1}^{n+1} i = \frac{(n+1)((n+1)+1)}{2}$.
Damit folgt $\forall n \Sigma_{i=1}^{n} i = \frac{n(n+1)}{2} \to \Sigma_{i=1}^{n+1} i = \frac{(n+1)((n+1)+1)}{2}$.
Induktiv folgt nun $\forall n \Sigma_{i=1}^{n} i = \frac{n(n+1)}{2}$.
Qed.

Goaltracing

In Diproche, one can declare proof goals through formulations such as:

Wir zeigen: Für alle Mengen A, B mit $A \subseteq B$ gilt $(A \cup B) = B$.

The goal tracing then attempts to determine the current goal at each position in the proof test.
For example, the annotation "$\Rightarrow$" has the effect that the current proof goal is changed from $(\phi \leftrightarrow \psi$ to $(\phi \rightarrow \psi)$
It is then checked for every proof ending marker whether the corresponding goal has indeed been reached. The goal only counts as reached when it is explicitly stated as the final statement before the endmarker.

Goaltracing

In Diproche, one can declare proof goals through formulations such as:

Wir zeigen: Für alle Mengen A, B mit $A \subseteq B$ gilt $(A \cup B) = B$.

The goal tracing then attempts to determine the current goal at each position in the proof test.
For example, the annotation "$\Rightarrow$" has the effect that the current proof goal is changed from $(\phi \leftrightarrow \psi$ to $(\phi \rightarrow \psi)$
It is then checked for every proof ending marker whether the corresponding goal has indeed been reached. The goal only counts as reached when it is explicitly stated as the final statement before the endmarker.

Goaltracing

In Diproche, one can declare proof goals through formulations such as:

Wir zeigen: Für alle Mengen A, B mit $A \subseteq B$ gilt $(A \cup B) = B$.

The goal tracing then attempts to determine the current goal at each position in the proof test.

For example, the annotation "$\Rightarrow$" has the effect that the current proof goal is changed from $(\phi \leftrightarrow \psi$ to $(\phi \rightarrow \psi)$

It is then checked for every proof ending marker whether the corresponding goal has indeed been reached. The goal only counts as reached when it is explicitly stated as the final statement before the endmarker.

Goaltracing

In Diproche, one can declare proof goals through formulations such as:

Wir zeigen: Für alle Mengen A, B mit $A \subseteq B$ gilt $(A \cup B) = B$.

The goal tracing then attempts to determine the current goal at each position in the proof test.
For example, the annotation "$\Rightarrow$" has the effect that the current proof goal is changed from $(\phi \leftrightarrow \psi$ to $(\phi \rightarrow \psi)$
It is then checked for every proof ending marker whether the corresponding goal has indeed been reached. The goal only counts as reached when it is explicitly stated as the final statement before the endmarker.

Goaltracing

In Diproche, one can declare proof goals through formulations such as:

Wir zeigen: Für alle Mengen A, B mit $A \subseteq B$ gilt $(A \cup B) = B$.

The goal tracing then attempts to determine the current goal at each position in the proof test.
For example, the annotation "$\Rightarrow$" has the effect that the current proof goal is changed from $(\phi \leftrightarrow \psi$ to $(\phi \rightarrow \psi)$
It is then checked for every proof ending marker whether the corresponding goal has indeed been reached. The goal only counts as reached when it is explicitly stated as the final statement before the endmarker.

Goaltracing

In Diproche, one can declare proof goals through formulations such as:

Wir zeigen: Für alle Mengen A, B mit $A \subseteq B$ gilt $(A \cup B) = B$.

The goal tracing then attempts to determine the current goal at each position in the proof test.
For example, the annotation "$\Rightarrow$" has the effect that the current proof goal is changed from $(\phi \leftrightarrow \psi$ to $(\phi \rightarrow \psi)$
It is then checked for every proof ending marker whether the corresponding goal has indeed been reached. The goal only counts as reached when it is explicitly stated as the final statement before the endmarker.

Type-Checking

Many beginner's mistakes happen at the type level. Either objects are used in a way they cannot ('Assume $7 + \emptyset$') or they are used without having been declared at all.
This is taken care of by a type-checking algorithm. Declaration can be made by formulations such as

"Es seien a, b und c natuerliche Zahlen und d eine reelle Zahl'."

In particular, it is checked whether the content of an assumption or a claim is indeed a proposition.

Type-Checking

Many beginner's mistakes happen at the type level. Either objects are used in a way they cannot ('Assume $7 + \emptyset$') or they are used without having been declared at all.
This is taken care of by a type-checking algorithm. Declaration can be made by formulations such as

"Es seien a, b und c natuerliche Zahlen und d eine reelle Zahl'."

In particular, it is checked whether the content of an assumption or a claim is indeed a proposition.

Type-Checking

Many beginner's mistakes happen at the type level. Either objects are used in a way they cannot ('Assume $7 + \emptyset$') or they are used without having been declared at all.

This is taken care of by a type-checking algorithm. Declaration can be made by formulations such as

"Es seien *a*, *b* und *c* natuerliche Zahlen und *d* eine reelle Zahl'."

In particular, it is checked whether the content of an assumption or a claim is indeed a proposition.

Type-Checking

Many beginner's mistakes happen at the type level. Either objects are used in a way they cannot ('Assume $7 + \emptyset$') or they are used without having been declared at all.

This is taken care of by a type-checking algorithm. Declaration can be made by formulations such as

> "Es seien a, b und c natuerliche Zahlen und d eine reelle Zahl'."

In particular, it is checked whether the content of an assumption or a claim is indeed a proposition.

Type-Checking

Many beginner's mistakes happen at the type level. Either objects are used in a way they cannot ('Assume $7 + \emptyset$') or they are used without having been declared at all.
This is taken care of by a type-checking algorithm. Declaration can be made by formulations such as

"Es seien a, b und c natuerliche Zahlen und d eine reelle Zahl'."

In particular, it is checked whether the content of an assumption or a claim is indeed a proposition.

Type-Checking

Many beginner's mistakes happen at the type level. Either objects are used in a way they cannot ('Assume $7 + \emptyset$') or they are used without having been declared at all.
This is taken care of by a type-checking algorithm. Declaration can be made by formulations such as

"Es seien a, b und c natuerliche Zahlen und d eine reelle Zahl'."

In particular, it is checked whether the content of an assumption or a claim is indeed a proposition.

Fallacy diagnosis: The Anti-ATP

A good tutor cannot just tell whether a step is right or wrong, but only have an educated guess at what's behind a mistake.

On the one hand, this allows one to specifically address misconceptions, and on the other hand, one can recognize and encourage possibly good ideas in a proof attempt.

At least the first function can be realized in Diproche to a certain extent.

A good tutor cannot just tell whether a step is right or wrong, but only have an educated guess at what's behind a mistake.
On the one hand, this allows one to specifically address misconceptions, and on the other hand, one can recognize and encourage possibly good ideas in a proof attempt.
At least the first function can be realized in Diproche to a certain extent.

Fallacy diagnosis: The Anti-ATP

A good tutor cannot just tell whether a step is right or wrong, but only have an educated guess at what's behind a mistake.
On the one hand, this allows one to specifically address misconceptions, and on the other hand, one can recognize and encourage possibly good ideas in a proof attempt.
At least the first function can be realized in Diproche to a certain extent.

A good tutor cannot just tell whether a step is right or wrong, but only have an educated guess at what's behind a mistake.
On the one hand, this allows one to specifically address misconceptions, and on the other hand, one can recognize and encourage possibly good ideas in a proof attempt.
At least the first function can be realized in Diproche to a certain extent.

Fallacy diagnosis: The Anti-ATP

A good tutor cannot just tell whether a step is right or wrong, but only have an educated guess at what's behind a mistake.
On the one hand, this allows one to specifically address misconceptions, and on the other hand, one can recognize and encourage possibly good ideas in a proof attempt.
At least the first function can be realized in Diproche to a certain extent.

The Anti-ATP works like an ATP, but has typical formal fallacies instead of sound proof rules, such as.

- From $A \vee B$ and A, derive $\neg B$
- From $A \rightarrow B$ and $\neg A$, derive $\neg B$
- From $\neg A$, derive $\neg(A \rightarrow B)$
- From $\neg(A \wedge B)$, derive $\neg A \wedge \neg B$

Each of these fallacies has an internal index. When the ATP cannot verify a step, it is passed on to the anti-ATP. When the proof step in question is realizable by a formal fallacy, an according feedback is given.

The Anti-ATP works like an ATP, but has typical formal fallacies instead of sound proof rules, such as.

- From $A \vee B$ and A, derive $\neg B$
- From $A \rightarrow B$ and $\neg A$, derive $\neg B$
- From $\neg A$, derive $\neg(A \rightarrow B)$
- From $\neg(A \wedge B)$, derive $\neg A \wedge \neg B$

Each of these fallacies has an internal index. When the ATP cannot verify a step, it is passed on to the anti-ATP. When the proof step in question is realizable by a formal fallacy, an according feedback is given.

The Anti-ATP works like an ATP, but has typical formal fallacies instead of sound proof rules, such as.

- From $A \vee B$ and A, derive $\neg B$
- From $A \rightarrow B$ and $\neg A$, derive $\neg B$
- From $\neg A$, derive $\neg(A \rightarrow B)$
- From $\neg(A \wedge B)$, derive $\neg A \wedge \neg B$

Each of these fallacies has an internal index. When the ATP cannot verify a step, it is passed on to the anti-ATP. When the proof step in question is realizable by a formal fallacy, an according feedback is given.

The Anti-ATP works like an ATP, but has typical formal fallacies instead of sound proof rules, such as.

- From $A \vee B$ and A, derive $\neg B$
- From $A \rightarrow B$ and $\neg A$, derive $\neg B$
- From $\neg A$, derive $\neg(A \rightarrow B)$
- From $\neg(A \wedge B)$, derive $\neg A \wedge \neg B$

Each of these fallacies has an internal index. When the ATP cannot verify a step, it is passed on to the anti-ATP. When the proof step in question is realizable by a formal fallacy, an according feedback is given.

The Anti-ATP works like an ATP, but has typical formal fallacies instead of sound proof rules, such as.

- From $A \vee B$ and A, derive $\neg B$
- From $A \rightarrow B$ and $\neg A$, derive $\neg B$
- From $\neg A$, derive $\neg(A \rightarrow B)$
- From $\neg(A \wedge B)$, derive $\neg A \wedge \neg B$

Each of these fallacies has an internal index. When the ATP cannot verify a step, it is passed on to the anti-ATP. When the proof step in question is realizable by a formal fallacy, an according feedback is given.

The Anti-ATP works like an ATP, but has typical formal fallacies instead of sound proof rules, such as.

- From $A \vee B$ and A, derive $\neg B$
- From $A \rightarrow B$ and $\neg A$, derive $\neg B$
- From $\neg A$, derive $\neg(A \rightarrow B)$
- From $\neg(A \wedge B)$, derive $\neg A \wedge \neg B$

Each of these fallacies has an internal index. When the ATP cannot verify a step, it is passed on to the anti-ATP. When the proof step in question is realizable by a formal fallacy, an according feedback is given.

The Anti-ATP works like an ATP, but has typical formal fallacies instead of sound proof rules, such as.

- From $A \vee B$ and A, derive $\neg B$
- From $A \rightarrow B$ and $\neg A$, derive $\neg B$
- From $\neg A$, derive $\neg(A \rightarrow B)$
- From $\neg(A \wedge B)$, derive $\neg A \wedge \neg B$

Each of these fallacies has an internal index. When the ATP cannot verify a step, it is passed on to the anti-ATP. When the proof step in question is realizable by a formal fallacy, an according feedback is given.

The Anti-ATP works like an ATP, but has typical formal fallacies instead of sound proof rules, such as.

- From $A \vee B$ and A, derive $\neg B$
- From $A \to B$ and $\neg A$, derive $\neg B$
- From $\neg A$, derive $\neg(A \to B)$
- From $\neg(A \wedge B)$, derive $\neg A \wedge \neg B$

Each of these fallacies has an internal index. When the ATP cannot verify a step, it is passed on to the anti-ATP. When the proof step in question is realizable by a formal fallacy, an according feedback is given.

Anti-Terms

The anti-ATP has a subroutine for false term manipluations, such as:

- $\frac{n^2}{n^3} = \frac{2}{3}$
- $\frac{n^2}{n^4} = \frac{n}{n^2}$
- $\frac{a}{b} + \frac{c}{d} = \frac{a+c}{b+d}$
- ...

These are dealt with in the same way as in the anti-ATP.

Anti-Terms

The anti-ATP has a subroutine for false term manipluations, such as:

- $\frac{n^2}{n^3} = \frac{2}{3}$
- $\frac{n^2}{n^4} = \frac{n}{n^2}$
- $\frac{a}{b} + \frac{c}{d} = \frac{a+c}{b+d}$
- ...

These are dealt with in the same way as in the anti-ATP.

Anti-Terms

The anti-ATP has a subroutine for false term manipluations, such as:

- $\frac{n^2}{n^3} = \frac{2}{3}$
- $\frac{n^2}{n^4} = \frac{n}{n^2}$
- $\frac{a}{b} + \frac{c}{d} = \frac{a+c}{b+d}$
- ...

These are dealt with in the same way as in the anti-ATP.

Anti-Terms

The anti-ATP has a subroutine for false term manipluations, such as:

- $\frac{n^2}{n^3} = \frac{2}{3}$
- $\frac{n^2}{n^4} = \frac{n}{n^2}$
- $\frac{a}{b} + \frac{c}{d} = \frac{a+c}{b+d}$
- ...

These are dealt with in the same way as in the anti-ATP.

Providing hints

The system provides three kinds of hints that users can request when they are stuck at a certain point in a proof:

- Prescribed, problem-specific hints. These are entered by the teacher by hand and resemble those hints that are typically found in the 'solution' part of a textbook.
- General strategical hints, either based on the goal ("In order to prove $A \to B$, assume A and try to prove B") or based on the assumptions ("When you have a disjunction in your assumptions, try a case distinction").
- Proposed intermediate steps: A 'real' (unrestricted) ATP attempts to complete the proof. If that works in a certain number of steps, a step from the 'middle' of that proof is offered as an intermediate step.

Providing hints

The system provides three kinds of hints that users can request when they are stuck at a certain point in a proof:

- Prescribed, problem-specific hints. These are entered by the teacher by hand and resemble those hints that are typically found in the 'solution' part of a textbook.
- General strategical hints, either based on the goal ("In order to prove $A \to B$, assume A and try to prove B") or based on the assumptions ("When you have a disjunction in your assumptions, try a case distinction").
- Proposed intermediate steps: A 'real' (unrestricted) ATP attempts to complete the proof. If that works in a certain number of steps, a step from the 'middle' of that proof is offered as an intermediate step.

Providing hints

The system provides three kinds of hints that users can request when they are stuck at a certain point in a proof:

- Prescribed, problem-specific hints. These are entered by the teacher by hand and resemble those hints that are typically found in the 'solution' part of a textbook.
- General strategical hints, either based on the goal ("In order to prove $A \to B$, assume A and try to prove B") or based on the assumptions ("When you have a disjunction in your assumptions, try a case distinction").
- Proposed intermediate steps: A 'real' (unrestricted) ATP attempts to complete the proof. If that works in a certain number of steps, a step from the 'middle' of that proof is offered as an intermediate step.

Providing hints

The system provides three kinds of hints that users can request when they are stuck at a certain point in a proof:

- Prescribed, problem-specific hints. These are entered by the teacher by hand and resemble those hints that are typically found in the 'solution' part of a textbook.
- General strategical hints, either based on the goal ("In order to prove $A \to B$, assume A and try to prove B") or based on the assumptions ("When you have a disjunction in your assumptions, try a case distinction").
- Proposed intermediate steps: A 'real' (unrestricted) ATP attempts to complete the proof. If that works in a certain number of steps, a step from the 'middle' of that proof is offered as an intermediate step.

Providing hints

The system provides three kinds of hints that users can request when they are stuck at a certain point in a proof:

- Prescribed, problem-specific hints. These are entered by the teacher by hand and resemble those hints that are typically found in the 'solution' part of a textbook.
- General strategical hints, either based on the goal ("In order to prove $A \to B$, assume A and try to prove B") or based on the assumptions ("When you have a disjunction in your assumptions, try a case distinction").
- Proposed intermediate steps: A 'real' (unrestricted) ATP attempts to complete the proof. If that works in a certain number of steps, a step from the 'middle' of that proof is offered as an intermediate step.

Providing hints

The system provides three kinds of hints that users can request when they are stuck at a certain point in a proof:

- Prescribed, problem-specific hints. These are entered by the teacher by hand and resemble those hints that are typically found in the 'solution' part of a textbook.
- General strategical hints, either based on the goal ("In order to prove $A \to B$, assume A and try to prove B") or based on the assumptions ("When you have a disjunction in your assumptions, try a case distinction").
- Proposed intermediate steps: A 'real' (unrestricted) ATP attempts to complete the proof. If that works in a certain number of steps, a step from the 'middle' of that proof is offered as an intermediate step.

Problem Database

For each area covered by Diproche, one can enter problems.

A problem consists of:

- An index.
- A verbal formulation for the user.
- A formalization of the proof goal.
- A specification of the degree of difficulty (=set of prover rules).
- A list of assumptions one may use during the solution.
- A list of declarations that can be used throughout the exercise.

Moreover, each problem index is associated with a (possibly empty) list of prescribed hints and (sometimes) a template solution.

Problem Database

For each area covered by Diproche, one can enter problems.
A problem consists of:

- An index.
- A verbal formulation for the user.
- A formalization of the proof goal.
- A specification of the degree of difficulty (=set of prover rules).
- A list of assumptions one may use during the solution.
- A list of declarations that can be used throughout the exercise.

Moreover, each problem index is associated with a (possibly empty) list of prescribed hints and (sometimes) a template solution.

Problem Database

For each area covered by Diproche, one can enter problems.
A problem consists of:

- An index.
- A verbal formulation for the user.
- A formalization of the proof goal.
- A specification of the degree of difficulty (=set of prover rules).
- A list of assumptions one may use during the solution.
- A list of declarations that can be used throughout the exercise.

Moreover, each problem index is associated with a (possibly empty) list of prescribed hints and (sometimes) a template solution.

Problem Database

For each area covered by Diproche, one can enter problems.
A problem consists of:

- An index.
- A verbal formulation for the user.
- A formalization of the proof goal.
- A specification of the degree of difficulty (=set of prover rules).
- A list of assumptions one may use during the solution.
- A list of declarations that can be used throughout the exercise.

Moreover, each problem index is associated with a (possibly empty) list of prescribed hints and (sometimes) a template solution.

Problem Database

For each area covered by Diproche, one can enter problems.
A problem consists of:

- An index.
- A verbal formulation for the user.
- A formalization of the proof goal.
- A specification of the degree of difficulty (=set of prover rules).
- A list of assumptions one may use during the solution.
- A list of declarations that can be used throughout the exercise.

Moreover, each problem index is associated with a (possibly empty) list of prescribed hints and (sometimes) a template solution.

Problem Database

For each area covered by Diproche, one can enter problems.
A problem consists of:

- An index.
- A verbal formulation for the user.
- A formalization of the proof goal.
- A specification of the degree of difficulty (=set of prover rules).
- A list of assumptions one may use during the solution.
- A list of declarations that can be used throughout the exercise.

Moreover, each problem index is associated with a (possibly empty) list of prescribed hints and (sometimes) a template solution.

Problem Database

For each area covered by Diproche, one can enter problems.
A problem consists of:

- An index.
- A verbal formulation for the user.
- A formalization of the proof goal.
- A specification of the degree of difficulty (=set of prover rules).
- A list of assumptions one may use during the solution.
- A list of declarations that can be used throughout the exercise.

Moreover, each problem index is associated with a (possibly empty) list of prescribed hints and (sometimes) a template solution.

Problem Database

For each area covered by Diproche, one can enter problems.
A problem consists of:

- An index.
- A verbal formulation for the user.
- A formalization of the proof goal.
- A specification of the degree of difficulty (=set of prover rules).
- A list of assumptions one may use during the solution.
- A list of declarations that can be used throughout the exercise.

Moreover, each problem index is associated with a (possibly empty) list of prescribed hints and (sometimes) a template solution.

Problem Generators

In order to increase the available training material, it is desirable to generate practice problems automatically. Currently, there are routines for automatically generating problems in the following areas:

- Propositional logic.
- Boolean Set Theory.
- Induction (Divisibility).
- Induction (Inequalities).

Problem Generators

In order to increase the available training material, it is desirable to generate practice problems automatically. Currently, there are routines for automatically generating problems in the following areas:

- Propositional logic.
- Boolean Set Theory.
- Induction (Divisibility).
- Induction (Inequalities).

In order to increase the available training material, it is desirable to generate practice problems automatically. Currently, there are routines for automatically generating problems in the following areas:

- Propositional logic.
- Boolean Set Theory.
- Induction (Divisbility).
- Induction (Inequalities).

Problem Generators

In order to increase the available training material, it is desirable to generate practice problems automatically. Currently, there are routines for automatically generating problems in the following areas:

- Propositional logic.
- Boolean Set Theory.
- Induction (Divisbility).
- Induction (Inequalities).

Problem Generators

In order to increase the available training material, it is desirable to generate practice problems automatically. Currently, there are routines for automatically generating problems in the following areas:

- Propositional logic.
- Boolean Set Theory.
- Induction (Divisbility).
- Induction (Inequalities).

Problem Generators

In order to increase the available training material, it is desirable to generate practice problems automatically. Currently, there are routines for automatically generating problems in the following areas:

- Propositional logic.
- Boolean Set Theory.
- Induction (Divisbility).
- Induction (Inequalities).

‘Playing fields’

In spite of the logical strive for unification:

Mathematical work takes place in a number of contexts, each of which has its own...

- Specific types (natural numbers, lines, points, angles, areas, matrices, vectors, sequences, functions...)
- Notational conventions (I_n, id, α, e, 0, 1,...)
- Elementary statements (unique prime factorization, sum of angles in a triangle, ...)
- Methods and deductions (Show $A = B$ via $A \subseteq B$ and $B \subseteq A$; show “f is bijective” via “f is injective” and “f ist surjective”)

A sensible modelling of mathematical practice, even at the beginner level, needs to take care of this. . For this, Diproche uses ‘playing fields’ (“Spielwiesen”).

'Playing fields'

In spite of the logical strive for unification:
Mathematical work takes place in a number of contexts, each of which has its own...

- Specific types (natural numbers, lines, points, angles, areas, matrices, vectors, sequences, functions...)
- Notational conventions (I_n, id, α, e, 0, 1,...)
- Elementary statements (unique prime factorization, sum of angles in a triangle, ...)
- Methods and deductions (Show $A = B$ via $A \subseteq B$ and $B \subseteq A$; show "f is bijective" via "f is injective" and "f ist surjective")

A sensible modelling of mathematical practice, even at the beginner level, needs to take care of this. . For this, Diproche uses 'playing fields' ("Spielwiesen").

‘Playing fields’

In spite of the logical strive for unification:
Mathematical work takes place in a number of contexts, each of which has its own...

- Specific types (natural numbers, lines, points, angles, areas, matrices, vectors, sequences, functions...)
- Notational conventions (I_n, id, α, e, 0, 1,...)
- Elementary statements (unique prime factorization, sum of angles in a triangle, ...)
- Methods and deductions (Show $A = B$ via $A \subseteq B$ and $B \subseteq A$; show “f is bijective” via “f is injective” and “f ist surjective”)

A sensible modelling of mathematical practice, even at the beginner level, needs to take care of this. . For this, Diproche uses ‘playing fields’ (“Spielwiesen”).

'Playing fields'

In spite of the logical strive for unification:
Mathematical work takes place in a number of contexts, each of which has its own...

- Specific types (natural numbers, lines, points, angles, areas, matrices, vectors, sequences, functions...)
- Notational conventions (I_n, id, α, e, 0, 1,...)
- Elementary statements (unique prime factorization, sum of angles in a triangle, ...)
- Methods and deductions (Show $A = B$ via $A \subseteq B$ and $B \subseteq A$; show "f is bijective" via "f is injective" and "f ist surjective")

A sensible modelling of mathematical practice, even at the beginner level, needs to take care of this. . For this, Diproche uses 'playing fields' ("Spielwiesen").

‘Playing fields’

In spite of the logical strive for unification:
Mathematical work takes place in a number of contexts, each of which has its own...

- Specific types (natural numbers, lines, points, angles, areas, matrices, vectors, sequences, functions...)
- Notational conventions (I_n, id, α, e, 0, 1,...)
- Elementary statements (unique prime factorization, sum of angles in a triangle, ...)
- Methods and deductions (Show $A = B$ via $A \subseteq B$ and $B \subseteq A$; show “f is bijective” via “f is injective” and “f ist surjective”)

A sensible modelling of mathematical practice, even at the beginner level, needs to take care of this. . For this, Diproche uses ‘playing fields’ (“Spielwiesen”).

'Playing fields'

In spite of the logical strive for unification:
Mathematical work takes place in a number of contexts, each of which has its own...

- Specific types (natural numbers, lines, points, angles, areas, matrices, vectors, sequences, functions...)
- Notational conventions (I_n, id, α, e, 0, 1,...)
- Elementary statements (unique prime factorization, sum of angles in a triangle, ...)
- Methods and deductions (Show $A = B$ via $A \subseteq B$ and $B \subseteq A$; show "f is bijective" via "f is injective" and "f ist surjective")

A sensible modelling of mathematical practice, even at the beginner level, needs to take care of this. . For this, Diproche uses 'playing fields' ("Spielwiesen").

'Playing fields'

In spite of the logical strive for unification:
Mathematical work takes place in a number of contexts, each of which has its own...

- Specific types (natural numbers, lines, points, angles, areas, matrices, vectors, sequences, functions...)
- Notational conventions (I_n, id, α, e, 0, 1,...)
- Elementary statements (unique prime factorization, sum of angles in a triangle, ...)
- Methods and deductions (Show $A = B$ via $A \subseteq B$ and $B \subseteq A$; show "f is bijective" via "f is injective" and "f ist surjective")

A sensible modelling of mathematical practice, even at the beginner level, needs to take care of this. . For this, Diproche uses 'playing fields' ("Spielwiesen").

‘Playing fields’

In spite of the logical strive for unification:
Mathematical work takes place in a number of contexts, each of which has its own...

- Specific types (natural numbers, lines, points, angles, areas, matrices, vectors, sequences, functions...)
- Notational conventions (I_n, id, α, e, 0, 1,...)
- Elementary statements (unique prime factorization, sum of angles in a triangle, ...)
- Methods and deductions (Show $A = B$ via $A \subseteq B$ and $B \subseteq A$; show “f is bijective” via “f is injective” and “f ist surjective”)

A sensible modelling of mathematical practice, even at the beginner level, needs to take care of this. . For this, Diproche uses ‘playing fields’ (“Spielwiesen”).

Subprojects

- "Playing fields": Boolean set theory, functions and relations, real numbers, elementary number theory, group theory, axiomatic geometry...
- Refine the hints
- Extend the problem generators
- Systematical foundations of the Anti-ATP, using e.g. empirical studies from didactics about common fallacies or classifications of fallacies from argumentation theory
- The trouble with class terms: How to safely deal with those without introducing beginner' students to formal set theory?

Subprojects

- "Playing fields": Boolean set theory, functions and relations, real numbers, elementary number theory, group theory, axiomatic geometry...
- Refine the hints
- Extend the problem generators
- Systematical foundations of the Anti-ATP, using e.g. empirical studies from didactics about common fallacies or classifications of fallacies from argumentation theory
- The trouble with class terms: How to safely deal with those without introducing beginner' students to formal set theory?

Subprojects

- "Playing fields": Boolean set theory, functions and relations, real numbers, elementary number theory, group theory, axiomatic geometry...
- Refine the hints
- Extend the problem generators
- Systematical foundations of the Anti-ATP, using e.g. empirical studies from didactics about common fallacies or classifications of fallacies from argumentation theory
- The trouble with class terms: How to safely deal with those without introducing beginner' students to formal set theory?

Subprojects

- "Playing fields": Boolean set theory, functions and relations, real numbers, elementary number theory, group theory, axiomatic geometry...
- Refine the hints
- Extend the problem generators
- Systematical foundations of the Anti-ATP, using e.g. empirical studies from didactics about common fallacies or classifications of fallacies from argumentation theory
- The trouble with class terms: How to safely deal with those without introducing beginner' students to formal set theory?

Subprojects

- "Playing fields": Boolean set theory, functions and relations, real numbers, elementary number theory, group theory, axiomatic geometry...
- Refine the hints
- Extend the problem generators
- Systematical foundations of the Anti-ATP, using e.g. empirical studies from didactics about common fallacies or classifications of fallacies from argumentation theory
- The trouble with class terms: How to safely deal with those without introducing beginner' students to formal set theory?

- Translating natural degrees of difficulty and proof methods into prover rules.
- Using, systematically testing and extending/improving
- Proof analysis (identification of unnecessary proof parts by backtracking from the goal; checking after each line whether the goal is already reachable etc.)
- Evaluating proof steps, e.g. by checking whether the shortest ATP-proof becomes shorter using a proposed intermediate step.
- Diproche-based textbook linked to the system (possible to refer to results from the book when doing exercises).

- Translating natural degrees of difficulty and proof methods into prover rules.
- Using, systematically testing and extending/improving
- Proof analysis (identification of unnecessary proof parts by backtracking from the goal; checking after each line whether the goal is already reachable etc.)
- Evaluating proof steps, e.g. by checking whether the shortest ATP-proof becomes shorter using a proposed intermediate step.
- Diproche-based textbook linked to the system (possible to refer to results from the book when doing exercises).

- Translating natural degrees of difficulty and proof methods into prover rules.
- Using, systematically testing and extending/improving
- Proof analysis (identification of unnecessary proof parts by backtracking from the goal; checking after each line whether the goal is already reachable etc.)
- Evaluating proof steps, e.g. by checking whether the shortest ATP-proof becomes shorter using a proposed intermediate step.
- Diproche-based textbook linked to the system (possible to refer to results from the book when doing exercises).

- Translating natural degrees of difficulty and proof methods into prover rules.
- Using, systematically testing and extending/improving
- Proof analysis (identification of unnecessary proof parts by backtracking from the goal; checking after each line whether the goal is already reachable etc.)
- Evaluating proof steps, e.g. by checking whether the shortest ATP-proof becomes shorter using a proposed intermediate step.
- Diproche-based textbook linked to the system (possible to refer to results from the book when doing exercises).

- Translating natural degrees of difficulty and proof methods into prover rules.
- Using, systematically testing and extending/improving
- Proof analysis (identification of unnecessary proof parts by backtracking from the goal; checking after each line whether the goal is already reachable etc.)
- Evaluating proof steps, e.g. by checking whether the shortest ATP-proof becomes shorter using a proposed intermediate step.
- Diproche-based textbook linked to the system (possible to refer to results from the book when doing exercises).

Thank you for your attention!

Bibliography

- J. Azzouni. The derivation-indicator view of mathematical practice
- M. Carl, M. Cramer and D. Kühlwein: Chapter 1 of Landau in Naproche, the first chapter of our Landau translation.
- M. Carl, P. Koepke: Interpreting Naproche – An algorithmic approach to the derivation-indicator view,
- M. Cramer: PhD thesis, Proof-checking mathematical texts in controlled natural language
- M. Cramer. Implicit dynamic function introduction and its connections to the foundations of mathematics
- M. Cramer, B. Schröder. Interpreting Plurals in the Naproche CNL
- M. Jamnik. On Automating Diagrammatic Proofs of Arithmetic Arguments
- R. Nelsen. Proofs Without Words. Exercises in Visual Thinking.
- Y. Rav. A Critique of a formalist-mechanist justification of arguments in mathematician's proof practice