Inference operators

A. Zelmer

Abstract

Certainly the beauty of mathematics also lies in the precision of mathematical sentences, as well as in the proofs of theorems. However, this can only be fully achieved by formalizing sentences and proofs.

Our computer-assisted proving system, which we have called ”Logic”, contains two fundamental modules: the sentence editor and the proof module. We presented the sentence editor in Mathematical Sentences as Matrices. The proof module of the "Logic" system is based on certain primitive functions that we have called inference operators. In this paper we present what inference operators are and how they can be used in formalized mathematical proofs.

Mathematical proofs

A proof of a theorem contains a numbered list of steps. Each step represents a sentence. Initially this list is empty. We can generate a proof step in two cases:

- In the first case, we can declare a sentence (axiom, definition, or theorem) as a premise of the proof.
- In the second case, we can call an inference operator for sentences already existing in the proof. If the inference operator can be applied, then its result is the sentence of the new step of the proof.

A proof is complete if the sentence of the last step coincides with the theorem we want to prove, or, in the case of a proof by reductio ad absurdum, it is a contradiction.

To explain how inference operators work, we will need some example sentences. For this purpose we will use the matrix form or the text form of the sentences. The following table shows the notation conventions for logical operators, which are the nodes of our sentences.

Logical operators

Operator	Designation	Description	Arg-Nr.	Arg-Kind	Res-Kind
A	$\forall$	Universal quantifier	2	Form	Form
E	$\exists$	Existential quantifier	2	Form	Form
K	{..}	Class generator	1	Form	Term
X	$\wedge$	Conjunction	2	Form	Form
V	$\vee$	Disjunction	2	Form	Form
D	$\leftrightarrow$	Double implication	2	Form	Form
C	$\rightarrow$	Implication	2	Form	Form
N	Not	Negation	1	Form	Form
T	-	True	0	-	Form
^	-	Link to A, E or K	0	-	Term
@	$\in$	Belonging, Membership	2	Term	Form
=	=	Equality	2	Term	Form
S	Set	Class is a set	1	Term	Form
User def.	User def.	New relation	User def.	Term	Form
User def.	User def.	New function	User def.	Term	Term

Arg-Nr. = number of arguments of the logical operator

Arg-Kind = kind of the arguments of the logical operator

Res-Kind = kind of the result of the logical operator

We have extended the notion of "logical operator" to include operators that are terms.

As we can see, we use binary quantifiers and call them (conditional/restricted) quantifiers. This allows us to generate sentences closer to the usual mathematical formulations.

The operators "K", "^ ", "@ ", "= ", "S" are necessary for a consistent implementation of the requirements for sentences and proofs.

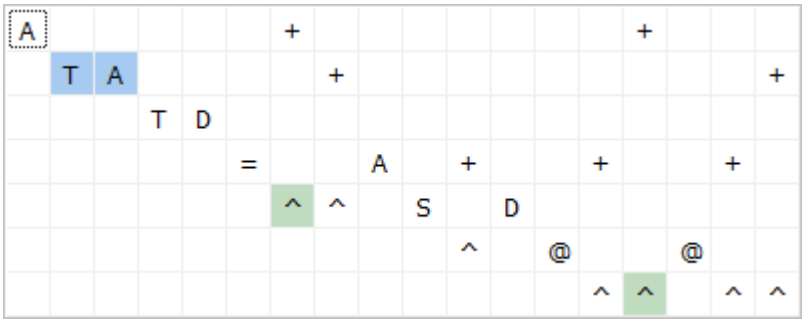
In this paper we present an example of a mathematical sentence along with the formalizations used in the description of inference operators.

Axiom of extensionality

*Let **X**, **Y** be classes.*
***X** = **Y** if and only if, for any set **u**, **u** belongs to **X** if and only if **u** belongs to **Y**.*

```
A(X) A(Y) D(=(X,Y),A(u|S(u)) D(@(u,X),@(u,Y)))
```

The matrix form of the sentence:



The text form of the sentence:

```
A - - - - - + - - - - - - - - + - - -
  T A - - - - + - - - - - - - - - - +
      T D
          =     A - + - - + - - + -
            ^ ^   S   D
                  ^   @     @
                        ^ ^   ^ ^
```

In the matrix form of a sentence, each logical operator occupies a single column of the matrix. By the **position** of a logical operator in a sentence we will understand the column in which it is located.

To explain the use of inference operators, we need to define some notions related to the matrix structure of sentences.

Prefix

A list of quantifiers of a sentence is called **pre-q-list** (prefix-quantifier-list) if the following conditions are satisfied:

- The first quantifier from the pre-q-list is the root of the sentence.
- Each quantifier from the pre-q-list, except the first one, is the second argument of the previous quantifier from the pre-q-list.
- Each quantifier from the pre-q-list, except the last one, has as its second argument the next quantifier from the pre-q-list.

We call **prefix-arg** the second argument of the last quantifier in the pre-q-list.

The **prefix** of the sentence is the part of the sentence from the root to (exclusively) the prefix-arg.

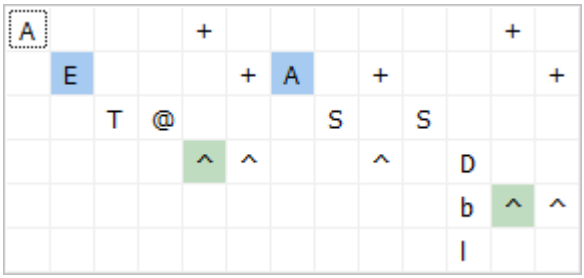
If in the definition of the pre-q-list we refer only to universal quantifiers, we obtain the notions of **u-pre-q-list**, **u-prefix-arg**, and **u-prefix**.

If the prefix and the u-prefix of a sentence are equal, then we say that the sentence has a universal prefix. If the prefix/u-prefix of the sentence is empty, then the prefix-arg/u-prefix-arg is the root of the sentence (the first logical operator of the sentence).

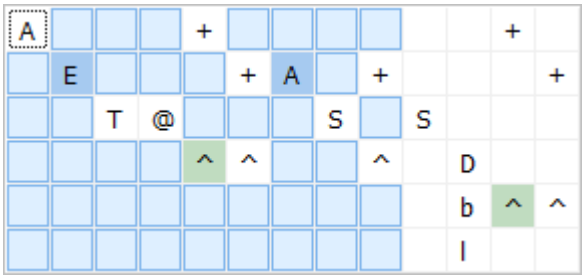
Example

Let **p** be a quantifier from the pre-q-list and **q** a quantifier of the sentence. If **q** is the first argument of **p**, or if the first argument of **p** is an ancestor of **q**, then **q** belongs to the prefix of the sentence but not to the pre-q-list.

Let us consider the following example:



In this example, the pre-q-list consists of the quantifiers at positions 1 and 7. The columns of the sentence from the first to the ninth column form the prefix. To highlight it, we have marked its columns light blue:



The prefix-arg is the logical operator "S" at position 10. The quantifier at position 2 is the first argument of the quantifier at position 1, so it is part of the prefix but not of the pre-q-list.

Relationships between the logical operators of a sentence

*As we have seen, a **sentence** is a tree having logical operators as nodes, starting from a form and including a graphical structure to represent links.*

*Each logical operator is the **parent** of its arguments, and each logical operator except the **root** (the first logical operator of the sentence) has a parent. Each logical operator except the root has **ancestors** (the parent, the parent of the parent, …). If **p** is an ancestor of **q**, then **q** is a **descendant** of **p**.*

*Two logical operators of a sentence, which are not equal to the root, have a **first common** ancestor.*

*Let **p** be a logical operator of a sentence. By the **subtree** of **p** we mean **p** and all its descendants. We denote the subtree of **p** by Sub(**p**). The root's subtree coincides with the entire sentence.*

*Let **p**, **q**, **x** be logical operators in a sentence. We say that **x** is **between p** and **q** if **p** is an ancestor of **x** and **x** is an ancestor of **q**.*

*We say that **p** is a conjunction ancestor of **q** if **p** is a conjunction, **p** is an ancestor of **q**, and all the logical operators between **p** and **q** are conjunctions.*

*A **link** is a reference from an argument of a logical operator to a quantifier or class generator. The quantifier or the class generator must be an ancestor of the logical operator.*

Inference operators

Inference operator	Step-Nr.	Pos-Nr.
Attachment	1	2
Class generator to form	1	1
Commutativity	1	1
Cross attachment	2	1
Cross replacement	2	1
Deleting a quantifier	1	1
Deleting quantifiers from right to left	1	1
Dual quantifier distributivity	1	1
Duplication (conjunction)	1	2
Duplication (implication)	1	2
Existential generalization (using equality)	1	2
Extracting the condition of the quantifier	1	1
Extracting the hypothesis	0	0
Form to class generator	1	1
Inserting a sentence into another sentence	2	1
Inserting the condition into the quantifier	1	1
Interchanging quantifiers	1	1
Negation	1	1
Partial quantifier distributivity	1	1
Particularization	1	2
Quantifier distributivity	1	1
Reductio ad absurdum	0	0
Reflexivity	1	1
Replacement	1	2
Restricted quantifier distributivity	1	1
Tautologize	0	0
Universal generalization (using equality)	1	2

Step-Nr. = number of arguments representing steps of the proof

Pos-Nr. = number of arguments representing positions of the last referenced step

In our system, a proof of a theorem follows exactly the idea of the traditional proof, but it is complete; that is, it mentions absolutely all the references to other statements and cannot be wrong.

In order to be used by inference operators, a sentence must be declared as a premise of the proof we are working on. Thus, the desired sentence becomes a step of the demonstration and can be used in calls of inference operators by indicating its step number.

Before calling an inference operator, we must do the following:

- Decide which inference operator we want to use.
- Select from the proof the necessary steps as arguments for the inference operator.
- Select from the sentence of the last selected step the necessary positions of logical operators as arguments for the inference operator.

The hypothesis of the sentence we want to prove can be used in the demonstration. By calling the inference operator "**Extracting the hypothesis**" or the inference operator "**Tautologize**", we generate a step of the proof containing the hypothesis. If we want to make a demonstration by reductio ad absurdum, the first inference operator we need to call in the proof is "**Reductio ad absurdum**". It generates a step of the proof containing the negation of the sentence we want to prove.

A special category of inference operators consists of operators that rely exclusively on tautologies from propositional logic, having as their root an implication or a double implication (“C” or ”D”). We will call these operators "Formulas", but be aware that they are inference operators in the sense we have described. For propositional logic formulas we will use prefix notation, for example, “CXpCpqq”.

An important group of inference operators processes the various properties of logical quantifiers: “**Quantifier distributivity**”, ”**Restricted quantifier distributivity**”, ”**Partial quantifier distributivity**”, ”**Interchanging quantifiers**”.

The quantifiers we use are conditional quantifiers. To process them, we need the inference operators “**Extracting the condition of the quantifier**” and ”**Inserting the condition into the quantifier**”.

In informal proofs, the mathematician often extracts parts from the sentences he wants to process and then applies various logical inferences to those parts. In most cases, he does not ask himself whether these manipulations are allowed, and only his intuition helps him to reach correct results.

We believe that processing a sentence is only allowed in situ, that is, in the complete context of the sentence. To achieve this, we have defined several inference operators specialized in copying parts of a sentence from one position to another:

Attachment
Duplication (conjunction)
Duplication (implication)
Replacement

In informal proofs, the magic formula "from theorem … it follows that..." is often used. Thus, the author wants to tell us that he has adapted the theorem to the conditions of the working sentence and has used this adaptation in it. We do not know the details of this "inference" and we have no possibility to verify its correctness. The inference operators "**Cross attachment**" and "**Cross replacement**" solve this problem elegantly.

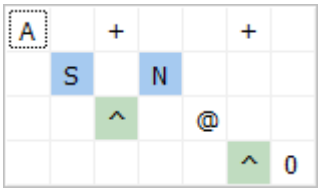
An example of a proof

Let's assume that we have already proven Theorem 15 and created Definition 11. With their help we want to prove the following theorem:

Theorem

No set belongs to 0.

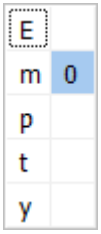
`A(x|S(x)) N(@(x,0))`



Proof

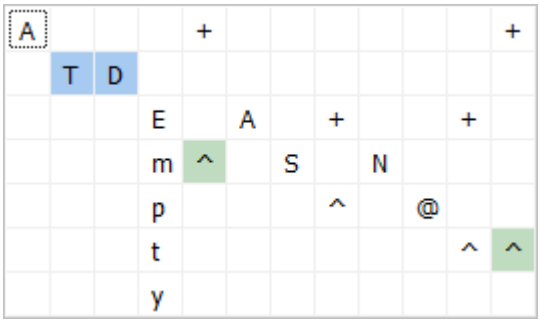
We declare theorem 15 as a premise of the proof:

Step 1.



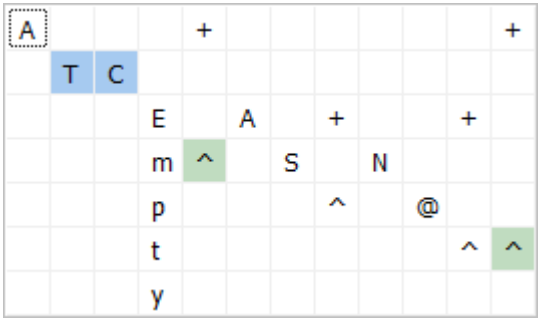
We declare definition 11 as a premise of the proof:

Step 2.



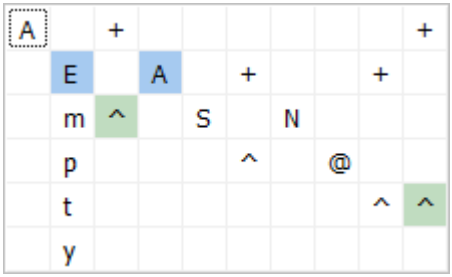
In the next step we want to transform the double implication from the third position of the second step into an implication. For this, we will use the formula "**Double implication to implication**" (CDpqCpq) for step 2 and position 3:

Step 3.



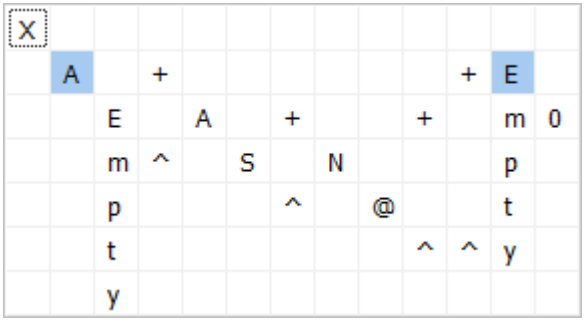
In step 3, the quantifier at position 1 has the condition "T"; that is, it has no condition. Its second argument, which is at position 3, is an implication. By applying the inference operator "**Inserting the condition into the quantifier**" for step 3 and position 1, we transfer the first argument of this implication to the condition of the quantifier.

Step 4.



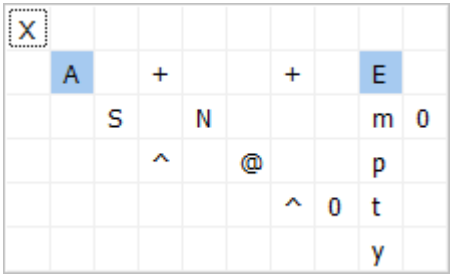
We now want to insert the sentence from step 1 into the sentence from step 4 at position 1. To do this, we call the inference operator "**Inserting a sentence into another sentence**" for steps 1 and 4 and position 1:

Step 5.



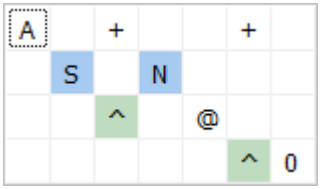
In step 5 at position 13 there is the empty set (0), which satisfies the "Empty" condition. We can therefore particularize the universal quantifier at position 2 using the empty set. For this, we apply the inference operator "**Particularization**" for step 5 and positions 13 and 2:

Step 6.



The sentence in step 6 is a conjunction. Applying the formula "**Subtraction (left)**" (CXpqp) for step 6 and position 1, we obtain the following result:

Step 7.



The theorem is proven because the sentence in step 7 coincides with the theorem we want to prove.

How inference operators work

In the description of the inference operators, we use the principle of duplicability and the principle of implicability.

Duplicability

Let ***p*** and ***q*** be different forms of a sentence and ***o*** their first common ancestor. We say that ***p*** is duplicable on ***q*** (***q*** is in the influence zone of ***p***) if one of the following conditions is met:

- ***o*** is “A” or ”C”
  - The first argument of ***o*** is ***p,*** or a conjunction ancestor of **p**.
  - The second argument of ***o*** *is* ***q****, or an ancestor of* ***q***.
- ***o*** is “E” or ”X”
  - The first argument of ***o*** is ***p,*** or a conjunction ancestor of ***p***.
  - The second argument of ***o*** *is* ***q****, or an ancestor of* ***q***.
- ***o*** is “E” or ”X”
  - The first argument of ***o*** is ***q,*** or an ancestor of ***q***.
  - The second argument of ***o*** is ***p,*** or a conj. ancestor of ***p.***

If ***p*** *is duplicable on* ***q****, then, by replacing Sub(****q****) by the conjunction of Sub(****p****) and Sub(****q****) or by the implication of Sub(****p****) and Sub(****q****) in our sentence, we obtain an equivalent sentence.*

Implicability

We say that a form of a sentence is implicable if it verifies one of the following conditions:

- The form is the root of the sentence (the first logical operator).
- The form is the second argument of an implicable ”A” or ”C”.
- The form is one of the arguments of an implicable ”E”, ”X”, or ”V”.

If a conjunction of a sentence is implicable, then, the replacing of the subtree of this conjunction with the subtree of one of its arguments is a logical inference.

Implicability is a dynamic property of the logical operators of a sentence. If an operator is implicable, it means that it is "active" for logical inferences. Implicability should not be confused with Gottlob Frege's theory of polarity nor with van Dalen's positive/negative subsentences ($Sub^{+}(\varphi)/Sub^{-}(\varphi)$*), which are more static properties.*

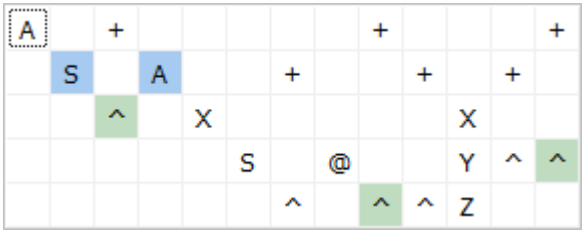
In the following, when describing inference operators, we will use this simplified expression:

Suppose we have selected a step of a proof. Instead of saying that we select a logical operator from the result of the selected step, we will briefly say that we select a logical operator from the selected step.

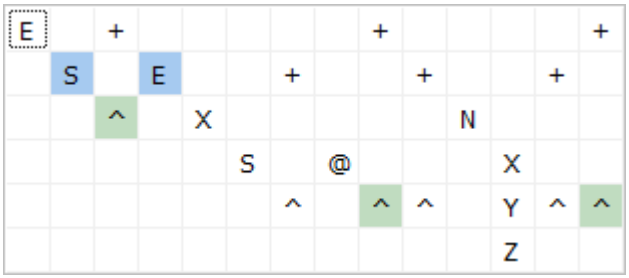
The purpose of this paper is to present the principles of using inference operators. For this reason, we will present the more important inference operators.

Reductio ad absurdum

This inference operator generates a step of the proof with the result equal to the negation of the sentence we want to prove. If the sentence we want to prove is:



The result of the inference operator is:



Negation

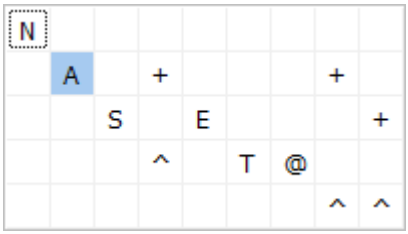
To use this inference operator, select:

- a step of the proof, and
- a negation from the selected step.

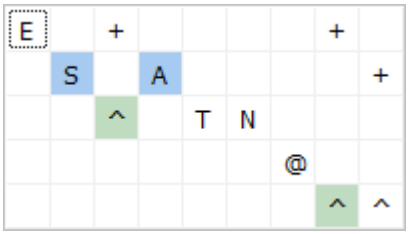
The operator uses the logical rules of negation.

Like for the logical operator "N", for the inference operator "Negation", we will use the word "not" and not the word "non".

Example



If we apply “Negation” to the root of the sentence, we obtain:



*The action of some inference operators is explained by a logical **transformation formula**. Let us consider the formula:*
AcCpq → CEcpq
This means that the selected form has the structure AcCpq, which in the result is replaced by the structure CEcpq.

```
a = universal quantifier (“A”)
c = condition part of the quantifier
i = implication (“C”)
p = the first argument of the implication
q = the second argument of the implication
e = existential quantifier (“E”)
```

In simplified representations of sentences, by repeating a letter we will understand a logical operator together with its subtree.

Selected step:

```
- - - A c c c C p p p q q q q - - -

      a c     i p     q
```

The result of the inference operator is:

- - - C E c c c p p p q q q q - - -

i e c p q

Class generator to form

To use this inference operator, select:

- a step of the proof, and
- a form from this step.

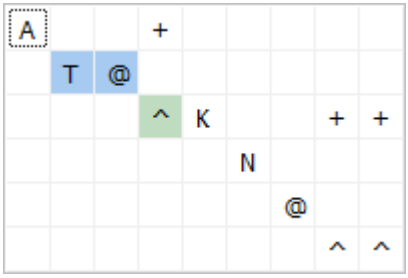
This operator verifies whether:

- The selected form is a belonging (“@”).
- The second argument of this belonging is a class generator (“K”).

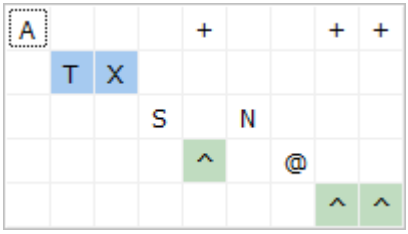
This inference operator transforms a belonging to a class generator into a form without the class generator. The action of this inference operator is based on the following formula, where **x** and **y** are variables, and p(**y**) is a predicate of variable **y**:

$x \in \{y \mid p(y)\} \leftrightarrow (S(x) \wedge p(x))$

Let us suppose that we have the following step in the proof of a theorem:



At position 3 we have a belonging to a class generator. With the inference operator “Class generator to form”, we obtain:



As we can see, in position 3, we now have the explanation of belonging to a class generator.

Form to class generator

To use this inference operator, select:

- a step of the proof, and
- a form from this step.

This operator verifies whether:

- The selected form is a conjunction.
- The first argument of the conjunction is a set operator.

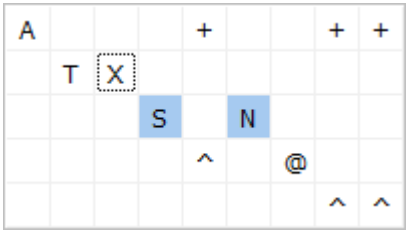
This inference operator transforms a form that satisfies the conditions above into a belonging to a class generator.

The action of this inference operator is based on the following formula, where **x** and **y** are variables, and p(**y**) is a predicate of variable **y**:

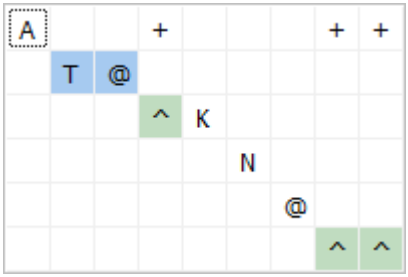
$(S(x) \wedge p(x)) \leftrightarrow x \in \{y \mid p(y)\}$

The inference operator “Form to class generator” is exactly the reverse of the inference operator “Class generator to form”.

Example



By using “Form to class generator” we obtain:



Attachment

To use this inference operator, select:

- a step of the proof, and
- two forms from the selected step (the source and the destination).

This operator verifies whether:

- The source is the first argument of an implication.
- The parent of the source is duplicable on the destination.
- The subtree of the source and the subtree of the destination are equal.

This inference operator works in two modes: removing/attaching. To present schematically the action of the inference operator, we use the abbreviations below:

```
s = source
sp = parent of the source
sp2 = the second argument of sp
d = destination
dp = parent of the destination
dp1 = the first argument of dp
dp2 = the second argument of dp
```

If the selected step has one of the structures below (removing):

Sub(**sp2**) is equal to Sub(**dp2**)

```
- - - C p p p q q q q - - - X p p p q q q q - - -

      s s     s             d d     d
      p       p             p       p
              2                     2
```

Sub(**sp2**) is equal to Sub(**dp1**)

```
- - - C p p p q q q - - - X q q q q p p p - - -

      s s     s             d d       d
      p       p             p p
              2               1
```

then the result of the inference operator is:

```
- - - C p p p q q q - - - p p p - - -

      s s     s
      p       p
              2
```

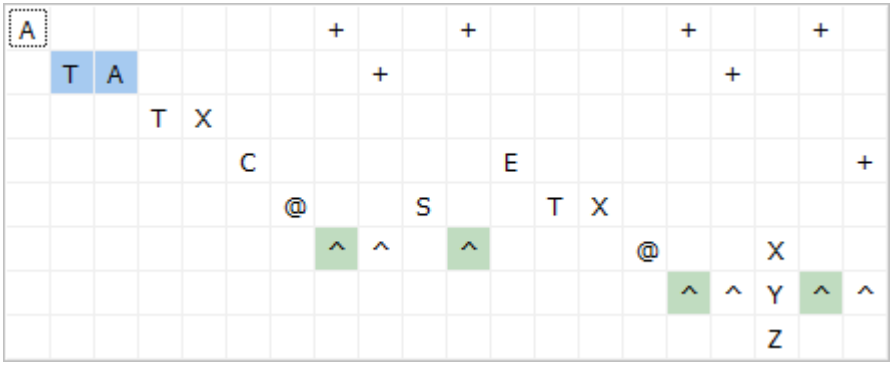
Otherwise, the selected step has the structure (attaching):

```
- - - C p p p q q q - - - p p p - - -

      s s     s             d
      p       p
              2
```

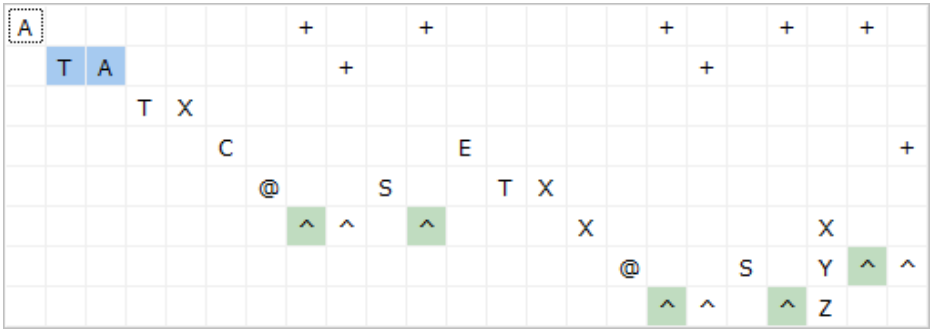
and the result of the inference operator is:

```
- - - C p p p q q q - - - X p p p q q q q - - -

      s s     s
      p       p
              2
```

Let's assume that the following example is a proof step. We notice that in position 7, we have the same belonging as in position 15, and the parent of the belonging in position 7 is an implication:



If we select the position 7 and then the position 15 of the sentence, followed by a call of the inference operator "Attachment", we obtain a new step of the proof:



If we apply the "Attachment" operator to this result for positions 7 and 16, we get the initial sentence from our example.

Replacement

To use this inference operator, select:

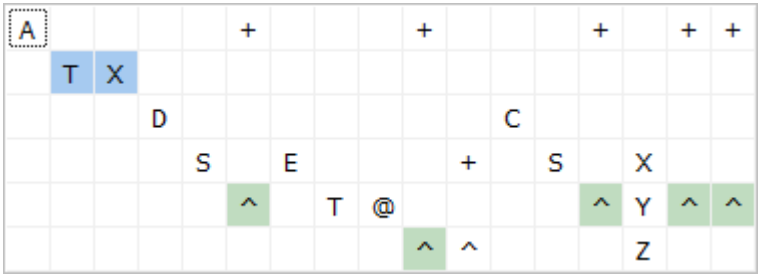
- a step of the proof, and
- two logical operators of the step (the source and the destination).

This operator verifies whether:

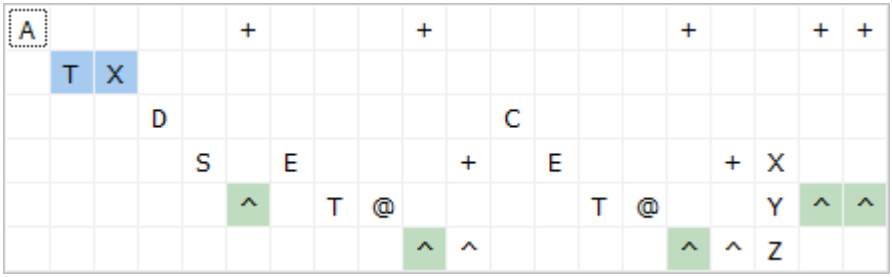
- The parent of the source is a double implication, an equality, or an implication. In the case of the implication, the source must be its first argument.
- The subtree of the source and the subtree of the destination are equal.
- If the parent of the source is a double implication or an implication, then the destination must be a form, and the parent of the source must be duplicable on the destination.
- If the parent of the source is an implication, then the destination must be implicable.
- If the parent of the source is an equality, then the destination must be a term, and the parent of the source must be duplicable on the first ancestor form of the destination.

The function replaces the subtree of the destination with the subtree of the other argument of the parent of the source.

Let us consider that our proof contains the following sentence:



If we select the logical operator at position 5 and then the logical operator at position 13, by applying the inference operator “Replacement” we obtain:



Duplication (conjunction)

To use this inference operator, select:

- a step of the proof, and
- two forms from the selected step (the source and the destination).

This operator verifies whether:

- The source and the destination are at different positions.
- The source is duplicable on the destination.

Let **s** be the source and **d** the destination.

If Sub(**s**) and Sub(**d**) are different, the operator replaces Sub(**d**) with X(Sub(**s**), Sub(**d**)).

If Sub(**s**) and Sub(**d**) are equal and the parent of **d** is a conjunction, the operator replaces the subtree of this conjunction with the subtree of the other argument of the conjunction.

As you can see, this inference operator works in two modes: duplicating/removing.

If the selected step has the structures below (duplicating):

```
- - - p p p - - - q q q q - - -
      s           d
```

the result of the inference operator is:

```
- - - p p p - - - X p p p q q q q - - -
```

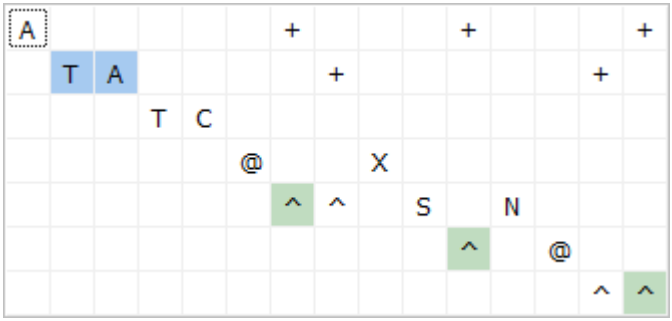
If the selected step has one of the following structures (removing):

```
- - - p p p - - - X p p p q q q q - - -
      s             d
- - - p p p - - - X q q q q p p p - - -
      s                       d
```

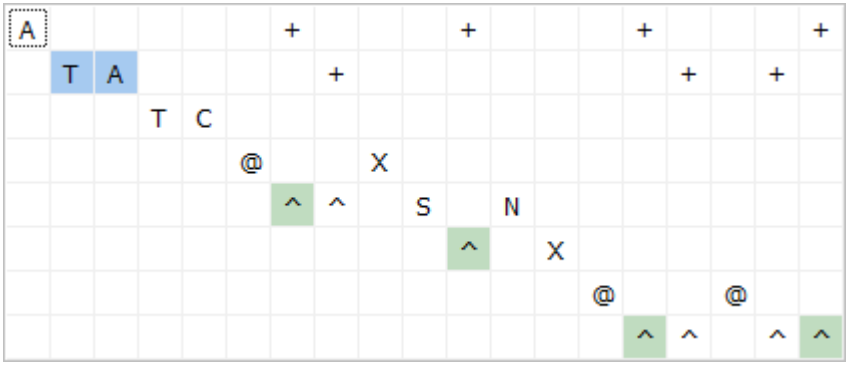
the result of the inference operator is:

```
- - - p p p - - - q q q q - - -
```

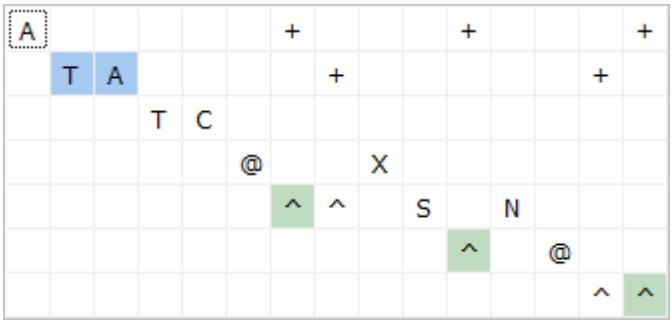
Example



We have applied “Duplication (conjunction)” for the source at position 6 and the destination at position 13:



Now we apply the same operator for the source at position 6 and the destination at position 14:



Duplication (implication)

To use this inference operator, select:

- a step of the proof, and
- two forms from the selected step (the source and the destination).

This operator verifies whether:

- The source and the destination are at different positions.
- The source is duplicable on the destination.

The action of this inference operator is very similar to the operator "Duplication (conjunction)".

Let **s** be the source and **d** the destination.

If Sub(**s**) and Sub(**d**) are different, the operator replaces Sub(**d**) with C(Sub(**s**), Sub(**d**)).

If Sub(**s**) and Sub(**d**) are equal and **d** is the first argument of an implication, the operator replaces the subtree of this implication with the subtree of its second argument.

As you can see, this inference operator works in two modes: duplicating/removing.

If the selected step has the structure below (duplicating):

```
- - - p p p - - - q q q q - - -
      s           d
```

the result of the inference operator is:

```
- - - p p p - - - C p p p q q q q - - -
      s             d
```

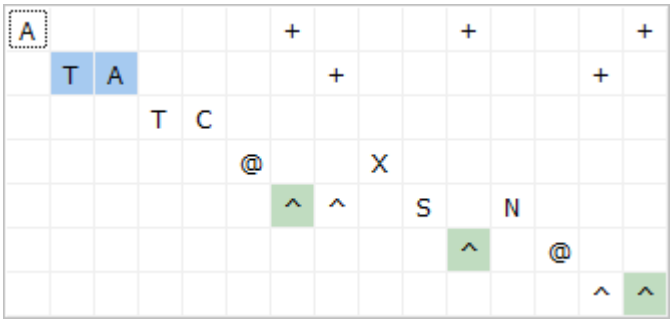
If the selected step has the following structure (removing):

```
- - - p p p - - - C p p p q q q q - - -
      s             d
```

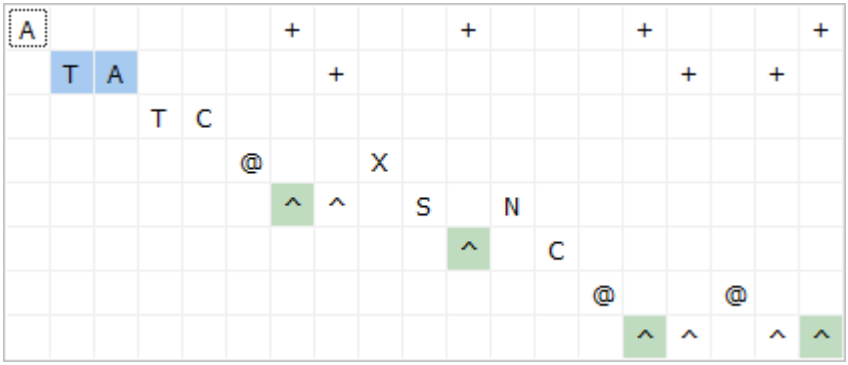
the result of the inference operator is:

```
- - - p p p - - - q q q q - - -
```

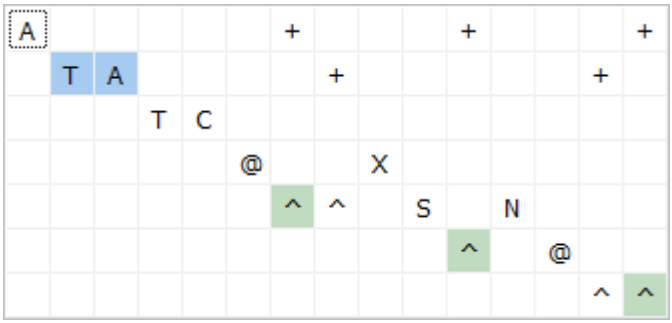
Example



We apply “Duplication (implication)” for the source at position 6 and the destination at position 13:



Now we apply the same operator for the source at position 6 and the destination at position 14:



Existential generalization, Universal generalization (using equality)

The operator is based on the formulas Q25 and Q26.

To use these inference operators, select:

- a step of the proof,
- a form from the selected step, and
- a term from the selected step.

These operators verify whether:

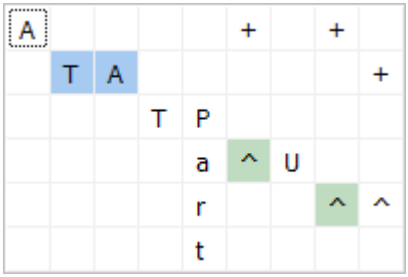
- The selected form is an ancestor of the selected term.
- The selected term has no references (links) to the selected form.

The inference operator replaces the subtree of the selected form with an existential/universal quantifier. The condition (first argument) of the new quantifier

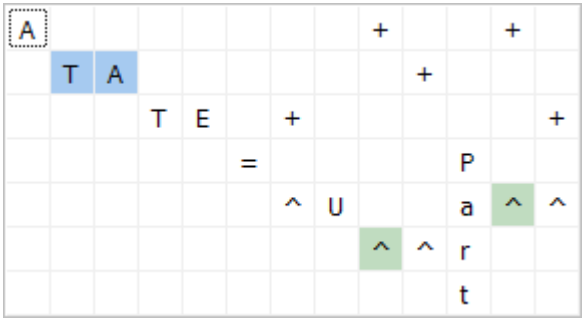
is the equality of a link to the quantifier and the subtree of the selected term. The execution (second argument) of the quantifier is the subtree of the selected form, where each occurrence of the subtree of the selected term is replaced through links to the new quantifier.

Note that the inference operator “universal generalization” does not correspond to the notion of universal generalization in predicate logic.

Example



We select the form at position 5 (“Part”) and its term at position 7 (“U”), and then we call the operator “Existential generalization”:



Extracting the condition of the quantifier

The operator is based on the formulas Q3 and Q4.

To use this inference operator, select:

- a step of the proof, and
- a form from the selected step.

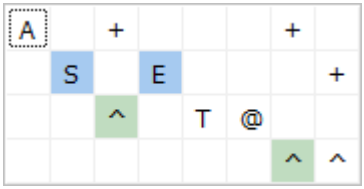
This operator verifies whether the selected form is a quantifier.

We defined the quantifiers as having two arguments: the condition and the execution part. This inference operator moves the subtree of the condition, or, if the condition is a conjunction, the subtree of the second argument of the condition, into the subtree of the execution argument of the quantifier.

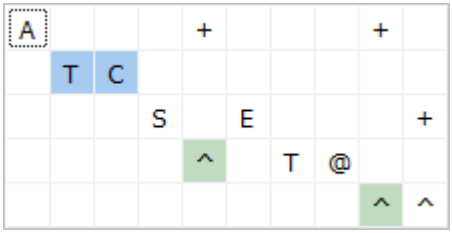
If the selected form is:

- a universal quantifier with no conjunction as the condition part, then
  `Apq` $\rightarrow$ `ATCpq`
- a universal quantifier with a conjunction as its condition part, then
  `AXpqr` $\rightarrow$ `ApCqr`
- an existential quantifier with no conjunction as the condition part, then
  `Epq` $\rightarrow$ `ETXpq`
- an existential quantifier with a conjunction as its condition part, then
  `EXpqr` $\rightarrow$ `EpXqr`

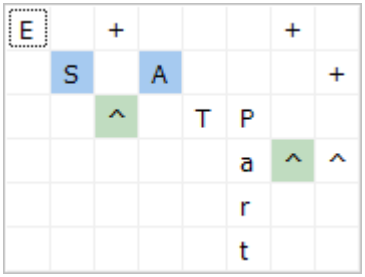
Example for a universal quantifier:



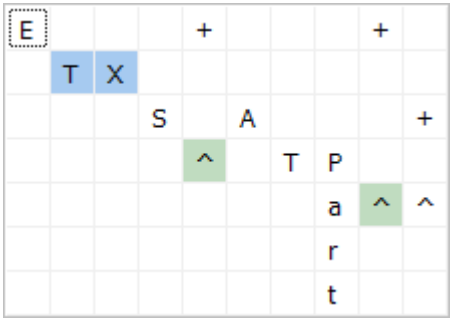
We call “Extracting the condition of the quantifier” for the first quantifier:



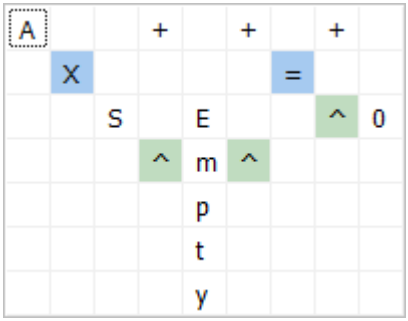
Example for an existential quantifier:



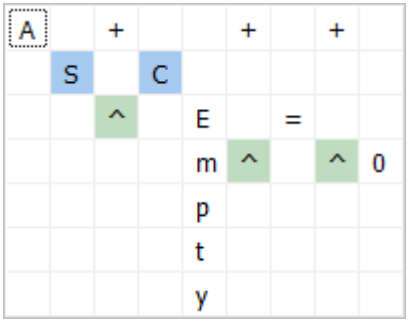
In this case the result is:



Example of a universal quantifier with a conjunction as condition:



The inference operator “Extracting the condition of the quantifier” only extracts the second argument of this conjunction:



Inserting the condition into the quantifier

The operator is based on the formulas Q3 and Q4.

To use this inference operator, select:

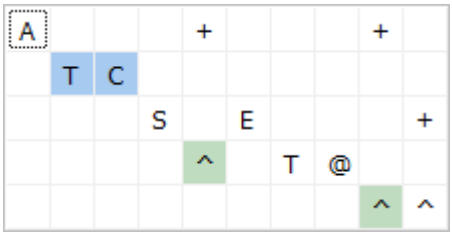
- a step of the proof, and
- a quantifier from the selected step.

If the quantifier is:

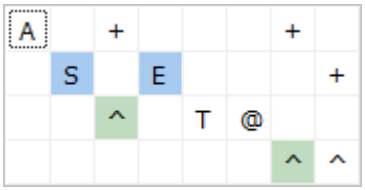
- universal, then its execution part must be an implication.
  `ATCpq` → `Apq`
  `ApCqr` → `AXpqr`
- existential, then its execution part must be a conjunction.
  `ETXpq` → `Epq`
  `EpXqr` → `EXpqr`

This inference operator is the reverse of “Extracting the condition of the quantifier”.

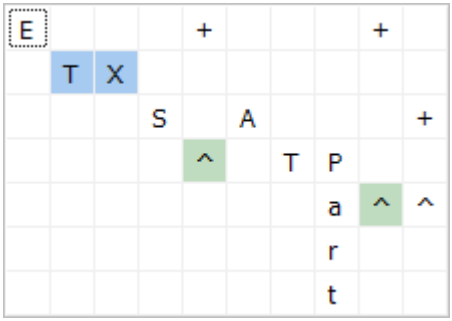
Example for a universal quantifier:



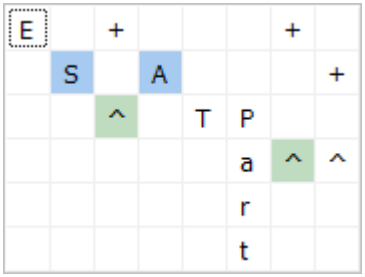
The result is:



Example for an existential quantifier:



The result of “Inserting the condition into the quantifier” for the first quantifier is:



Particularization

The operator is based on the formula Q5.

To use this inference operator, select:

- a step of the proof
- a quantifier, a link (to a quantifier), a function or a class generator (the source)
- an implicable universal quantifier (the destination).

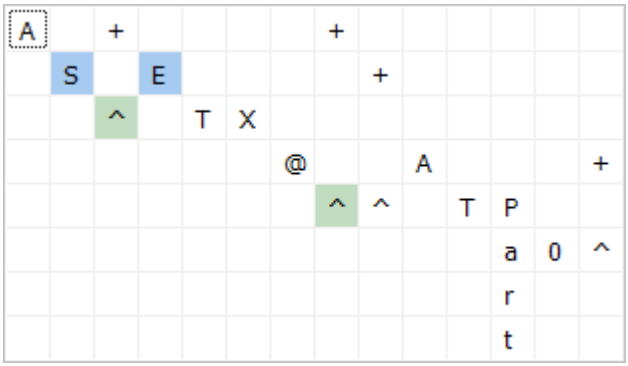
This operator verifies whether:

- If the source is a quantifier, it is an ancestor of the destination; if not, all the external references of the subtree of the source are ancestors of the destination.
- The source satisfies the condition of the destination.

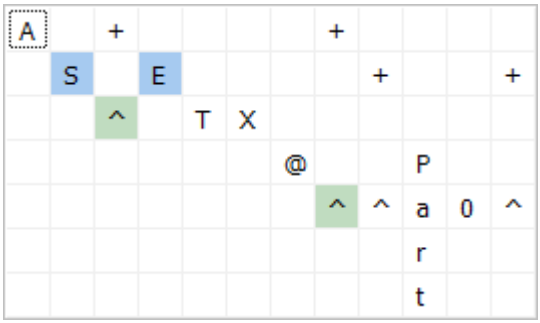
If the source is a quantifier, then the result is obtained from the selected step by replacing the subtree of the destination with the subtree of the execution part of the destination, where the links to the destination have been replaced by links to the source.

If the source is a link to a quantifier, a function, or a class generator, then the result is obtained from the selected step by replacing the subtree of the destination with the subtree of the execution part of the destination, where the links to the destination have been replaced by the subtree of the source.

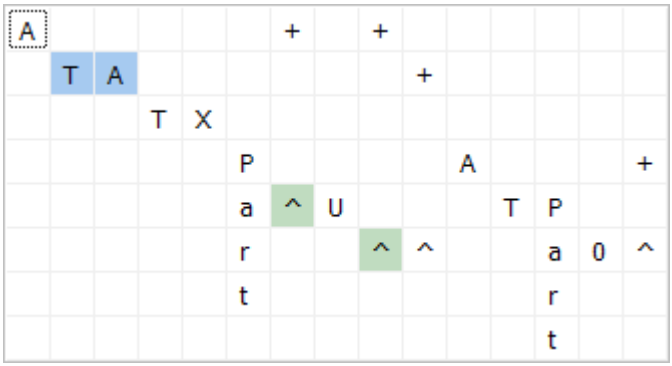
Example: The source is a quantifier.



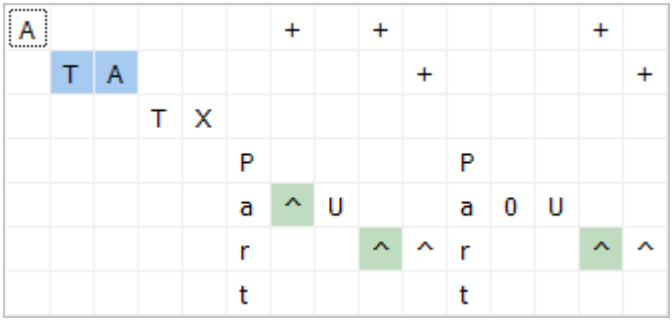
If we select the quantifier on position 4 as the source and the quantifier on position 10 as the destination, the result of the operator is:



Example: The source is a function.



If we select as source the function on position 8 and as destination the quantifier on position 11, the operator “Particularization” generates the sentence:



The functionality of “Particularization” is similar when the source is a link or a class generator.

Cross attachment

To use this inference operator, select:

- two steps of the proof (the source step and the destination step), and
- a form from the destination step (the destination).

This operator verifies whether:

- The prefix of the source step is universal.
- The prefix-arg of the source step is an implication. Let’s name the first argument of this implication the source and the second one the attachment. The subtree of the source references all the quantifiers of the pre-q-list of the source step.
- The subtree of the source is equal to the subtree of the destination.

If the parent of the destination is a conjunction, and if the subtree of the other argument of this conjunction is equal to the subtree of the attachment, then the operator replaces the subtree of the parent of the destination with the subtree of the destination. Otherwise, the operator replaces the subtree of the destination with a conjunction of the subtree of the destination and the subtree of the attachment.

As you can see, this inference operator works in two modes: removing/attaching.

The action of the operator is based on the following tautology:

```
CCpqDpXpq
C(C(p,q),D(p,X(p,q)))
(p → q) → (p ↔ (p ∧ q))
```

It is very important to understand that the inference operator "Cross attachment" compares the source with the destination by identifying their corresponding links. All other operators in these two forms must be identical.

To present schematically the action of the inference operator, we use the abbreviations below:

```
p = prefix
pa = prefix-arg
s = source = first argument of pa
a = attachment = second argument of pa
```

```
d = destination
dp = destination parent
dp1 = first argument of dp
dp2 = second argument of dp
```

Source step:

```
- - - C p p p q q q q

p     p s     a
      a
```

If the destination step has one of the structures below (removing):

```
- - - X p p p q q q q - - -

      d d     d
      p       p
              2

- - - X q q q q p p p - - -

      d d       d
      p p
        1
```

the result of the inference operator is

```
- - - p p p - - -
```

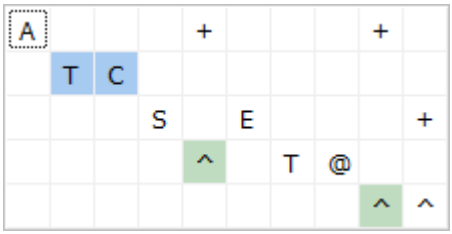
Otherwise, the destination step has the structure (attaching)

```
- - - p p p - - -

      d
```

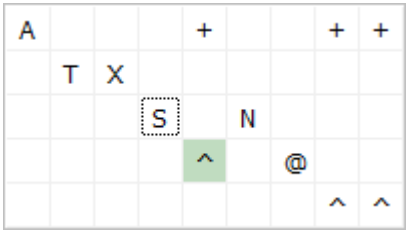
and the result of the inference operator is

```
- - - X p p p q q q q - - -
```

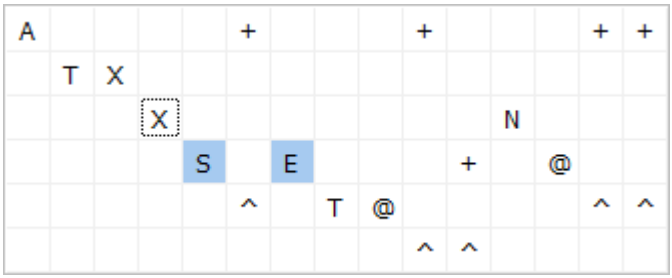
Let us suppose that the source step is:



The destination step is:



In the destination we have selected the logical operator “S” on position 4 (cell (4, 3)). The result of “Cross attachment” is:



In position 4, we have a conjunction of the destination and the attachment.

Cross replacement

To use this inference operator, select:

- two steps of the proof (the source step and the destination step) and
- a form or a term from the destination step (the destination).

This operator verifies whether:

- The prefix of the source step is universal.
- The prefix-arg of the source step is one of the following operators: double implication, implication, or equality. Let's name the first argument of this operator the source. The subtree of the source references all the quantifiers in the pre-q-list of the source step.
- If the prefix-arg of the source step is an implication, then the destination is implicable.
- The subtree of the destination is equal to the subtree of the source.

The result of the operator consists of the destination step, in which the subtree of the destination is replaced by the subtree of the second argument of the prefix-arg of the source step.

It is very important to understand that the inference operator "Cross replacement" compares the source with the destination by identifying their corresponding links. All other operators in these two forms must be identical.

To present schematically the action of the inference operator, we use the abbreviations below:

```
p = prefix
pa = prefix-arg
s = source = first argument of pa
pa2 = second argument of pa
d = destination
```

M represents one of the logical operators "**D**", "**C**", or "**=**".

Source step:

```
- - - M p p p q q q q

p     p s     p
      a       a
              2
```

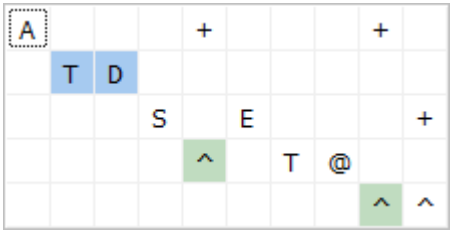
Destination step:

```
- - - p p p - - -

      d
```

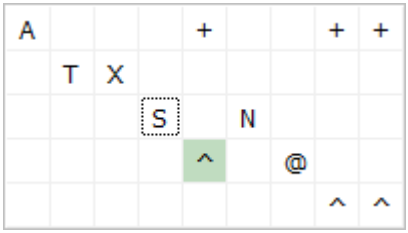
The result of the inference operator is:

```
- - - q q q q - - -
```

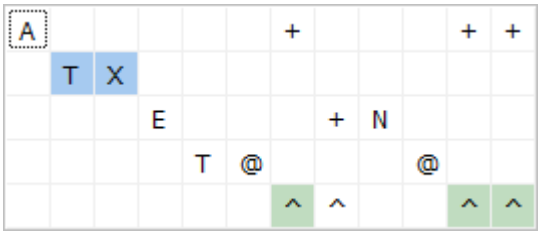
Let us suppose that the source step is the axiom of set:



The destination step is



We have selected the logical operator “S” on position 4 (cell (4, 3)). The result of the “Cross replacement” is



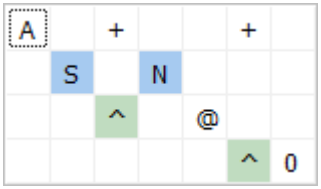
Inserting a sentence into another sentence

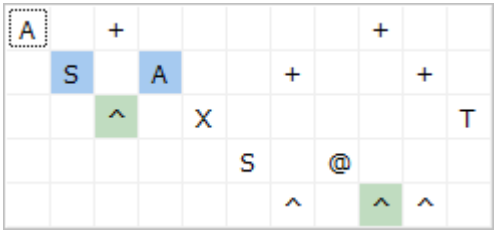
To use this inference operator, select:

- two steps of the proof (the source step and the destination step), and
- a form from the destination step (the destination).

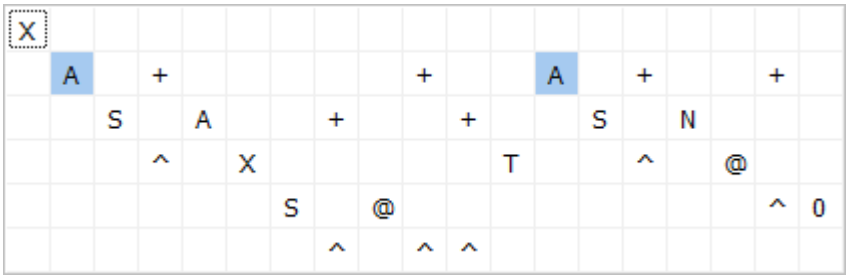
The inference operator generates a new step consisting of the destination step, where the subtree of the destination is replaced by the conjunction of the subtree of the destination and the source step.

Let us suppose we have selected two steps from the proof containing these sentences:

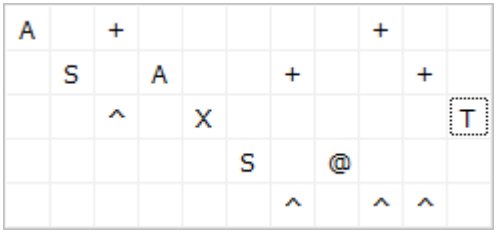




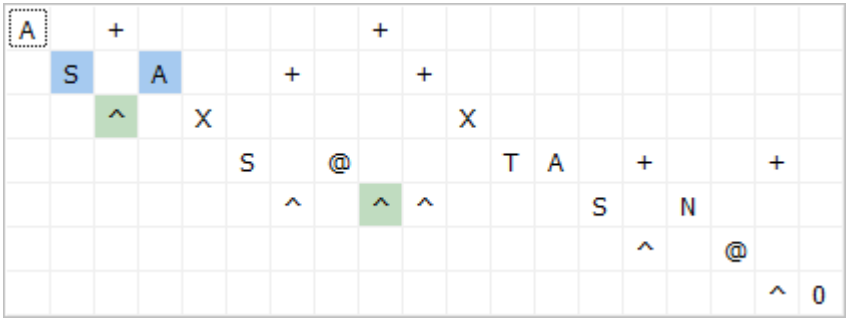
If we select the form in position 1 (cell (1, 1)) of the destination step, by using our operator, we obtain:



In the next example, we select the form at position 11 in the destination:



By using “Inserting a sentence into another sentence” we obtain:



Instead of conclusions

Our solution is a functional logic system named "Logic". The system consists of a relational database and a user interface (“Logic.exe”). The user interface provides an easy but powerful sentence editor using logical operators and a proof module based on inference operators. This system makes the mathematician’s work easier and absolutely precise. Of course, our system also offers other options, e.g., a simplified search according to various criteria, import and export functions for sentences and proofs, and much more.

We can use the sentence editor to generate sentences and the proof module to generate theorems. All the results are saved in the database for later use.

A sentence generated with the sentence editor is correct or unfinished. Wrong sentences cannot be generated.

A proof of a theorem, made by the proof module, is not essentially different from a traditional proof made by a mathematician. In the process of proving a theorem, the mathematician calls the desired inference operators. Because every step of a proof is made by the call of an inference operator, it can only be correct. Obviously, a proof of a theorem may not be complete yet, but it cannot be wrong. To show the advantages of using the Logic system, we have created an introduction to an axiomatic class theory based on an adapted axiomatic.

Bibliography

Bernays, P., *Axiomatic set theory*, New York, Dover Publications, 1968.

Fraenkel, A. A., *Set Theory and Logic*, Reading, Mass., Addison-Wesley, 1966

Halmos, P., *Naïve Set Theory*, Princeton, Van Nostrand, 1960.

Hamilton, A. G., *Logic for Mathematicians*, Press Syndicate, Cambridge, 1988.

Pinter, Charles C., *A Book of Set Theory*, New York, Dover Publications, 2017.

Suppes, P., *Axiomatic set theory*, New York, Dover Publications, 1972.

Van Dalen, Dirk, Logic and Structure, Berlin Heidelberg, Springer-Verlag, 2008.