Mathematical sentences as matrices

A. Zelmer

For our computer-assisted proof system, we have created a formalization of sentences, as well as a functional formalization of proofs. In the following, we will call our system the "Logic" system. In this paper, we present the formalization of sentences.

The sentences we propose in this paper have the following essential properties:

- A sentence is a tree of logical operators.
- Logical operators are forms or terms.
- Logical operators are predefined or user-defined.
- Sentences do not have to satisfy any syntactic rules.
- A sentence is self-explanatory in the sense that the arguments of each logical operator are highlighted.
- A sentence is unique in the sense that it does not contain elements on which it does not depend.
- A sentence is portable, i.e., it can be included in another sentence without changes.
- A sentence is easy to translate into common languages.
- A sentence is absolutely accurate.

What is the reason for using variables?

To generate sentences (axioms, definitions, theorems), mathematicians use variables. What the variables actually are is explained through more or less successful examples. If a variable is quantified, it is said to be a bound variable; otherwise, it is a free variable. A sentence is a formula that has no free variables. Let's note that in the process of creating a sentence, the variables are first used and then quantified.

Usually variables are presented as being part of formal languages. But if we think that to define a variable we need an identifier, this means that implicitly any identifier is part of that formal language. Strange! In what follows, we will look at the variables in a completely different way from the usual way presented above.

Let's try to analyze a simple example of a mathematical sentence:

*Let **a**, **b** be real numbers. Then the only solution of the equation **x** + **a** = **b** is the real number **b** - **a**.*

If the mathematician were more precise, he could write the sentence as follows:

*For any real numbers **a, b** the set {**x** ∈ **R** | **x** + **a** = **b**} is equal to the set {**b** - **a**},*

*where **R** means the set of real numbers.*

Our sentence can be formalized in the following way:

∀(a ∈ R) ∀(b ∈ R) {x ∈ R | x + a = b} = {b – a}

It should also be noted that there is a problem of principle. The connection between variables and quantifiers is combined with a condition (the belonging). This could be corrected by separating the two issues:

∀(a|a ∈ R) ∀(b|b ∈ R) {x ∈ R | x + a = b} = {b – a}

In this paper we call "class generator" an expression of the form **{x| ... }**.

If a quantifier has no condition, we simply write **∀(x)** or **∃(x)**, where x is the newly defined variable.

Variables can only be defined by using quantifiers or the class generator. In a sentence, a defined variable is unique. If a variable has been defined by a quantifier, then it can be used in the quantifier's arguments. If a variable has been defined by a class generator, then it can be used in the argument of this class generator.

In the following example, the second use of the variable **y** is incorrect, because it is not in an argument of the defining quantifier for the variable **y**:

∀(x|x ∈ R) ((∀(y|y ∈ R) (x ≤ y)) ∨ (y < x))

We will now analyze the theorem that characterizes the equality of two functions:

Theorem

*Let **f, g** be functions from **A** to **B**.*

***f** is equal to **g** if and only if **f(x) = g(x)**, ∀**x** ∈ **A**.*

To be able to say that **f** and **g** are two certain functions defined on **A** with values in **B**, we must first specify that **A** and **B** are two certain classes:

∀(A) ∀(B)

Now we will be able to specify what **f** and **g** represent:

∀(A) ∀(B) ∀(f|f:A→B) ∀(g|g:A→B)

Finally, the formalized sentence is

∀(A) ∀(B) ∀(f|f:A→B) ∀(g|g:A→B) (f = g ↔ ∀(x|x ∈ A) f(x) = g(x))

In our example, we have used some variables: **A**, **B**, **f**, **g**, **x**. But it is absolutely clear that the meaning of our statement doesn't depend on these variables. We can use other variables instead.

What is the real reason for using variables? The answer is very easy: they are actually links to quantifiers! This simple observation will allow us to eliminate the variables from mathematics.

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

In the “Logic” system, the usual mathematical definitions are replaced by defining new logical operators (user-defined logical operators). They can be forms or terms. Double implication and equality are used to define them.

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.

Sentences

Let's begin with a sentence in a usual formalization:

∀(x|S(x)) ∃(y) x ∈ y

Unfortunately, this formalization does not satisfy any of the necessary conditions for sentences in the "Logic" system. We will now progressively transform this sentence into a formalization based on our logical operators.

To avoid brackets, we use the Polish notation of Jan Lukasiewicz.

The first quantifier has three arguments: "x", "S", and "E":

- "x" is the variable of the quantifier.
- "S(x)" means "x is a set", and "S" is its logical operator.
- The first operator of "∃(y) x ∈ y" is "E" (the existential quantifier).

The second quantifier has only two arguments, "y" and "@":

- "y" is the variable of the quantifier.
- In our notation, "x ∈ y" is "@(x, y)" and its logical operator is "@".

Because we want to use only conditional quantifiers, we can use the condition "T", which actually means there is no condition, so that the second quantifier will have the arguments "y", "T" and "@". In the table with logical operators, we defined quantifiers as having two arguments, but these quantifiers have three. We will solve this problem at the end of this transformation process.

"S" has the argument "x". The Polish notation of "S(x)" is "Sx".

"@" has the arguments "x" and "y". The Polish notation for "@(x, y)" is "@xy".

We can better see the result of these transformations in the following table:

x ∈ y	@xy
∃(y) x ∈ y	EyT@xy
∀(x|S(x)) ∃(y) x ∈ y	AxSxEyT@xy

The sentence "AxSxEyT@xy" cannot be easily understood… Let's try to illustrate it as a tree, using the relationship between logical operators and their arguments:

```
A
  x S   E
      x   y T @
                x y
```

The arguments of logical operators are highlighted by placing them on the next line, to the right of the operator. In the next step, we place the first argument of a quantifier (its variable) under the quantifier:

```
A
x S   E
    x y T @
            x y
```

Now we put "^" above each variable:

```
A
^ S   E
x   ^ ^ T @
    x y     ^ ^
            x y
```

To translate this text into natural language, we proceed from left to right. In the first vertical position, we find the logical operator, or, in the case of user-defined operators, the first character of the logical operator. Under quantifiers and class generators, we have their variables with a "^" above. As we use the Polish notation, we must take care to be consistent. We don't say "p and q", but "the conjunction of p and q", and so on. Don't forget that natural languages are not perfect. Now we can read our sentence:

*For any class **x**, satisfying **x** is a set, there exists a class **y**, such that we have the belonging of **x** and **y**.*
or
*For any set **x**, there exists a **y**, such that we have the belonging of **x** and **y**.*

We can say that a variable is defined by a quantifier (or a class generator) and then is used as an argument of other logical operators. For example, the variable "x" is defined by the first quantifier and is used as the first argument of the operator "S" in the second position.

The next step will be to indicate with a "+" the relationship between a used variable and its defining quantifier (or class generator):

```
A - + - - - + -
^ S   E - - - +
x   ^ ^ T @
    x y     ^ ^
            x y
```

For example, the first “+” of the first line shows that the variable “x” is related to the first quantifier. For a better visualization of the quantifier line, we marked the empty spaces with "-".

We use this representation of a sentence as a compromise made for readers who have not yet become familiar with the final form of a sentence.

To obtain the final form of a sentence, we completely give up everything written under the quantifiers and then everything written under the links (the variable names). By doing this, the meaning of the sentence does not change:

```
A - + - - - + -
  S   E - - - +
    ^   T @
            ^ ^
```

For example, we can see that the argument of “S” is related to the first quantifier. It’s nice to see that by this step, the quantifiers have become binary, and the variables are replaced by links to the quantifiers!

The arguments of a logical operator are the logical operators located on the next line (the line below), to the right of the operator, up to (exclusively) the first logical operator located on the same line as or above our logical operator.

For example, the first quantifier has two arguments, “S” and “E”. The belonging operator “@” has two arguments “^”. The first is a link to the first quantifier, and the second is a link to the second quantifier.

This last formalization is exactly the formalization based on our logical operators, and it satisfies all the necessary conditions for our system. We will call it the text form of a sentence.

Our example shows that we can consider a sentence as a tree formed of logical operators, starting from a form and including a graphical structure to represent links.

In the “Logic” system, the text form of a sentence only serves the purpose of visualizing sentences.

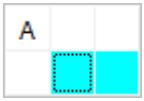
Generate a sentence using the sentence editor

In the "Logic" system, sentences are generated in the sentence editor. The sentence editor provides the user with all the resources necessary to generate a sentence: all predefined and user-defined logical operators (definitions). The result is a matrix of logical operators. The cell (m, n) of the matrix is the intersection of the column m with the row n. Remember that for the cell (m, n), m represents the column and n the line!

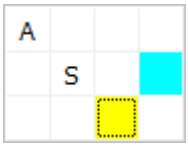
Because each logical operator occupies exactly one column of the matrix, we can refer to a logical operator by specifying the column (position) on which it is located.

The root is always the cell (1, 1) of the sentence editor. In other words, the root is the upper left corner of the sentence editor. Initially, its background color is sky blue. This means that the root is free, and you can insert here a logical operator with a form as a result:

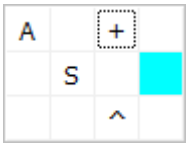
The button “A” is enabled, and we use it to insert the first quantifier:



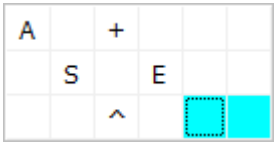
The cursor is now on the condition of the quantifier, that is, on the cell (2, 2). The cells (2, 2) and (3, 2) are the arguments of the quantifier. Their background color is sky blue. This means that these arguments are not completed, and you can insert logical operators with a form as a result. In order to insert in cell (2, 2) the condition of the quantifier, we use the enabled button “S”:



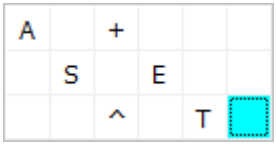
The cursor is now on the cell (3, 3) with a yellow background. This means that this cell is not completed, and you can insert a logical operator with a term as a result. Of course, only such operators are enabled. We want to make a link between this cell and the quantifier. In order to do that, we click the cell (3, 1):



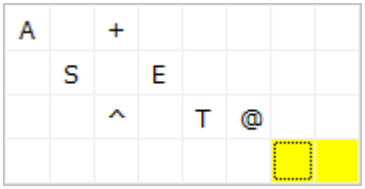
The result is a link between the argument of “S” and the quantifier. This is our modality to replace variables with links. Then we click the cell (4, 2), and we use the button “E” to insert the second quantifier:



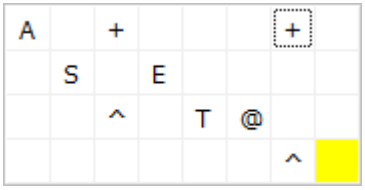
This existential quantifier has no condition, so we put here the true operator:



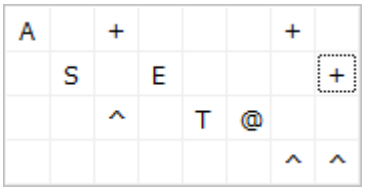
For the cell (6, 3), we use the button “@” for the belonging:



The yellow cells are term cells. They are the arguments for belonging. For the first argument, we want to define a link to the first quantifier. In order to do that, we must click on the cell (7, 1):



Similarly, we generate a link between the second argument of the belonging and the second quantifier:



When the matrix no longer contains sky blue or yellow cells, the sentence is ready, and it can be saved in the database. We can also illustrate the sentence as a text:

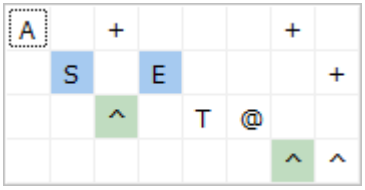
```
A - + - - - + -
  S   E - - - +
    ^   T @
          ^ ^
```

In the version without variable names, we can understand the meaning of the sentence by following the graphical representation of the links. If we wish, we can also define variable names:

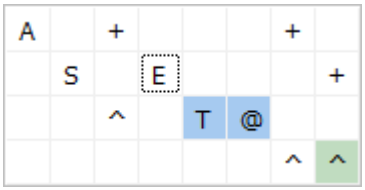
```
A - + - - - + -
^ S   E - - - +
x   ^ ^ T @
    x y     ^ ^
            x y
```

In both forms of a sentence (matrix and text), we can identify the arguments of any logical operator.

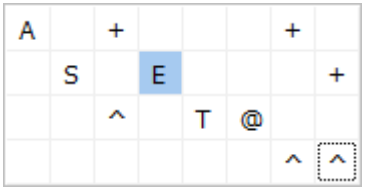
In the matrix form, we can use the position of the cursor to highlight the arguments of a logical operator. In the case of a quantifier, we can also see all the links to it. In our example, if we put the focus on the first quantifier, we can see its arguments (blue) and all the links to it (green, (3, 3) and (7, 4)):



The same is valid for the second quantifier:



If we put the cursor on a link, in this case on the cell (8, 4), the background color of the pointed quantifier or class generator becomes blue:



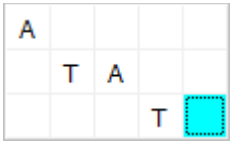
Generating a sentence containing a class generator

The class generator is a unary logical operator with a form as argument and a term as result. But it is also a pseudo quantifier, so we can define links to it.

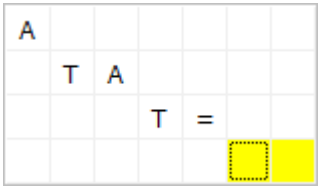
Let us try to use the sentence editor to generate the following sentence:

A(X) A(Y) X∩Y = { u | u ∈ X ∧ u ∈ Y}

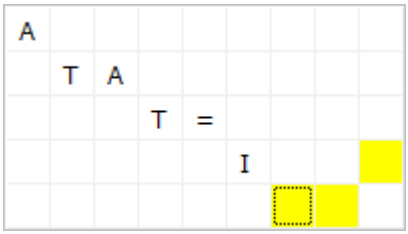
First we insert two universal quantifiers with no condition:



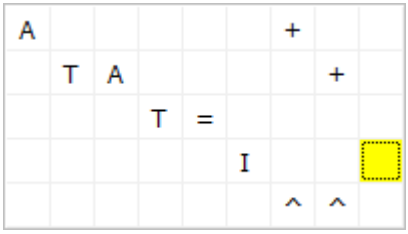
Now, let us insert at position 5 (cell (5, 3)) the equality operator:



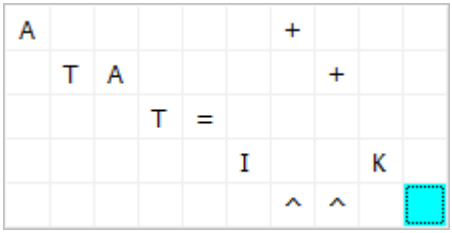
At position 6, we insert the operator “I” (intersection), which we choose from the table of defined operators:



By clicking on the cell (7, 1), we link the first argument of the intersection to the first quantifier. Similarly, by clicking on the cell (8, 2), we link the second argument of the intersection to the second quantifier, and then we select the position 9:



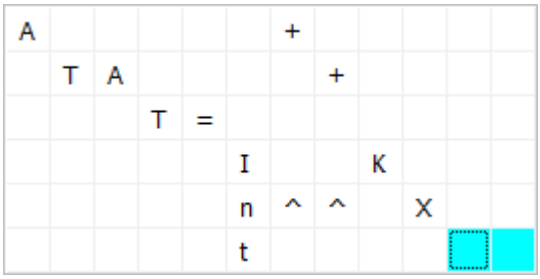
For the second argument of the equality, we use the class generator “K” in position 9:



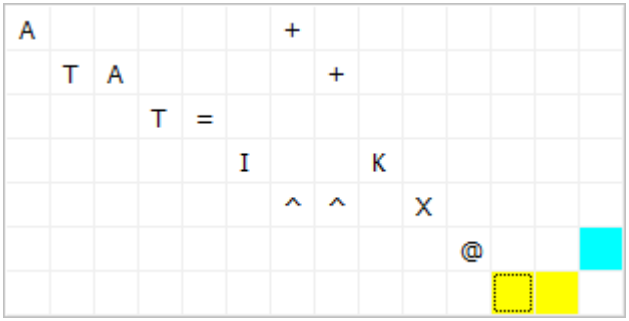
As the argument of “K” has a blue background, we have to insert a form here. The form, which we want to insert, is

$u \in X \wedge u \in Y$

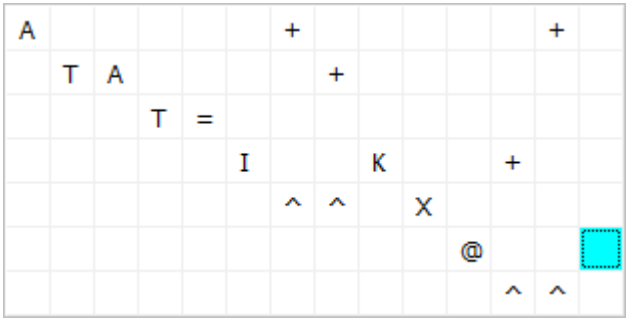
First, we insert the conjunction:



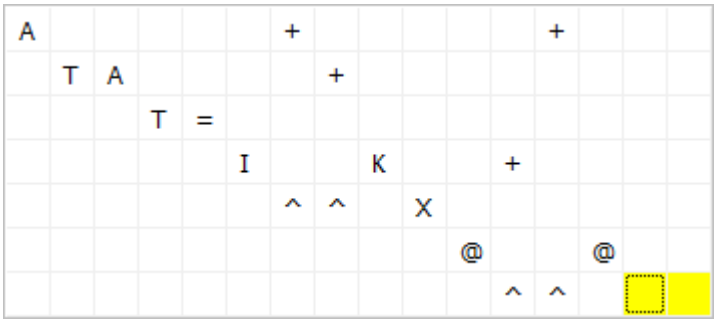
The first argument of the conjunction is a belonging:



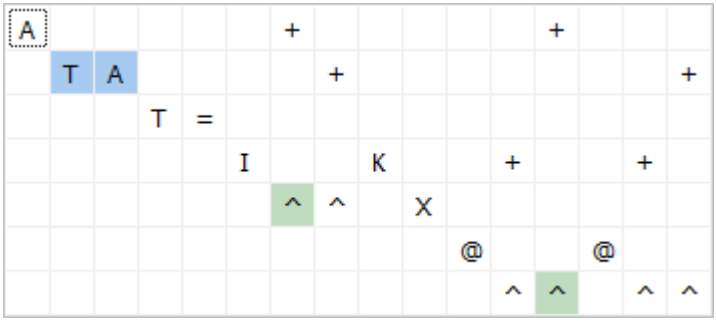
Instead of the variables **X** and **Y**, we use links to the first and the second quantifier, but what to do with the variable **u**? The answer is quite simple: we use a link to the class generator! By clicking on the cell (12, 4), we link the first argument of the belonging to the class generator, and by clicking on the cell (13, 1), we link the second argument of the belonging to the first quantifier, and then we select the position 14:



At position 14, we define the second belonging:



Finally, we link the first argument of the belonging to the class generator and the second argument of the belonging to the second quantifier, and then we select the position 1. Now our sentence is ready:



We can give the names “X”, “Y”, and “u” to the variables, so the text form of the sentence is

```
A - - - - - + - - - - - + - - -
^ T A - - - - + - - - - - - - +
X   ^ T =
    Y     I     K - - + - - + -
            ^ ^ ^ X
            X Y u   @     @
                  ^ ^   ^ ^
                  u X   u Y
```

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.

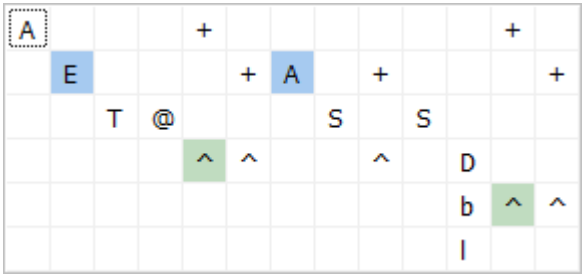
If in the definition of the pre-q-list we refer only to universal quantifiers, we obtain the notions of **u-pre-q-list**, **u-prefix-arg**, and the **u-prefix**.

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).

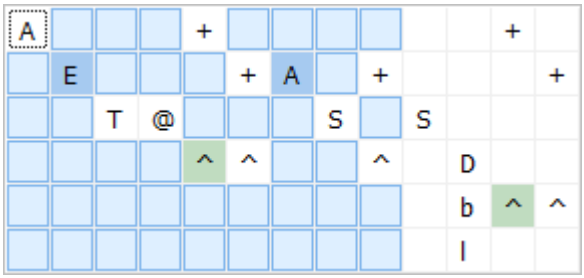
Example

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.

User-defined logical operators

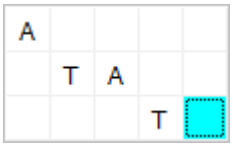
In our system, mathematical definitions are made by creating new logical operators (user-defined logical operators), which are forms or terms. This can be done easily using the sentence editor. For this, we must first insert into the sentence editor the prefix of a new sentence, containing only universal quantifiers. Then we will insert a double implication “D” for a new form or an equality “=” for a new term, and we will press the “Define” button. The sentence editor asks us for the name of the new form or term and automatically generates the first argument of the double implication or equality to be defined. We only have to complete the second argument and save the definition.

The names of the user-defined logical operators must be representative and cannot be changed later.

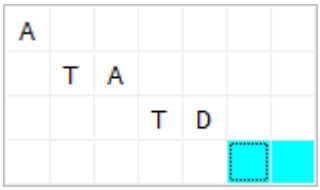
For example, let us define the new (primitive) relation XYZ like this:

`A(x) A(y) (XYZ(x,y) ↔ ...)`

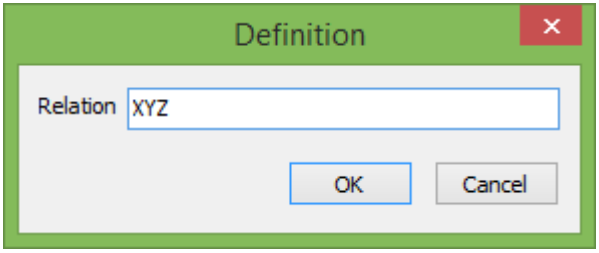
We first insert the prefix into the sentence editor:



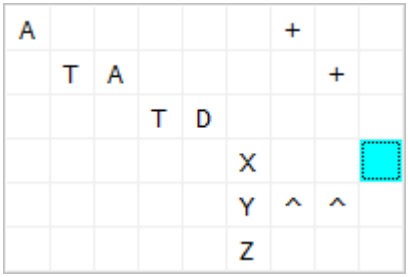
Now, we insert the double implication:



At this moment, we press the “Define” button, and the sentence editor asks us for the name of the new relation:



After we have filled in the name “XYZ”, we press the “OK” button:



We must now complete the definition with the part “…” and save the sentence. The new (primitive) relation XYZ is ready to be used.

Each definition receives a unique name.

Let's now try to define the notion of disjoint classes (Dis) in our system:

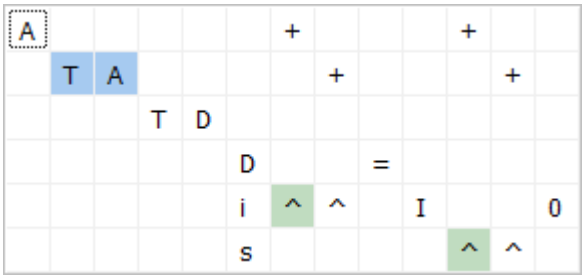
Definition

*Let **X, Y** be classes. **X, Y** are disjoint if and only if the intersection of **X, Y** is equal to 0.*

In string form, the definition of disjoint classes is:

```
A(X) A(Y) D(Dis(X,Y),=(I(X,Y),0))
```

In our system, this definition looks like this:



As we can see, the names of the user-defined logical operators are displayed vertically so that each logical operator occupies only one column. In the text form, we do the same. We can easily recognize the tree structure of the sentence.

It is very important to notice that the text form of a sentence also includes a graphical representation of the links to quantifiers:

```
A - - - - - + - - - + - -
  T A - - - - + - - - + -
      T D
          D     =
          i ^ ^   I     0
          s         ^ ^
```

To facilitate the conversion of our sentences to the usual mathematical language, we can define variables:

```
A - - - - - + - - - + - -
^ T A - - - - + - - - + -
X   ^ T D
    Y     D     =
          i ^ ^   I     0
          s X Y     ^ ^
                    X Y
```

Saving sentences to the database

After we have generated a sentence in the sentence editor, we can save it to the database. A definition that creates a new logical operator, which is a form, becomes the kind “R” in the database. A definition that creates a new logical operator, which is a term, becomes the kind “F” in the database. For other sentences, we have the

possibility to declare them as axioms with the kind “A” or to let them be unknown with the kind “U”. We can later generate a proof for such a sentence, and then the sentence becomes the kind “T” (theorem).

After saving a sentence to the database, we have the possibility of using a memo field to describe the sentence. The title and the description of a sentence can be changed at any time.

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, …).

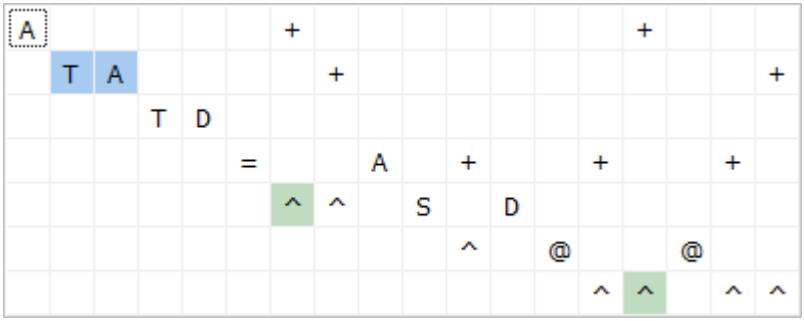
Two logical operators of a sentence, which are not the root, have a **first common** ancestor.

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**.

A **link** is a reference from a logical operator “^” to a quantifier or class generator, which must be an ancestor of this logical operator.

For sentences, we can use two types of representation: **matrix** and **text**.

Matrix:



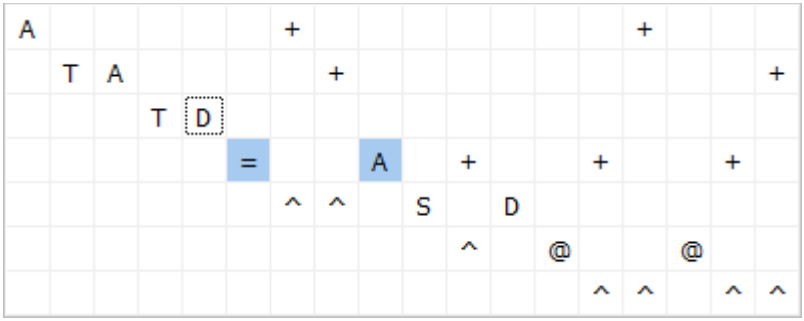
Text:

```
A - - - - - + - - - - - - - + - - -
^ T A - - - - + - - - - - - - - - - +
X   ^ T D
    Y     =     A - + - - + - - + -
            ^ ^ ^ S   D
            X Y u   ^   @     @
                    u     ^ ^   ^ ^
                          u X   u Y
```

As can be seen in our example, in the text representation of a sentence, we can define variables with names. Defining variable names only aims to make it easier to translate our sentences into natural language.

In both representations, each column represents a logical operator. For this reason, to indicate a logical operator of a sentence, it is enough to indicate the column of the logical operator. We read the name of a logical operator top-down.

In the matrix representation, if we put the cursor on a logical operator, the background of its arguments turns blue:



In this example, the cursor is on the cell (5, 3). The arguments of the double implication have a blue background. But we can also easily identify the arguments of a logical operator from the tree structure of a sentence. The arguments of a logical operator are the logical operators located on the next line, right from the operator, up to (exclusively) the first logical operator located on the same line as or above our logical operator.

The character “+” defines the links, i.e., it illustrates the connection between a quantifier and a logical operator “^”.

The main qualities of our sentences are:

- They are unique in the sense that they do not contain elements on which they do not depend.
- They are portable, meaning they can be combined without having to be adapted.
- The arguments of each logical operator are highlighted by their position.
- Each logical operator occupies exactly one column, so the number of logical operators in a sentence coincides with the number of columns.

We can hide auxiliary sentences. Auxiliary sentences can be seen by checking the button “Aux”.

The reverse process (from our sentences to the usual mathematical language)

To make it easier to understand the use of our sentences, we will translate these sentences into a mathematical language close to the usual language. If a quantifier has the condition "T" (i.e., it has no condition), it will be translated as follows:

```
A(x) q(x)
E(x) q(x)
```

If the quantifier has the condition p(x), then it will be translated into:

```
A(x|p(x)) q(x)
E(x|p(x)) q(x)
```

Let's not forget the meaning of conditional quantifiers:

```
A(x|p(x)) q(x) ≡ A(x) (p(x) → q(x))
E(x|p(x)) q(x) ≡ E(x) (p(x) ∧ q(x))
```

We used the signs “$\rightarrow$” and “$\wedge$” here, but in our system, we will use the logical operators "C" and "X" instead. Here are four concrete examples:

```
A(X) A(Y|@(X,Y)) S(X)
A(X) A(Y) A(Z) C(X(=(X,Y),=(Y,Z)),=(X,Z))
A(x|S(x)) A(y|S(y)) =(Dbl(x,y),{u|V(=(u,x),=(u,y))})
A(x|S(x)) E(y) @(x,y)
```