Functional Logic for

Axiomatic Class Theory

A. Zelmer

At the beginning, mathematics consisted of a few scratches on stone or circles on sand, with a touch of universality and perfection. Then curious signs appeared on paper, which later tormented the text editors with new, useful, or useless signs: circles, dashes, brackets, and more. All these elements were assembled through the natural language, forming axioms and theorems. The theorems had to be proved, and the proofs were also made in natural language. The use of natural language in statements and proofs has turned them into more or less accurate stories. Does math really need all these elements? Is it not possible to reach, by simple means, based on a few primordial elements used consistently, the full rigor and perfection of the expression of mathematics?

Based on the notion of primitive function, we have built an integrated computer system, which generates an elegant and simple environment, capable of representing statements and proofs in an absolutely precise way. For this, we have used logical operators, predefined or user-defined, which are primitive functions whose arguments and results are forms or terms. By composing such logical operators, we can generate any statement. We have also encapsulated the notions of logic in our inference operators, which are used to generate the proofs of the theorems. The system allows us to generate only correct statements, and theorem proofs become calls of inference operators. The proofs, although in principle similar to traditional demonstrations, are absolutely rigorous.

Sentences are obtained by composing logical operators, so they can be represented as trees, with the arguments of each logical operator highlighted. This representation allows us to make global visual assessments and comparisons of the statements. It should be noted that the statements can be easily translated into traditional form.

In order to present our system, we will build into it an introduction to the axiomatic theory of classes. In this way, we will concretely highlight the concepts set out above.

A. Zelmer

Contents

Preliminary considerations 9
What does mathematics mean to us? 9
For whom this work is intended and how it can be used 10
How are variables defined in mathematics? 11
What is the real reason for using variables? 12
The fight against brackets 14
Avoiding erroneous or imprecise sentences 15
Requirements for sentences and proofs 18
Our solution 19
Logical operators 22
Sentences 25
No language. No syntax. No brackets. No variables. No predicates. 25
Generate a sentence using the sentence editor 27
Generating a sentence containing a class generator 31
Prefix 34
User-defined logical operators 36
Saving sentences to the database 38
Relationships between the logical operators of a sentence 39
The reverse process (from our sentences to the usual mathematical language) ...41
Proofs 42
An example of a proof 44
Defining a logical operator using an existence and a uniqueness theorem 55
Loading sentences into the proof 57
Inference operators 58
Classification 61

Extracting the hypothesis 62
Tautologize 63
Reductio ad absurdum 64
Negation 65
Class generator to form 66
Form to class generator 67
Attachment 68
Replacement 70
Duplication (conjunction) 71
Duplication (implication) 73
Existential generalization, Universal generalization (using equality) 75
Extracting the condition of the quantifier 76
Inserting the condition into the quantifier 79
Deleting a quantifier 80
Deleting quantifiers from right to left 83
Dual quantifier distributivity 83
Partial quantifier distributivity 85
Restricted quantifier distributivity 87
Quantifier distributivity 88
Interchanging quantifiers 89
Particularization 91
Commutativity 93
Reflexivity 93
Cross attachment 93
Cross replacement 96
Inserting a sentence into another sentence 98

Formulas 99
Using formulas 100
Generating new formulas 101
Verifying the proof 101
The “Logic” program 103
Edit 105
Proofs 106
Definitions 107
Categories 108
An introduction to the axiomatic class theory 109
Classes and sets 110
Cartesian product 152
Graphs 158
Generalized union and intersection 165
Graph functions 172
Simple graph functions 178
Axiom of global choice 188
Various properties of graph functions 193
Functions 199
Relations 214
Natural numbers 220
Finite recursion 227
Arithmetic of natural numbers 235
Various properties of the natural numbers 0, 1, 2 253
Natural numbers as domain (Sets, Finite, Path) 255
Division of natural numbers 304

Less “natural“ properties of natural numbers 310
Instead of conclusions 311
Bibliography 312
Index 313

Preliminary considerations

What does mathematics mean to us?

For many, mathematics is something difficult, too difficult, which is of no use in life anyway. For others, such as engineers and physicists, mathematics is a toolbox. They look for the right tool and use it to the best effect. This generally means knowing how to do complicated calculations without making too many mistakes and, above all, without asking yourself about the validity of the used formulas. For mathematicians, it means an enthusiastic search for new interesting properties, but there is also the institutionalized mathematics, where titles, doctorates, and the number of published works are measures of knowledge of mathematics. What could mathematics mean for us? Couldn't we look at math as a game? Let's try!

Our game pieces will be of two colors: sky blue and yellow. We will call the sky blue ones forms and the yellow ones terms. Forms and terms are just words; let's not try to give them any interpretation. Our game pieces will be able to be combined by some very simple rules, forming items that we will call sentences. For now, the sentences are no good, but this will change. We will start by declaring as axioms the sentences that we like more, but we will do this very carefully and sparingly so that we have as few axioms as possible. Certain sentences may be declared as definitions, a process that will generate new forms or terms. Newly defined forms and terms will receive new names so that we can use them later. But most of the sentences we can create will be neither axioms nor definitions. In the state in which we created them, they are of no use. Then why should we create them?! Well, here begins our math game!

Like any game, our game will have very clear and precise rules. We will call all these rules logic. We don't have to be scared; the "logic" here has nothing to do with Aristotle or Gödel! In principle, the rules of our game only have to somehow show us if a sentence is a theorem. How do we proceed? First, we make a new sentence, which we think is very beautiful. To become a theorem, our sentence needs a demonstration, which is done by using the rules of our game (logic!). If the demonstration is correct, logic will say "bravo" and mark the sentence as a theorem. You have won a point, and you go further and further and further, endlessly. This is the math we're talking about.

Of course, our game will be installed on our computer or tablet. There, we will have everything we need. We will be able to see all forms and terms, and we will be able to combine them to form sentences using the sentence editor. We can prove the sentences using logic. Well, the beauty and difficulty of our game consist of creating sentences that we can prove. But that's not all! It is even more interesting to exchange theorems with friends. The challenge "who makes the most beautiful theorem" can also be invented.

Is the game we proposed just a joke? No, it's not a joke; on the contrary, it's much more. First, it is our opinion that true mathematics should not be based on a natural language. We know very well that natural languages appeared as a necessity for more complex communication. Communication means the exchange of information, but that does not mean that the information transmitted is true. Natural languages are also very good at transmitting false or imprecise information. Secondly, we suggested that logic is just a convention, a collection of rules that must be respected in the process of making a demonstration. Moreover, we suggested that there is no need to learn logic but that it can be very well packed in a simple-to-use module. Although in our paper we tried to use the principles of usual logic, we did this only to show that traditional mathematics can be perfectly redefined using our game. However, we do not exclude the use of other logic modules, known or new. From our point of view, anything can be used.

Another aspect of the math game is the fact that it allows us to use the elements precisely. For example, the sentence editor will only allow the creation of correct sentences, and the demonstration module will only allow correct demonstrations. Our demonstrations will be complete. It will not be enough to know that something is "true". We will have to put it in the demonstration, that is, the axiom, definition, or theorem on which we rely. This is not easy because many elements are involved in a demonstration, but it is absolutely necessary. Only such a demonstration is truly accurate. Our game will immediately help us find the elements we need. Everything will be available on our computer or tablet.

For whom this work is intended and how it can be used

The present paper is addressed to those interested in the process of mathematical thinking in its most accurate form and who are convinced of the invaluable help of a computer in this process.

Ideally, to understand the functionality of our system, it must first be installed on the computer. Using this system, the reader can independently verify each step of the mathematical process. Only in this way can the reader realize the power and simplicity of the system we propose. To make this easier, we will provide the user with a database with all the elements mentioned in this work, as well as an empty database on which experiments can be done. For example, within a group of passionate students, the database can be installed on a server, and everyone will have access to all the elements and will be able to add their own new theorems. All elements of our system are open source.

For those who do not want to install our system, we have also prepared a text file containing all the sentences from the database together with the proofs of the theorems. Using a simple text editor and a font of constant width, all elements can be analyzed.

The work is not addressed to those who want to quickly pass through the fundamentals of mathematics in order to reach certain mathematical fields, which they want to use for various purposes.

How are variables defined in mathematics?

In mathematics, variables are not defined; they just appear: x, y, or something like that. It's no wonder that people still try to understand what they are:

- In mathematics, a variable (from Latin variabilis, "changeable") is a symbol, typically a letter, that refers to an unspecified mathematical object.
- In math, a variable is an alphabet or term that represents an unknown number, unknown value, or unknown quantity.
- A variable is any characteristic, number, or quantity that can be measured or counted. A variable may also be called a data item.
- A variable is a placeholder for an unknown quantity.
- In simple words, a variable is a quantity that can be changed and is not fixed.
- Variables in math are symbols, often letters, that represent different values in various situations. They help us understand and solve problems with changing values.
- A variable is a quantity that, during a calculation, is assumed to vary or be capable of varying in value.

- A variable is an element, feature, or factor that is liable to vary or change.
- Variables are lowercase letters that represent variable values.

We could complete this list with many other sad or joyful attempts, depending on how we look at things!

What is the real reason for using variables?

We have never heard such a sentence: "For any x such that x is a dog, x barks.". In natural languages, pronouns act as variables to avoid repetition. But they are not precisely defined, so we can say that natural languages do not use variables like in mathematics. Programming languages do use variables. In this case, a variable is a memory space where information can be stored, used, or changed within the program. In mathematical statements and in proofs of theorems, what a variable represents cannot be changed in any way, so the term “variable” is not appropriate.

Maybe the notion of variable makes sense for predicates. We say that p is a predicate of variable x. That means p is defined using the variable x. We don't know what a variable is, but we use it to define something else...

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.*

The usual language is not precise enough to formulate sentences or make a proof. We think mathematics is one of the few domains that can be perfect. But to be perfect, it must be absolutely precise. For this it is necessary to formalize the mathematical sentences. Our sentence can be formalized in the following way:

$\forall(\mathbf{a} \in \mathrm{R})\ \forall(\mathbf{b} \in \mathrm{R})\ \{\mathbf{x} \in \mathrm{R} \mid \mathbf{x} + \mathbf{a} = \mathbf{b}\} = \{\mathbf{b} - \mathbf{a}\}$

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:

$\forall(\mathbf{a}|\mathbf{a} \in \mathrm{R})\ \forall(\mathbf{b}|\mathbf{b} \in \mathrm{R})\ \{\mathbf{x}|\mathbf{x} \in \mathrm{R} \wedge \mathbf{x} + \mathbf{a} = \mathbf{b}\} = \{\mathbf{b} - \mathbf{a}\}$

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

If a quantifier has no condition, we simply write $\forall(\mathbf{x})$ or $\exists(\mathbf{x})$, 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. After a variable has been defined, it can be used in various expressions necessary to the sentence we want to create.

In the following example, we will 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)**, $\forall \boldsymbol{x} \in \boldsymbol{A}$.*

Proof

*First, let us assume that **f** = **g**. Then, for an arbitrary $\boldsymbol{x} \in \boldsymbol{A}$,*

$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) \Leftrightarrow (\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{f} \Leftrightarrow (\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{g} \Leftrightarrow \boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x})$

*Thus, **f(x)** = **g(x)**.*

*Conversely, assume that **f(x)** = **g(x)**, $\forall \boldsymbol{x} \in \boldsymbol{A}$. Then*

$(\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{f} \Leftrightarrow \boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) \Leftrightarrow \boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}) \Leftrightarrow (\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{g}$

*Thus, **f** = **g**.*

Is this really mathematics? Yes, it is, but in our opinion, this is only a story/sketch about mathematics! “Let f … be” is like song lyrics, but “Let it be” is something different! What does “let us assume that” mean? What does “for an arbitrary x” mean? Of course, we understand all of that, but we cannot accept it as mathematics. We can understand all of these lines as a proof’s sketch, but nothing more. Let's still try to formalize this sentence.

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.

The fight against brackets

In mathematics, definitions are used to generate new primitive functions or relations. Defining a primitive function/relation means specifying its name/symbol, its arguments, and, for primitive functions, its result. It is customary for the value of a primitive function to be represented by its name/symbol followed by the list of arguments, for example:

```
f(x, y)
+(a, b)
:(12, 3)
sin x
∩(X, Y)
```

Depending on the compatibility of their arguments, primitive functions and relations can be composed, resulting in more complicated expressions:

```
+(12, ·(5, -2))
sin(x² + 1)
ln(x + 1)
```

```
∩(X, \(Y, Z))
```

For some binary operations, the first argument is placed before and the second after the operation name/symbol:

```
x + y
5 – 2
12 + (5 · (-2))
X ∩ (Y \ Z)
```

In the case of associative operations, an abbreviated writing is also used, for example:

```
x + y + z
5 · 2 · 9 · 14
X ∪ Y ∪ Z
X ∩ Y ∩ Z
```

The purpose of these conventions is to reduce the number of brackets needed to write more complicated expressions. Eliminating brackets can only work by introducing a collection of rules for prioritizing the order of operations.

If we analyze the discussions on the internet about evaluating simple expressions, such as

```
3 * 3 - (3 : 3) + 3
```

(Math Challenge), we can only wonder how something like that is possible. And this happens after years of school! Is it only the school's fault?!

Wouldn't it be possible to completely eliminate the use of brackets and make mathematical expressions easier to understand?

Avoiding erroneous or imprecise sentences

The purpose of this work is not to find imprecise or incorrect mathematical sentences. However, in the process of adapting the sentences, we had various problems. We will show here some examples:

- Let's try to analyze the following theorem:

Let x_1, x_2, … , x_n be distinct elements.
The function $f : \{x_1, x_2, \dots, x_n\} \to \{x_1, x_2, \dots, x_n\}$ is …

What does "*Let x_1, x_2, … , x_n be distinct elements.*" mean? Obviously, this means "For any x_1, and for any x_2, and so on for any x_n". Yes, in natural languages, we can use the expression "and so on" with more or less meaning. The same cannot be said in the case of formal languages, where such expressions do not exist. Let's see now what happens with the set *$\{x_1, x_2, \dots, x_n\}$*. We can easily define the sets $\{x_1\}$ and $\{x_1, x_2\}$:

$\{x_1\} = \{x \mid x = x_1\}$

$\{x_1, x_2\} = \{x \mid x = x_1 \vee x = x_2\}$

But the logical expression “$x = x_1 \vee x = x_2 \vee \dots \vee x = x_n$” cannot be accepted, so the set $\{x_1, x_2, \dots, x_n\}$ cannot be used either.

Some authors, who want to be more precise, define the set *$\{x_1, x_2, \dots, x_n\}$* recursively:

$\{x_1\} = \{x \mid x = x_1\}$
$\{x_1, x_2, \dots, x_n, x_{n+1}\} = \{x_1, x_2, \dots, x_n\} \, U \, \{x_{n+1}\}$

If we analyze the set $\{x_1, x_2, \dots, x_n\}$, we find that x_n is nothing but an undeclared function. Using the properties of natural numbers, the notion of function, and the notion of range of a function, we can define the set $\{x_1, x_2, \dots, x_n\}$ correctly, elegantly, and above all, without "and so on".

In the particular case of the set {0, 1, ... , n}, where n is a natural number, there is a simpler and more elegant solution to avoid the expression "and so on". The natural number n + 1 is exactly the set we are looking for!

• To make statements simpler, elliptical expressions are often used. For example, it is said that the set of natural numbers is well-ordered, but the specification of the order relation is omitted. If in ordinary speech there is no problem, the same is not permitted in mathematics, where we want the statements to be perfect. In other words, the reference to a primitive relation or function must contain all the necessary arguments, exactly as they were defined.

• Another serious problem appears in the proofs of some theorems. The statement and the proof of a helping theorem have no place in the proof of our theorem, so formulas like "let x be a natural number" have no place in a proof either. We can always state and prove the auxiliary theorem before our theorem and then use the auxiliary theorem in our proof.

• We consider that the inferences of a demonstration must be made in situ and not by the arbitrary extraction and separate processing of some elements belonging to the sentence.

• A big problem is the overloading of the names of many elements. For example, the "+" symbol is "successfully" used for adding natural numbers, integers, and other numbers, as well as for other "additions", such as adding vectors. In fact, it is also used to denote a certain binary operation. In other words, "+" denotes completely different things.

No one is surprised if someone writes 7 - 3 = 4 and says that -3 is an integer number. But the first "-" represents the subtraction, which is a binary operation, and the second "-" represents the opposite of an integer, which is a unary operation.

It is a logical mistake to denote two different things by the same sign (principle of identity).

• As in the examples above, "<" and "≤" are used for various sets of numbers, such as natural, integer, rational, and real numbers. Apart from this, there is another abusive use, but somewhat reversed. We write "x < y" to say that x is less than y. To say that x is positive, we write "x > 0" without defining the new relation ">", as if "<" is an arrow that we can turn in any direction to indicate which is the smaller element!

• It is customary to use chains of equivalences or equalities in demonstrations to make the demonstration more intuitive, for example:

a - b = c ↔ a = b + c ↔ 0 = b + c – a

where the equality "a = b + c" is simultaneously part of two equivalences!

From this chain of equivalences, without invoking the transitivity property, it is concluded that a - b = c ↔ 0 = b + c - a.

• Abusive and unnecessary use of variables. Let's analyze the following examples:

$$\lim_{n\to\infty} f(n) \qquad \sum_{k=0}^{n} f(k) \qquad \int_a^b f(x)dx$$

In the first example, **f** is a sequence of real numbers, and **n** is a letter followed by an arrow pointing to something strange. **f**(**n**) is the value of the sequence **f** for a certain natural number **n**. Which **n**? Does the limit of this sequence depend on the value of **n**? Not at all.

In the second example, **n** is a natural number and **f** is a sequence of real numbers. It is absolutely clear that the expression does not depend on **k**.

The third expression is of particular beauty. It can be seen as a symbol of mathematics! Where the variables **a** and **b** are placed doesn't really matter to us, but what role does **x** play in this expression? Does the expression depend on **x**? Obviously not. What can we say about the expression "**dx**"? It is truly mysterious.

In the case of these examples, as in many other beautiful cases, some mathematicians would say that the variables (**k**/**n**/**x**) are bound variables. But where are the necessary quantifiers? They don't exist. If we look closely at these expressions, we see that these variables also appear in the defined symbols, as if these symbols were quantifiers.

The problem is generated by the confusion between a function and a certain value of the function. In all our examples, as in many other similar cases, the value of the function **f**(**n**) / **f**(**k**) / **f**(**x**) is used instead of the function **f**.

Of course, all these expressions, like many others, have a historical explanation. But that doesn't mean they're correct.

Requirements for sentences and proofs

Our purpose is to create a mathematical system that uses sentences and proofs that are truly accurate, that is, precise. How can we achieve this? What properties must the sentences and the proofs have?

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

- A sentence is a composite function of logical operators.
- Logical operators are primitive functions.
- Logical operators are forms or terms.
- Logical operators are predefined or user-defined.
- We do not need syntactic rules to generate a sentence.
- 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.

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

- A proof of a theorem is a list of proof steps.
- A step in a proof is the call of an inference operator.
- Inference operators are powerful logical functions that use none, one, or two steps of the proof to produce an intermediate sentence as a result.
- The proof is finished when the call of the last inference operator generates a sentence equal to the theorem we want to prove, or, in the case of a proof by reductio ad absurdum, a contradiction.
- A proof must be based exclusively on existing axioms, definitions, or theorems.
- After a sentence (axiom, a definition, or a theorem) has been introduced in a proof, that sentence can no longer be changed, and in the case of a theorem, its proof cannot be changed either.
- When generating the proof of a possible new theorem, it is not allowed to edit sentences or parts of sentences.

Our solution

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 done 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.

The essential difference between proofs in our system and traditional proofs is their completeness. In a proof, we cannot have things that are obvious or left up to the reader. A proof will be based exclusively on existing axioms, definitions, and theorems.

To show the advantages of using the "Logic" system, we have created an introduction to an axiomatic class theory based on an adapted axiomatic. The space of this paper does not allow us to present all the proofs of this theory. However, we must specify that all the theorems have been proved in our system and can be studied in the system “Logic" or in the file that contains all the sentences and proofs. The proofs in this paper are necessary exercises for understanding the presented system. However, the actual work in our system is only possible within the user interface.

An axiomatic theory is said to be formalized if its axioms and theorems are written in a formal language and all its proofs are formal proofs. It is commonly accepted that ideally an axiomatic theory should be formalized. But mathematicians consider that such an approach is too complicated because the sentences written in a formal language are too difficult to decipher, and the proofs are too long. In conclusion, mathematicians are satisfied to consider that an axiomatic theory can be formalized, after which they develop the theory informally... Our system will show that it is possible to work in mathematics absolutely precisely. The statements are easy to decipher, and the proofs can be made reasonably short.

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

What is a primitive function? Like anything "primitive", a primitive function cannot be defined. It is something taken for granted, accepted as such. It corresponds to our experience, and we use it, consciously or unconsciously, for various things. We can say that it is an assignment, a rule, or a law, but this does not help us at all... However, we know what we should expect from it. First and foremost, we give a primitive function a name or a symbol so that we can recognize it later. Then we know how to use it. We have to provide it with a fixed and finite number of "arguments", depending on which it delivers us a result. We also know that for the same arguments, a primitive function delivers the same result.

In the above, by form (Form) and term (Term) we mean two different primitive notions.

A logical operator is a primitive function. It is identifiable by its name, has arguments, and has a result. The arguments of a logical operator are either all forms or all terms. The result of a logical operator is a form or a term. If the result of a logical operator is a form/term, we consider that the operator is a form/term. For example, "X" and "@" are forms, but "K" and "^" are terms.

An argument of a logical operator can be considered as a location, which is empty, or it contains another logical operator compatible with the argument (both forms, or both terms). Setting a logical operator as an argument of another logical operator means composing the two operators, and the result is a composite operator (of the logical operators). A composite operator is complete if it contains no empty arguments. A logical operator with no arguments is also considered a complete composite operator.

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 definition of conditional quantifiers is based on the formulas:

$\forall(x|p(x))\ q(x) \equiv \forall(x)\ (p(x) \rightarrow q(x))$

$\exists(x|p(x))\ q(x) \equiv \exists(x)\ (p(x) \wedge q(x))$

If a quantifier has no condition, we use the logical operator "T" as its condition.

The operators "K", "^ ", "@ ", "= ", "S" appear in the list of predefined logical operators. They are necessary for a consistent implementation of the requirements for sentences and proofs.

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

How do we express logical operators? Always the same, for example:
“=(x, y)” is "the equality of x and y" (“the equality of x, y”),
“U(X, Y)” is "the union of X and Y" ("the union of X, Y"), or
“Pair(x, y)” is “the pair of x and y” (“the pair of x, y”).

Sentences

No language. No syntax. No brackets. No variables. No predicates.

A sentence is a complete composite operator (of logical operators) whose root (its first logical operator) is a form.

In order to generate a sentence, we use the sentence editor. This editor provides all the allowed possibilities at the moment. The user can select only meaningful operators. When ready, the sentence is correct. Of course, the user can build stupid sentences, but not incorrect ones!

Let's begin with an example. While working with our system, you never have to take these steps. This example only shows you the way to our system. The sentence we want to define is

$\forall(x|S(x))\ \exists(y)\ x \in y$

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 "$\exists(y)\ x \in y$" is "E" (the existential quantifier).

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

- "y" is the variable of the quantifier.
- In our notation, "$x \in y$" 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:

$x \in y$	@xy
$\exists(y)\ x \in y$	EyT@xy
$\forall(x\|S(x))\ \exists(y)\ x \in y$	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 **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.

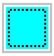
For example, the first quantifier has two arguments, “S” and “E”. The belonging operator “@” has two arguments “^”. The first is a link to the first quantifier, and the second is a link to the second quantifier.

Generate a sentence using the sentence editor

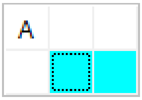
Now we will generate the sentence above using the sentence editor. The result is a matrix of logical operators. The cell (m, n) of the matrix is the intersection of the column m with the row n. Remember that for the cell (m, n), m represents the column and n the line!

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.

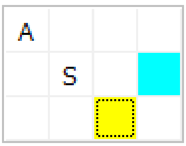
The root is always the cell (1, 1) of the sentence editor. In other words, the root is the upper left corner of the sentence editor. Initially, its background color is sky blue. This means that the root is free, and you can insert here a logical operator with a form as a result:



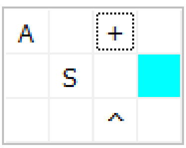
The button “A” is enabled, and we use it to insert the first quantifier:



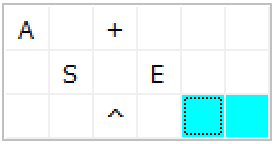
The cursor is now on the condition of the quantifier, that is, on the cell (2, 2). The cells (2, 2) and (3, 2) are the arguments of the quantifier. Their background color is sky blue. This means that these arguments are not completed, and you can insert logical operators with a form as a result. In order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



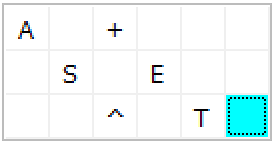
The cursor is now on the cell (3, 3) with a yellow background. This means that this cell is not completed, and you can insert a logical operator with a term as a result. Of course, only such operators are enabled. We want to make a link between this cell and the quantifier. In order to do that, we click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



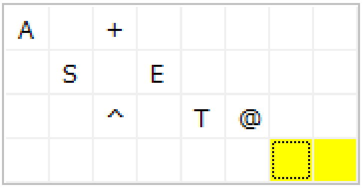
The result is a link between the argument of “S” and the quantifier. This is our modality to replace variables with links. Then we click the cell (4, 2), and we use the button “E” to insert the second quantifier:



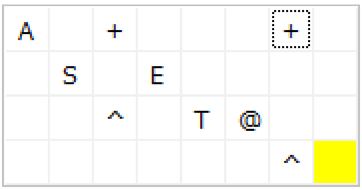
This existential quantifier has no condition, so we put here the true operator:



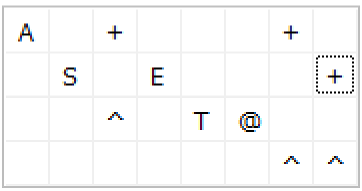
For the cell (6, 3), we use the button “@” for the belonging:



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:

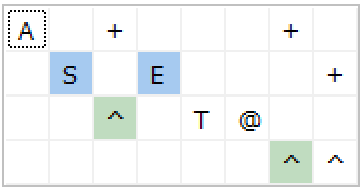
```
A - + - - - + -
  S   E - - - +
    ^   T @
            ^ ^
```

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:

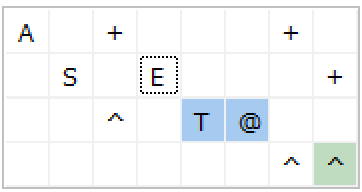
```
A - + - - - + -
^ S   E - - - +
x   ^ ^ T @
    x y     ^ ^
            x y
```

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

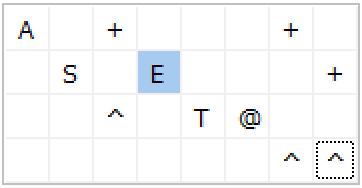
In the matrix form, we can use the position of the cursor to highlight the arguments of a logical operator. In the case of a quantifier, we can also see all the links to it. In our example, if we put the focus on the first quantifier, we can see its arguments (blue) and all the links to it (green, (3,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



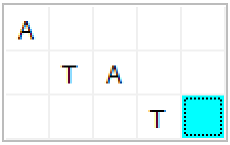
Generating a sentence containing a class generator

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.

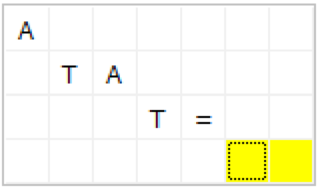
Let us try to use the sentence editor to generate the following sentence:

$$\mathbf{A(X)\ A(Y)\ X \cap Y = \{\ u \mid u \in X \land u \in Y\}}$$

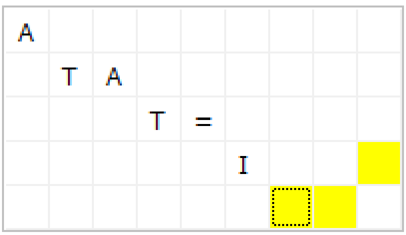
First we insert two universal quantifiers with no condition:



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:

A						+		
	T	A					+	
			T	=				
					I			
						^	^	

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

A						+			
	T	A					+		
			T	=					
					I			K	
						^	^		

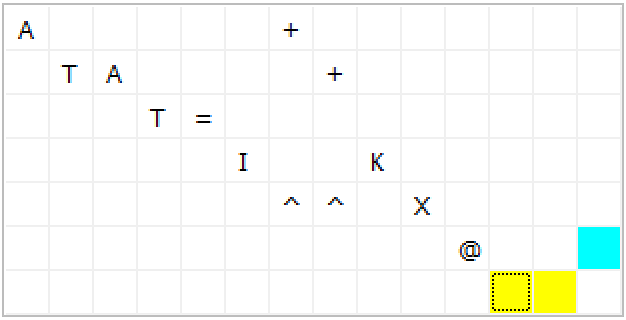
As the argument of “K” has a blue background, we have to insert a form here. The form, which we want to insert, is

$\mathbf{u} \in \mathbf{X} \wedge \mathbf{u} \in \mathbf{Y}$

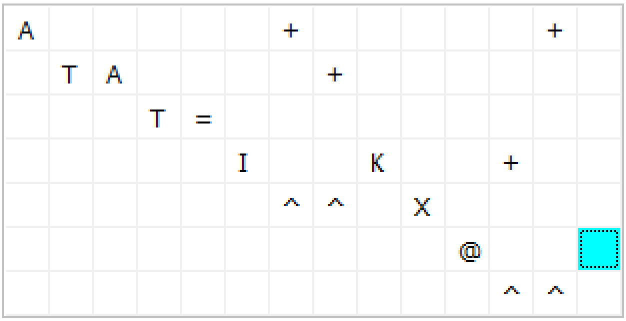
First, we insert the conjunction:

A						+					
	T	A					+				
			T	=							
					I			K			
					n	^	^		X		
					t						

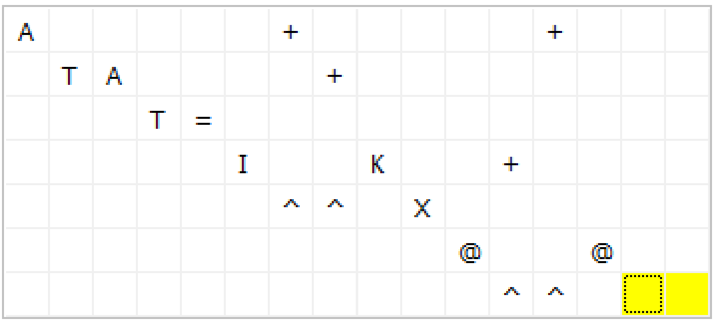
The first argument of the conjunction is a belonging:



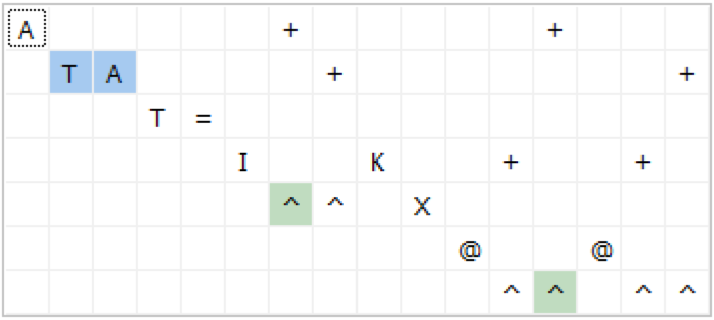
Instead of the variables X and Y, we use links to the first and the second quantifier, but what to do with the variable u? The answer is quite simple: we use a link to the class generator! By clicking on the cell (12, 4), we link the first argument of the belonging to the class generator, and by clicking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

A				+							+	
	E				+	A		+				+
		T	@				S		S			
				^	^			^		D		
										b	^	^
										l		

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:

A				+							+	
	E				+	A		+				+
		T	@				S		S			
				^	^			^		D		
										b	^	^
										l		

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

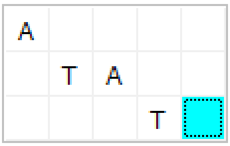
In our system, mathematical definitions are made by creating new logical operators (user-defined logical operators), which are (primitive) relations or (primitive) functions. This can be done easily using the sentence editor. For this, we must first insert into the sentence editor the prefix of a new sentence, containing only universal quantifiers. Then we will insert a double implication “D” for a new (primitive) relation or an equality “=” for a new (primitive) function, and we will press the “Define” button. The sentence editor asks us for the name of the new (primitive) relation or function 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.

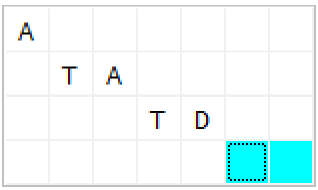
For example, let us define the new (primitive) relation XYZ like this:

A(x) A(y) (XYZ(x,y) ↔ ...)

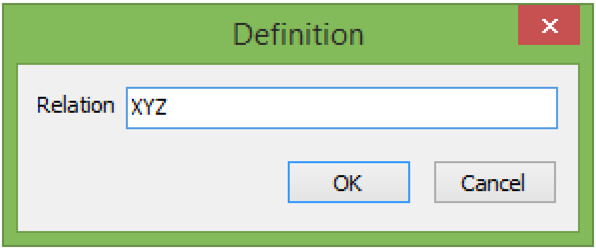
We first insert the prefix into the sentence editor:



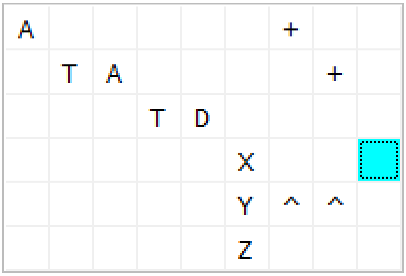
Now, we insert the double implication:



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:

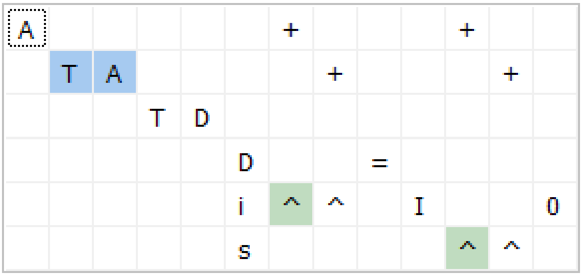
Definition

*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. New (primitive) relations become the kind “R” in the database. New (primitive) functions become the kind “F”. 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

We have defined a **sentence** as a complete composite operator (of logical operators) whose root is a logical operator with a form as a result.

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 equal to 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 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.

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

Matrix:

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

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:

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

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 depend on the names of variables.
- 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 “→” and “∧” 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)
```

Proofs

After generating a sentence in the sentence editor, if we think that it can become a theorem, we can try to prove it in the proof editor. The proof consists of inference operator calls. Such a call is also named a step of the proof. The call of an inference operator is actually the action of using a logical principle in order to prove the theorem. There are three ways to generate a step of a proof:

- by selecting an axiom, theorem, or definition from the list of sentences (using the pseudo-inference operator "Load a sentence").
- by selecting an inference operator from the list of inference operators.
- by selecting an inference operator from the list of formulas.

For each step of the proof, the system shows us the result of the inference operator, which is a sentence generated by the operator. We study the result of the inference operator, and, depending on it, we call the next inference operator. Our goal is to call an inference operator with the sentence to prove as a result, or, in the case of a “reductio ad absurdum” proof, to call an inference operator with a contradiction as a result. Finally, we call the pseudo-inference operator “Verifying the proof”. It changes the “Kind” field of the sentence in the database from “U” to “T”. Our sentence is now a theorem! If we want to make a “reductio ad absurdum” proof, the first operator we call is “Reductio ad absurdum”. The result is the negation of our sentence.

To understand the action of the inference operators, imagine that they are really implementations of inference rules. We tried to make them powerful and easy to use. Many inference operators are based on simple logical formulas. We have grouped them under the name “Formulas”. It is possible to add new formulas to this list.

As we said, a proof of a theorem is a list of inference operator calls (named steps of the proof). When we have selected a step of the proof, in the matrix below we see the sentence that is the result of the selected step. We can see and select all the logical operators in this sentence.

In the chapter dedicated to inference operators, we will find descriptions of the arguments of each operator. The inference operators on the "Formulas" page always need as arguments a step of the proof and a form from the result of that step.

In order to create a new step in a proof, we have to:

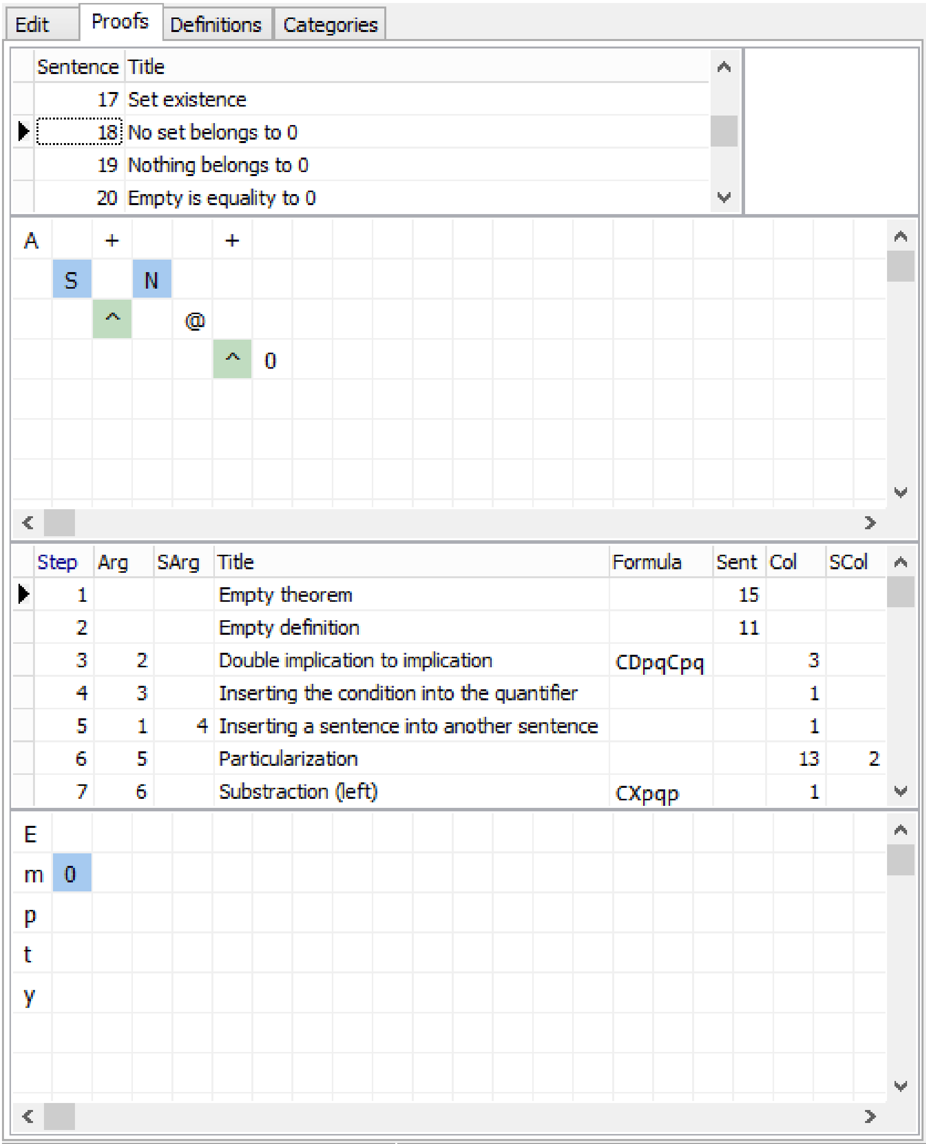
- Decide which inference operator we want to use.

- Select from the proof the necessary steps as arguments for the inference operator we decided to use.
- Select from the result of the last selected step the necessary logical operators as arguments for the inference operator we decided to use.
- Call from the "Inferences" or "Formulas" page the inference operator we decided to use.

If all the necessary conditions are met, a new proof step is created. If not, we receive an error message. One step of a proof is a call of an inference operator, but the choice of this operator and of its arguments can also be seen as a brief description of our intention.

A proof of a theorem made in our system 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.

An example of a proof



The picture above shows the left half of the program's main window, positioned on the "Proofs" tab (proof module). The first table is positioned on the theorem 18.

Under it, we can see the matrix form of this theorem. The next table is named "Steps" and contains the proof of the theorem. Each record in this table is a step of the proof. They are automatically generated by the system at the call of an inference operator. Before calling an inference operator, the user must make various selections; that is, tell the system what he wants to do. This information will form the content of the fields of the new step. Any inference operator also generates a sentence as a result, which can be seen in matrix form under the table "Steps".
The user cannot change the content of the fields of a step. If the proof is not yet finished, so the sentence still has Kind = 'U', then the user can delete the last step of the proof or delete the entire proof.

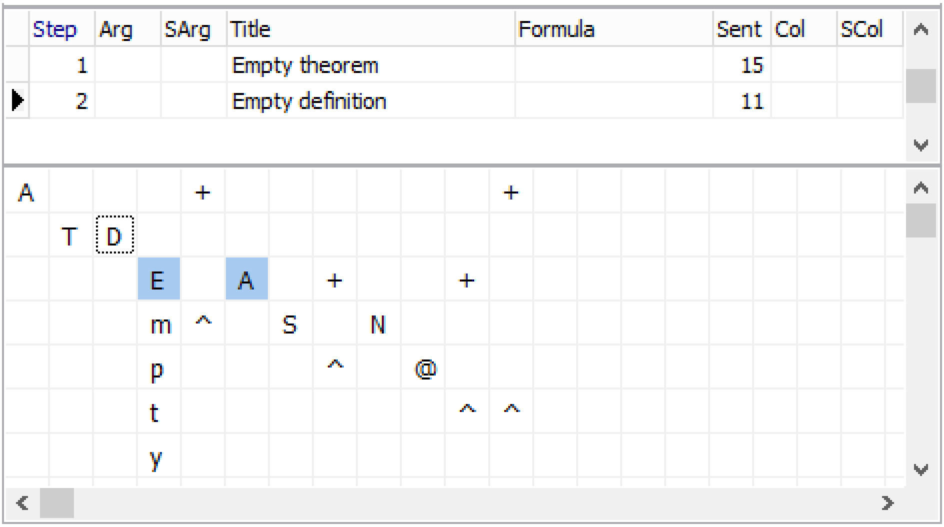
To understand how to prove a theorem in the "Logic" system, we will assume that the sentence at position 18 is not yet a theorem, that is, it has Kind = "U", and the "Steps" table is empty. At the beginning, we want to introduce theorem 15 (Empty theorem) and definition 11 (Empty definition) into the proof. For this, we position the "Sentences" table on theorem 15 and double-click on it. We do the same with definition 11.

Sentences | Inferences | Formulas | I | •

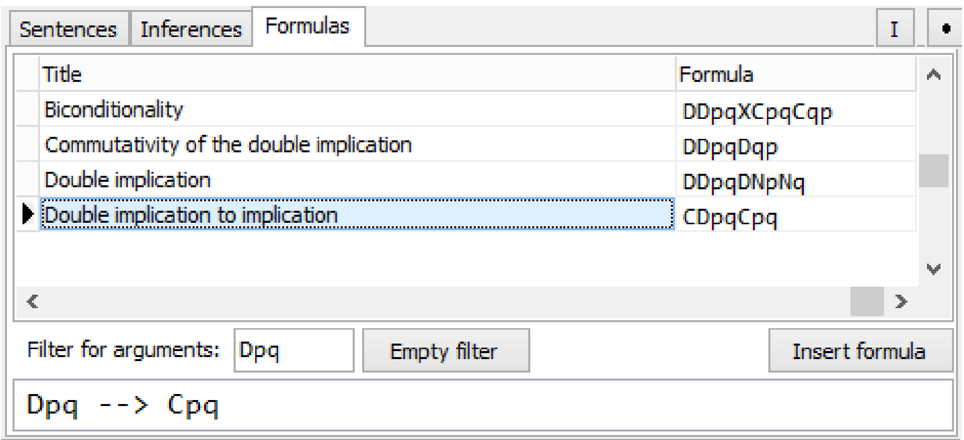
☑ Axiom ☑ Theorem ☑ Relation ☑ Function ☑ Unproved ☐ Aux

Sentence	Title	Kind	Description
1	Axiom of set	A	
2	Belonging implies set condition	T	
3	Belonging is set condition and belonging	T	See 1234 for Nat.
4	Set condition and equality	T	
5	Axiom of extensionality	A	The usual form of the Extensionality axiom.
6	Extensionality corollary	T	See 572 for included classes.
7	Extensionality corollary	T	
8	Equality is reflexive	T	
9	Equality is transitive	T	
10	Equality is symmetric	T	
11	Empty definition	R	Definition of an empty class.
12	Empty class	F	0 is the empty class (set, see 16).
13	Empty corollary	T	
14	Empty uniqueness	T	
15	Empty theorem	T	
16	Axiom of empty set	A	0 is a set.

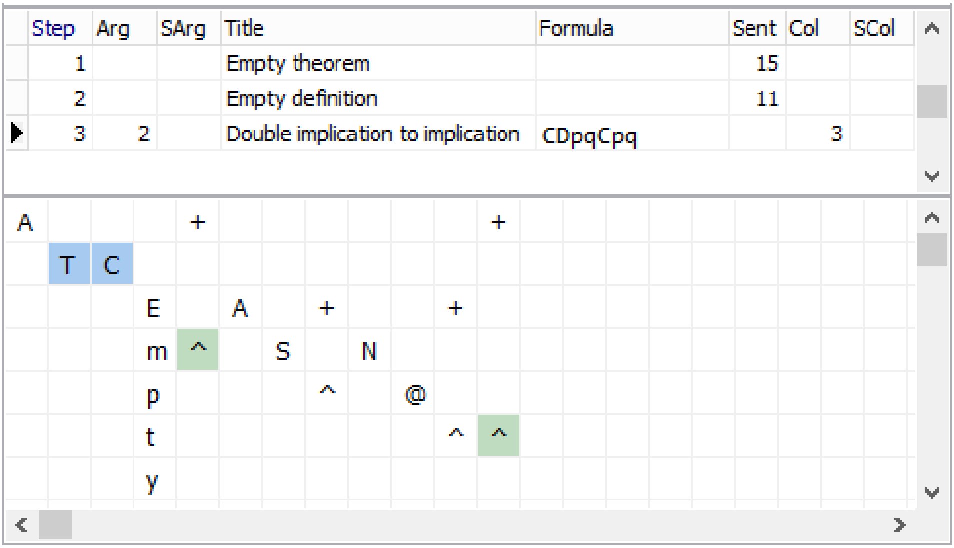
Both sentences will appear in the "Steps" table for the proof of sentence 18:



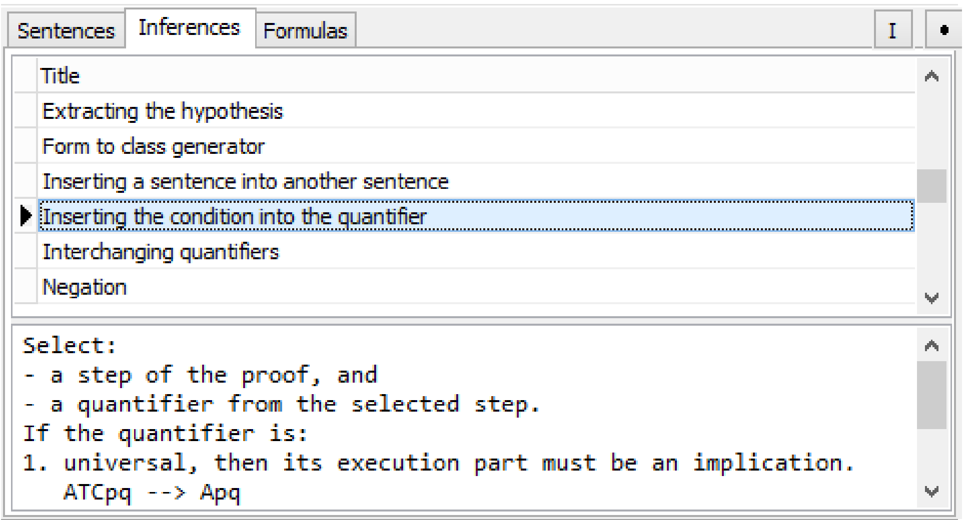
In the matrix form of definition 11, we have positioned the cursor on the double implication ("D" at position 3). We want to transform this double implication ("D") into an implication ("C"). For this, we open the "Formulas" page and look for the "Double implication to implication" formula:



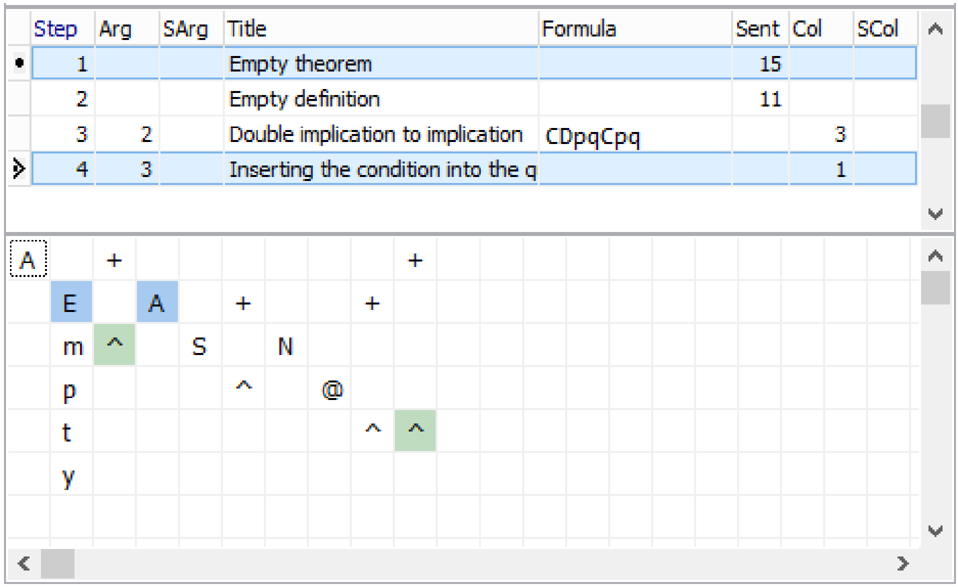
After double-clicking on this formula, we get the third step of the proof:



In the result of the third step, the cursor is positioned on the first quantifier. On the "Inferences" page, we double-click on the "Inserting the condition into the quantifier" operator:



Thus we obtain the fourth step of the proof. By holding down the Ctrl key, we now select step 1 (source step) and step 4 (destination step) of the proof. In the result of step 4, we position the cursor on the first logical operator, i.e., on the universal quantifier "A" (destination):



Step	Arg	SArg	Title	Formula	Sent	Col	SCol
1			Empty theorem		15		
2			Empty definition		11		
3	2		Double implication to implication	CDpqCpq		3	
4	3		Inserting the condition into the q			1	

A		+							+
	E		A		+			+	
	m	^		S		N			
	p				^		@		
	t							^	^
	y								

If we now call the inference operator "Inserting a sentence into another sentence", we get:

Step	Arg	SArg	Title	Formula	Sent	Col	SCol
1			Empty theorem		15		
2			Empty definition		11		
3	2		Double implication to implication	CDpqCpq		3	
4	3		Inserting the condition into the q			1	
5	1	4	Inserting a sentence into anothe			1	

X												
	A		+							+	E	
		E		A		+			+		m	0
		m	^		S		N				p	
		p				^		@			t	
		t							^	^	y	
		y										

We now want to particularize the universal quantifier at position 2 ("A") using the logical operator at position 13 ("0"). For this, we first click on "0" (position 13, the source) and then on "A" (position 2, the destination), after which, by double-clicking, we call the inference operator "Particularization" from the "Inferences" page:

Step	Arg	SArg	Title	Formula	Sent	Col	SCol
1			Empty theorem		15		
2			Empty definition		11		
3	2		Double implication to implication	CDpqCpq		3	
4	3		Inserting the condition into the q			1	
5	1	4	Inserting a sentence into anothe			1	
6	5		Particularization			13	2

X									
	A		+			+		E	
		S		N				m	0
			^		@			p	
						^	0	t	
								y	

The result of step 6 of the proof is a conjunction ("X" in position 1). We select here the position 1, and then, by double-clicking, we call the “Subtraction (left)” formula from the "Formulas" page:

Sentences | Inferences | Formulas | I | •

Rule	Title	Formula
101	Commutativity of the conjunction	DXpqXqp
158	Conjunction implies disjunction	CXpqVpq
143	Conjunction implies implication	CXpqCpq
126	Conjunction to double implication	CXpqDpq
185	Equivalence of modus ponens (reverse)	DXpqXpCpq
119	Subtraction (left)	CXpqp
152	Subtraction (right)	CXpqq

Filter for arguments: Xpq | Empty filter | Insert formula

Xpq <--> Xqp

As we expected, the result of step 7 of the proof is:

A		+			+	
	S		N			
		^		@		
					^	0

We have obtained exactly the sentence we want to prove. The demonstration is ready. We only have to call the (pseudo) inference operator "Verifying the proof", which turns the sentence into a theorem, i.e., changes the value of the field "Kind" from "U" to "T".

In the top table on the "Proofs" page, at the level of each sentence, we can call the menu items:

- Save proof
  It is used if we want to send the proof of the theorem to a person who uses this system.
- Save the sentences of the proof
  It is used if we want to send the proof of the theorem to a person who does not use this system.
- Delete proof
- Load proof
  It is used if we want to import the proof of a theorem made by someone using this system.

In the following, we will analyze the text file produced by "Save the sentences of the proof" for theorem 18 (No set belongs to 0).

For each step of the proof, this function will generate a line containing the information from that step, followed by a ruler (which helps us see the positions of the logical operators) and by the result of the inference operator as text.

First, we will show only the lines of the file containing the information from the proof steps:

```
1. Empty theorem (Sentence: 15)
2. Empty definition (Sentence: 11)
3. CDpqCpq (Step: 2, Col: 3)
4. Inserting the condition into the quantifier (Step: 3, Col: 1)
5. Inserting a sentence into another sentence (Step: 1, SStep: 4, Col: 1)
6. Particularization (Step: 5, Col: 13, SCol: 2)
7. CXpqp (Step: 6, Col: 1)
```

At the first position we have the step number, followed by the content of the field "Title" or the field "Formula".

To present the values of the other defining fields from the "Steps" table, we have used other names for these fields, as can be seen in the table below:

```
Arg = Step
SArg = SStep (second step)
Sent = Sentence

Col (column)
SCol (second column)
```

The information provided by the steps of a proof can be used to understand the proof. It can be seen as a brief description of our intention. How can we use it for this? We have the following distinct cases:

1. The proof step does not contain any parameters. In this case, the proof step represents the call of "Extracting the hypothesis", "Tautologize", or "Reductio ad absurdum".
2. The proof step contains only the parameter "Sentence". In this case, the proof step represents the loading of the sentence indicated by the parameter "Sentence" into the proof.
3. The proof step contains the parameters Step (source step) and Col (destination).
4. The proof step contains the parameters Step (source step), Col (source), and SCol (destination).
5. The proof step contains the parameters Step (source step), SStep (destination step), and Col (destination).

Here is the entire file, marked with comments necessary for understanding:

The first step is the loading of theorem 15 into the proof:

1. Empty class theorem (Sentence: 15)

```
1 2
E
m 0
p
t
y
```

The second step is the loading of definition 11 into the proof:

2. Empty class definition (Sentence: 11)

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2
A - - - + - - - - - - +
  T D
      E   A - + - - + -
      m ^   S   N
      p       ^   @
      t             ^ ^
      y
```

Step 3 of the proof is performed by applying the "CDpqCpq" formula from the "Formulas" page for step 2 (source step) and for position 3 (destination):

3. CDpqCpq (Step: 2, Col: 3)

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2
A - - - + - - - - - - +
  T C
      E   A - + - - + -
      m ^   S   N
      p       ^   @
      t             ^ ^
      y
```

Step 4 of the proof is performed by applying the inference operator "Inserting the condition into the quantifier" from the "Inferences" page for step 3 (source step) and position 1 (destination):

4. Inserting the condition into the quantifier (Step: 3, Col: 1)

```
                  1
1 2 3 4 5 6 7 8 9 0
A - + - - - - - - +
  E   A - + - - + -
  m ^   S   N
  p       ^   @
  t             ^ ^
  y
```

Step 5 of the proof is performed by applying the inference operator "Inserting a sentence into another sentence" from the "Inferences" page for step 1 (source step) and step 4 (destination step) to position 1 (destination):

5. Inserting a sentence into another sentence (Step: 1, SStep: 4, Col: 1)

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2 3
X
  A - + - - - - - - + E
    E   A - + - - + - m 0
    m ^   S   N       p
    p       ^   @     t
    t             ^ ^ y
    y
```

Step 6 of the demonstration is performed by applying the inference operator "Particularization" from the "Inferences" page for step 5 (the source step), positions 13 (source) and 2 (destination):

6. Particularization (Step: 5, Col: 13, SCol: 2)

```
                  1
1 2 3 4 5 6 7 8 9 0
X
  A - + - - + - E
    S   N       m 0
      ^   @     p
            ^ 0 t
                y
```

Step 7 of the proof is performed by applying the "CXpqp" formula from the "Formulas" page for step 6 (source step) and position 1 (destination).

7. CXpqp (Step: 6, Col: 1)

```
1 2 3 4 5 6 7
A - + - - + -
  S   N
    ^   @
          ^ 0
```

The logic of a proof is not fundamentally different from the usual proof of a theorem. But, since there are no more obvious steps or steps left to the reader, every step of a proof must be perfectly justified. To make the proof of a theorem as clear as possible, we recommend proving some helpful theorems in advance, which we will use in the proof of our theorem.

Defining a logical operator using an existence and a uniqueness theorem

In a previous chapter, we showed how to directly define a new (primitive) function. But there is another very important method to define a new (primitive) function, namely by using an existence theorem and a uniqueness theorem for a certain property.

Let us suppose that we have proved the following existence theorem:

```
A - + - - - - + - - - - - - - - - +
^ F   A - + - - - - - - - - - + - -
F u ^ ^ @       E - - - + - - - + -
  n F x   ^ D   ^ T X
  c       x o ^ y     S   @
            m F         ^   P     ^
                        y   a ^ ^ F
                            i x y
                            r
```

`A(F|Func(F)) A(x|@(x,Dom(F))) E(y) X(S(y),@(Pair(x,y),F))`

“Func” is a graph function, “Dom” is the domain of a graph, and “Pair” is an ordered pair.

*Let **F** be a graph function. For any **x** in the domain of **F**, there exists a set **y** such that the ordered pair of **x** and **y** belongs to F.*

We have also proved the uniqueness theorem for the same property:

```
A - + - - - - + - - - - - - - - + - - - - - - - - + - - -
^ F   A - + - - - - - - - - + - - - - - - - - + - - - - -
F u ^ ^ @       A - - + - - - + - - - - - - - - - - - + -
  n F x   ^ D   ^ X               A - - + - - - + - - - +
  c       x o ^ y   S   @         ^ X               =
            m F 1     ^   P     ^ y   S   @           ^ ^
                      y   a ^ ^ F 2     ^   P     ^   y y
                      1   i x y         y   a ^ ^ F   1 2
                          r   1         2   i x y
                                            r   2
```

```
A(F|Func(F)) A(x|@(x,Dom(F))) A(y1|X(S(y1),@(Pair(x,y1),F)))
A(y2|X(S(y2),@(Pair(x,y2),F))) =(y1,y2)
```

*Let **F** be a graph function. For any **x** in the domain of **F**, and for any set **y1** with the ordered pair of **x** and **y1** belonging to **F**, and for any set **y2** with the ordered pair of **x** and **y2** belonging to **F**, **y1** = **y2**.*

It should be noted that the execution part (the second argument) of the existential quantifier from the existence theorem coincides with the condition parts (the first arguments) of the last two quantifiers from the uniqueness theorem.

In order to define a new (primitive) function based on these two theorems, we must first generate a new sentence in the sentence editor. This sentence only contains the “T” operator. After saving the sentence with the kind “U”, we insert the existence theorem and the uniqueness theorem into the proof of our new sentence. Finally, we call the inference operator “Verifying the proof”. This operator proves that our sentence contains only “T” and verifies whether the first sentence of the proof is an existence theorem and the second a uniqueness theorem (for the same condition!). If so, the operator “Verifying the proof” requires a name for the new function. Let us suppose that we named the new (primitive) function “Val”. The inference operator “Verifying the proof” also changes the content of the sentence ("T") into a sentence created from the existence theorem by making two changes. First, it replaces all the links to the existential quantifier with the newly defined logical operator ("Val"), followed by links to all universal quantifiers in the prefix. Then, it replaces the existential quantifier with its second argument. This is the new sentence:

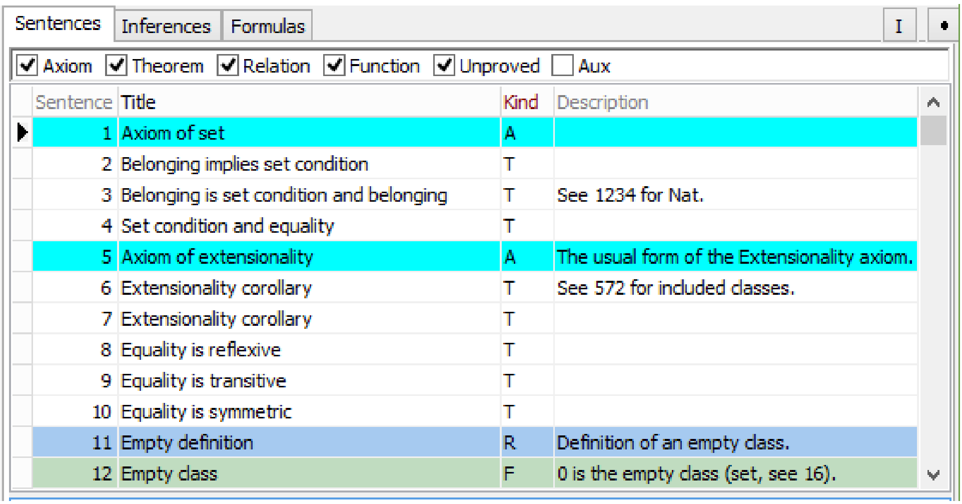
```
A - + - - - - + - - - + - - - - - + - +
^ F   A - + - - - - - - + - - + - - + -
F u ^ ^ @       X
  n F x   ^ D     S       @
  c       x o ^     V       P         ^
            m F     a ^ ^   a ^ V     F
                    l F x   i x a ^ ^
                            r   l F x
```

A(F|Func(F)) A(x|@(x,Dom(F))) X(S(Val(F,x)),@(Pair(x,Val(F,x)),F))

*Let **F** be a graph function. For each **x** in the domain of **F**, Val(**F**, **x**) is a set, and the ordered pair of **x** and Val(**F**, **x**) belongs to **F**.*

Loading sentences into the proof

In order to load a sentence into a proof, simply double-click the sentence in the “Sentences” table. It is not permitted to use sentences with the kind “U”. For example, if we want to introduce the extensionality axiom in the proof of a theorem, we double-click in the "Sentences" table on record number 5.



Sentence	Title	Kind	Description
1	Axiom of set	A	
2	Belonging implies set condition	T	
3	Belonging is set condition and belonging	T	See 1234 for Nat.
4	Set condition and equality	T	
5	Axiom of extensionality	A	The usual form of the Extensionality axiom.
6	Extensionality corollary	T	See 572 for included classes.
7	Extensionality corollary	T	
8	Equality is reflexive	T	
9	Equality is transitive	T	
10	Equality is symmetric	T	
11	Empty definition	R	Definition of an empty class.
12	Empty class	F	0 is the empty class (set, see 16).

This will create a new step of the proof with the result:

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

Inference operators

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

Duplicability

Let ***p***, ***q*** be forms of a sentence. 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.

Let ***p*** and ***q*** be forms of a sentence and ***o*** their first common ancestor. Let's consider the following 3 cases:

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

If one of the above cases is met, then, by replacing ***q*** by the conjunction or implication of ***p*** and ***q*** (X**pq**, C**pq**) in our sentence, we obtain an equivalent sentence. We say that ***p*** is duplicable on ***q*** (***q*** is in the influence zone of ***p***).

The principle of duplicability is justified by the following tautologies, where p, q, r are sentences, and M represents one of the logical operators X, V, D, or C.

To simplify the formulas, we will represent the double implication (D) by " ↔ ".

```
X(p,q) ↔ X(p,X(p,q))
X(p,q) ↔ X(p,C(p,q))
C(p,q) ↔ C(p,X(p,q))
C(p,q) ↔ C(p,C(p,q))

D(X(p,M(q,r))) ↔ X(p,M(X(p,q),r))
D(X(p,M(q,r))) ↔ X(p,M(C(p,q),r))
D(C(p,M(q,r))) ↔ C(p,M(X(p,q),r))
D(C(p,M(q,r))) ↔ C(p,M(C(p,q),r))

X(p,N(q)) ↔ X(p,N(X(p,q)))
X(p,N(q)) ↔ X(p,N(C(p,q)))
C(p,N(q)) ↔ C(p,N(X(p,q)))
C(p,N(q)) ↔ C(p,N(C(p,q)))

X(X(p,q),M(r,s)) ↔ X(X(p,q),M(X(p,r),s))
X(X(p,q),M(r,s)) ↔ X(X(p,q),M(C(p,r),s))
C(X(p,q),M(r,s)) ↔ C(X(p,q),M(X(p,r),s))
C(X(p,q),M(r,s)) ↔ C(X(p,q),M(C(p,r),s))

X(X(p,q),N(r)) ↔ X(X(p,q),N(X(p,r)))
X(X(p,q),N(r)) ↔ X(X(p,q),N(C(p,r)))
C(X(p,q),N(r)) ↔ C(X(p,q),N(X(p,r)))
C(X(p,q),N(r)) ↔ C(X(p,q),N(C(p,r)))
```

In the following four formulas, p(x) and q(x) represent two predicates of variable x:

```
A(x|p(x)) q(x) ↔ A(x|p(x)) X(p(x),q(x))
A(x|p(x)) q(x) ↔ A(x|p(x)) C(p(x),q(x))
E(x|p(x)) q(x) ↔ E(x|p(x)) X(p(x),q(x))
E(x|p(x)) q(x) ↔ E(x|p(x)) C(p(x),q(x))
```

In the following formulas, p does not contain the free variable x, and q(x) and r(x) represent two predicates of variable x:

```
X(p,A(x|q(x)) r(x)) ↔ X(p,A(x|X(p,q(x))) r(x))
X(p,A(x|q(x)) r(x)) ↔ X(p,A(x|C(p,q(x))) r(x))
C(p,A(x|q(x)) r(x)) ↔ C(p,A(x|X(p,q(x))) r(x))
C(p,A(x|q(x)) r(x)) ↔ C(p,A(x|C(p,q(x))) r(x))

X(p,E(x|q(x)) r(x)) ↔ X(p,E(x|X(p,q(x))) r(x))
X(p,E(x|q(x)) r(x)) ↔ X(p,E(x|C(p,q(x))) r(x))
C(p,E(x|q(x)) r(x)) ↔ C(p,E(x|X(p,q(x))) r(x))
C(p,E(x|q(x)) r(x)) ↔ C(p,E(x|C(p,q(x))) r(x))
```

```
X(p,A(x|q(x)) r(x)) ↔ X(p,A(x|q(x)) X(p,r(x)))
X(p,A(x|q(x)) r(x)) ↔ X(p,A(x|q(x)) C(p,r(x)))
C(p,A(x|q(x)) r(x)) ↔ C(p,A(x|q(x)) X(p,r(x)))
C(p,A(x|q(x)) r(x)) ↔ C(p,A(x|q(x)) C(p,r(x)))

X(p,E(x|q(x)) r(x)) ↔ X(p,E(x|q(x)) X(p,r(x)))
X(p,E(x|q(x)) r(x)) ↔ X(p,E(x|q(x)) C(p,r(x)))
C(p,E(x|q(x)) r(x)) ↔ C(p,E(x|q(x)) X(p,r(x)))
C(p,E(x|q(x)) r(x)) ↔ C(p,E(x|q(x)) C(p,r(x)))
```

Implicability

Let ***p*** be a form of a sentence. We say that ***p*** is implicable if the result of replacing ***p*** with a form implied by ***p*** is an inference.

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 has an implicable parent ”E”, ”X”, or ”V”.
- The form has an implicable parent ”A” or ”C”, and the form is the second argument of its parent.

In the following, when describing inference operators, we will use some simplified expressions.

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.

*We have seen that a sentence is a tree with logical operators as nodes. If **p** is a logical operator from a sentence, by the **content** of **p** we mean the whole branch of the tree starting from **p** (inclusive). That is, by the content of **p**, we understand the operator, its arguments, the arguments of its arguments, and so on.*

*In the description of inference operators, when comparing, replacing, or generating the content of logical operators, the notion of content of a logical operator will be implicit. For example, if **p** and **q** are two logical operators of a sentence, instead of saying that the content of the first argument of **p** is equal to the content of **q**, we simply say that the first argument of **p** is equal to **q**.*

*In the following example, let **p** be the logical operator at position 8. According to the convention above, we can say that the first argument of **p** is different from the* *second argument of **p**, even though they are both "=", because they have different arguments.*

A									+						+	
	T	A								+		+				
			T	A									+			+
					T	C										
							X							=		
								=			=				^	^
									^	^		^	^			

Classification

Processing the hypothesis

These inference operators generate a step of the proof, whose result is based on the sentence we want to prove.

- Extracting the hypothesis
- Tautologize
- Reductio ad absurdum

One step processing

To use these inference operators, we must select a step of the proof and one or two logical operators of the step.

- Negation
- Class generator to form
- Form to class generator
- Attachment
- Replacement
- Duplication (conjunction)
- Duplication (implication)
- Existential generalization (using equality)
- Universal generalization (using equality)
- Extracting the condition of the quantifier
- Inserting the condition into the quantifier

- Deleting a quantifier
- Dual quantifier distributivity
- Partial quantifier distributivity
- Restricted quantifier distributivity
- Quantifier distributivity
- Interchanging quantifiers
- Particularization
- Commutativity
- Reflexivity

Two steps processing

To use these inference operators, we must select two steps of the proof and one logical operator of the second step.

- Cross attachment
- Cross replacement
- Inserting a sentence into another sentence

Extracting the hypothesis

Let us suppose we want to prove a sentence. The whole sentence cannot be loaded into the proof, but we could need the hypothesis of the sentence for the proof. What should we do? One possibility is the use of the (pseudo) inference operator "Extracting the hypothesis".

If the sentence we want to prove has the form:

– <u-prefix>Cpq, then the result is <u-prefix>CpT.
– <u-prefix>Dpq, then the result is <u-prefix>XCpTCqT.
– In any other case, the result is <u-prefix>T.

Let us suppose we want to prove the sentence:

A		+						+				+
	S		A			+			+		+	
		^		X						X		
					S		@			Y	^	^
						^		^	^	Z		

At the beginning, the proof is empty. Using “Extracting the hypothesis” we obtain:

A		+						+		
	S		A			+			+	
		^		X						T
					S		@			
						^		^	^	

This is true and contains the hypothesis of our sentence. We can use it for other inference operators to finish the proof.

Tautologize

This inference operator generates a proof step based on the sentence we want to prove. The result of this step contains all the logical operators of the sentence in a tautologized form.

- If the sentence has the form <u-prefix>CpDqr, then the result is <u-prefix>CpXDqqDrr.
- If the sentence has the form <u-prefix>Cpq, where q does not start with "D", then the result is <u-prefix>CpDqq.
- If the sentence has the form <u-prefix>p, where p does not start with "C", then the result is <u-prefix>Dpp.

Example:

A		+						+				+
	S		A			+			+		+	
		^		X						X		
					S		@			Y	^	^
						^		^	^	Z		

Using “Tautologize”, we obtain a step of the proof with the following result:

A		+						+					+			+
	S		A			+			+			+			+	
		^		X						D						
					S		@				X			X		
						^		^	^		Y	^	^	Y	^	^
											Z			Z		

Using “Extracting the hypothesis” instead of "Tautologize", we lose the operator XYZ:

A		+						+		
	S		A			+			+	
		^		X						T
					S		@			
						^		^	^	

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:

A		+						+				+
	S		A			+			+		+	
		^		X						X		
					S		@			Y	^	^
						^		^	^	Z		

The result of the inference operator is:

E		+						+					+
	S		E			+			+			+	
		^		X						N			
					S		@				X		
						^		^	^		Y	^	^
											Z		

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

N								
	A		+				+	
		S		E				+
			^		T	@		
							^	^

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

E		+					+	
	S		A					+
		^		T	N			
						@		
							^	^

The action of some inference operators is explained by a logical ***transformation formula****. Let us consider the formula:*
AcCpq $\rightarrow$ *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”)
```

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:

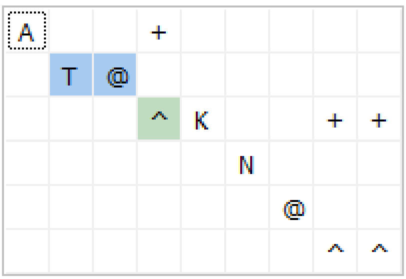
- The selected form is a belonging ("@").
- The second argument of this belonging is a class generator ("K").

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:

A				+			+	+
	T	X						
			S		N			
				^		@		
							^	^

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

A				+			+	+
	T	X						
			S		N			
				^		@		
							^	^

By using “Form to class generator” we obtain:

A			+				+	+
	T	@						
			^	K				
					N			
						@		
							^	^

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 source and the destination are equal.

If the parent of the destination is a conjunction and the other argument of this conjunction is equal to the second argument of the parent of the source, the function will replace the conjunction with the destination. Otherwise, the function will replace the destination with a conjunction formed by the destination and the second argument of the parent of the source.

As you can see, 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):

```
- - - 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
```

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

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

the result of the inference operator is:

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

A							+			+					+			+	
	T	A						+								+			
			T	X															
					C						E								+
						@			S			T	X						
							^	^		^				@			X		
															^	^	Y	^	^
																	Z		

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:

A							+			+						+			+		+	
	T	A						+									+					
			T	X																		
					C						E											+
						@			S			T	X									
							^	^		^				X						X		
															@			S		Y	^	^
																^	^		^	Z		

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 source and 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 destination with the other argument of the parent of the source.

Let us consider that our proof contains the following sentence:

A					+				+				+		+	+
	T	X														
			D								C					
				S		E				+		S		X		
					^		T	@					^	Y	^	^
									^	^				Z		

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:

A					+				+						+			+	+
	T	X																	
			D								C								
				S		E				+		E				+	X		
					^		T	@					T	@			Y	^	^
									^	^					^	^	Z		

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.

If the source and the destination have different contents, the operator replaces the destination with the conjunction of the source and the destination.

If the source and the destination have equal contents and the parent of the destination is a conjunction, the operator replaces this conjunction with 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
```

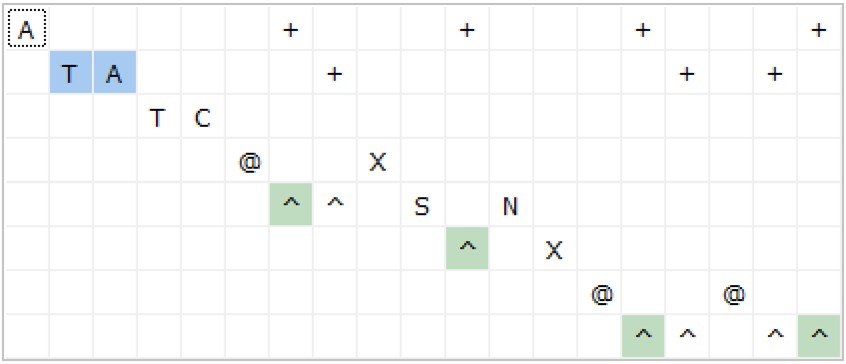
the result of the inference operator is:

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

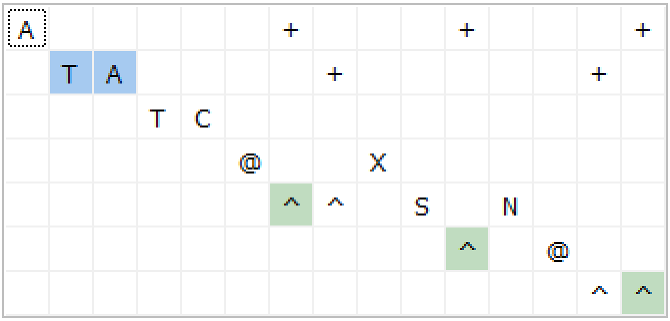
Example

A						+				+				+
	T	A					+						+	
			T	C										
					@			X						
						^	^		S		N			
										^		@		
													^	^

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)”.

If the source and the destination have different contents, the operator replaces the destination with the implication of the source and the destination.

If the source and the destination have equal contents and the destination is the first argument of an implication, the operator replaces this implication with 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 - - -

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

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

s d

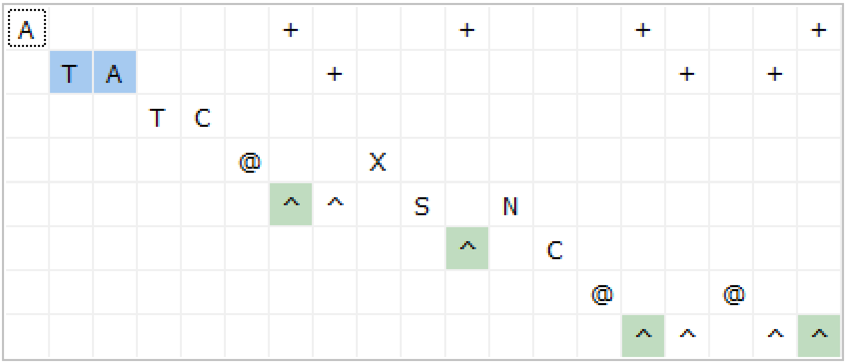
the result of the inference operator is:

- - - p p p - - - q q q q - - -

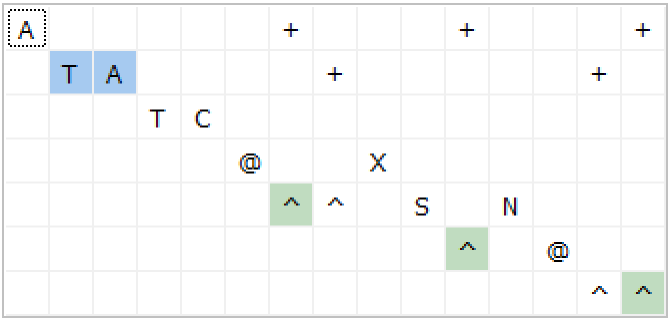
Example

A						+				+				+
	T	A					+						+	
			T	C										
					@			X						
						^	^		S		N			
										^		@		
													^	^

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)

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 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 selected term. The execution (second argument) of the quantifier is the selected form, where each occurrence 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.

This inference operator is justified by the following formulas, where p(x, y, z) is a predicate of variables x, y, z, and f(x, y) is a term:

```
A(x) E(y) x = y
p(x,y,f(x,y)) ↔ E(z|z=f(x,y)) p(x,y,z)
p(x,y,f(x,y)) ↔ A(z|z=f(x,y)) p(x,y,z)
```

Example

A					+		+	
	T	A						+
			T	P				
				a	^	U		
				r			^	^
				t				

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

A								+			+	
	T	A							+			
			T	E		+						+
					=					P		
						^	U			a	^	^
								^	^	r		
										t		

Extracting the condition of the quantifier

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 condition, or, if the condition is a conjunction, the second argument of the condition, into 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

Conditional quantifiers are defined by:

$$A(x|p(x))\ q(x) \leftrightarrow A(x)\ C(p(x),q(x))$$
$$E(x|p(x))\ q(x) \leftrightarrow E(x)\ X(p(x),q(x))$$

Example for a universal quantifier:

A		+				+	
	S		E				+
		^		T	@		
						^	^

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

A				+				+	
	T	C							
			S		E				+
				^		T	@		
								^	^

Example for an existential quantifier:

E		+				+	
	S		A				+
		^		T	P		
					a	^	^
					r		
					t		

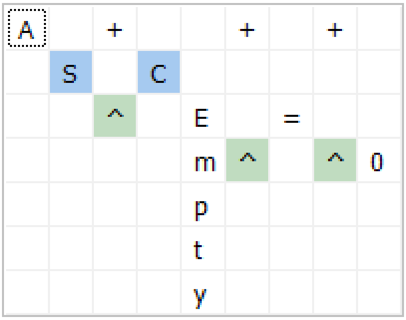
In this case the result is:

E				+				+	
	T	X							
			S		A				+
				^		T	P		
							a	^	^
							r		
							t		

Example of a universal quantifier with a conjunction as condition:

A			+		+		+	
	X					=		
		S		E			^	0
			^	m	^			
				p				
				t				
				y				

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



Inserting the condition into the quantifier

To use this inference operator, select:

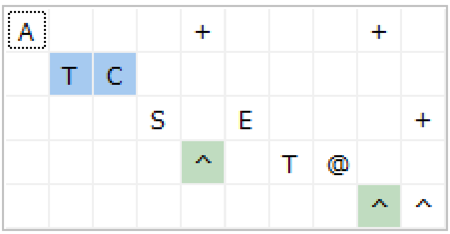
- a step of the proof, and
- a quantifier from the selected step.

If the quantifier is:

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

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

Example for a universal quantifier:



The result is:

A		+				+	
	S		E				+
		^		T	@		
						^	^

Example for an existential quantifier:

E				+				+	
	T	X							
			S		A				+
				^		T	P		
							a	^	^
							r		
							t		

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

E		+				+	
	S		A				+
		^		T	P		
					a	^	^
					r		
					t		

Deleting a quantifier

To use this inference operator, select:

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

This inference operator verifies whether the second argument of the quantifier (the execution argument of the quantifier) has no links to the quantifier (the quantifier is not used by its execution argument).

The result of the operator is the selected step where the selected quantifier is replaced by its execution part.

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

```
q = the selected quantifier
c = the condition part of the quantifier
e = the execution part of the quantifier
```

Q represents the quantifier “A” or “E”

Selected step:

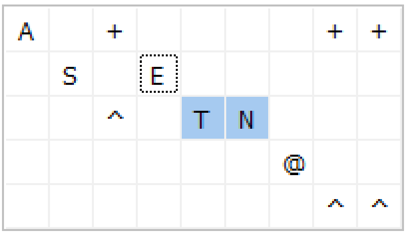
```
- - - Q c c c e e e e - - -

      q c     e
```

The result of the inference operator is:

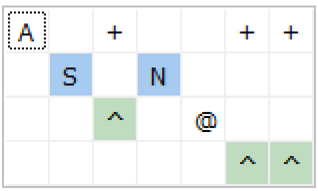
```
- - - e e e e - - -
```

The selected quantifier can be deleted if one of the following conditions is met:

1. The quantifier has the condition “T” (no condition!).



The result is:



2. The condition of the quantifier is an equality having as its first argument a reference (link) to the quantifier and as its second argument no references to the quantifier.

A						+			+
	T	A		+					
			=				P		
				^	R		a	0	^
					a	^	r		
					n		t		

The result is:

A				+
	T	P		
		a	0	^
		r		
		t		

3. The quantifier is existential and implicable.

We can delete the second quantifier of this sentence:

A		+						+	+
	S		E		+				
		^		S		N			
					^		@		
								^	^

The result is:

A		+			+	+
	S		N			
		^		@		
					^	^

4. The quantifier has a condition, and you have loaded in the proof an existence axiom or theorem for this condition.

Let us suppose we want to delete the first quantifier from this sentence of the proof:

A		+							
	S		A		+			+	+
		^		S		N			
					^		@		
								^	^

We can see that the quantifier has the set condition “S”. Before using the inference operator “Deleting a quantifier”, we must load the existence axiom or the theorem for this condition:

E			+
	T	S	
			^

Now, by using “Deleting a quantifier” we get the result:

A		+			+	+
	S		N			
		^		@		
					^	^

Deleting quantifiers from right to left

To use this inference operator, select:

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

This inference operator is a repeated call of the inference operator "Deleting a quantifier". If the selected quantifier can be deleted, then this operator also tries to delete the parent of the selected quantifier, which must be a quantifier. The process is repeated as many times as possible.

Dual quantifier distributivity

To use this inference operator, select:

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

This operator works in the following cases:

1. The destination is a universal quantifier, with its execution part an implication. The second argument of this implication has no references to the quantifier.
   AcCpq → CEcpq
2. The destination is an existential quantifier, with its execution part an implication. The second argument of this implication has no references to the quantifier. If the condition of the quantifier is not “T”, the selected form must be implicable.
   EcCpq → CAcpq
3. The destination is an implication, with its first argument a universal quantifier with no condition.
   CATpq → ETCpq
4. The destination is an implication, with its first argument an existential quantifier.
   CEcpq → AcCpq

The inference operator "Dual quantifier distributivity" is justified by the following formulas, where p(x) and q(x) are predicates by the variable x, and r does not contain the variable x:

```
A(x|p(x)) C(q(x),r) ↔ C(E(x|p(x)) q(x),r)
E(x|p(x)) C(q(x),r) ↔ C(A(x|p(x)) q(x),r)
C(A(x) q(x),r) ↔ E(x) C(q(x),r)
```

In this example, the execution part of the second quantifier is an implication. The second argument of this implication has no links to the quantifier:

A						+			+
	T	$\boxed{A}$					+		
			T	C					
					@			S	
						^	^		^

The result has only one quantifier in the prefix. The prefix-arg is the implication at position 3:

A						+			+
	T	C							
			E				+	S	
				T	@				^
						^	^		

Partial quantifier distributivity

To use this inference operator, select:

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

The destination must be a quantifier, a conjunction, a disjunction, or an implication.

- If the destination is a quantifier, then its execution part must be a conjunction, disjunction, or implication. The first argument of the execution part of the quantifier must be free of references to the quantifier.
  - If the destination is a universal quantifier and its execution part is a conjunction, then the condition of the quantifier must be “T” (no condition).
  - If the destination is an existential quantifier having a condition different from “T” and its execution part is a disjunction or an implication, then the destination must be implicable.

  `QcMpq` → `MpQcq` where Q is ”A” or ”E” (with the condition c) and M is ”X”, ”V”, or ”C”.

- If the destination is a conjunction, disjunction, or implication, then its second argument must be a quantifier.
  - If the destination is a conjunction and its second argument is a universal quantifier having a condition different from “T”, then the destination must be implicable.

- If the destination is a disjunction or an implication and its second argument is an existential quantifier, then the condition of this quantifier must be “T” (no condition).

`MpQcq → QcMpq` where Q is ”A” or ”E” (with the condition c) and M is ”X”, ”V”, or ”C”.

The operator is based on the following formulas, where p(x) and q(x) are predicates of variable x, and r does not contain the variable x:

```
X(r,A(x|p(x)) q(x)) → A(x|p(x)) X(r,q(x))
X(r,A(x) q(x)) ↔ A(x) X(r,q(x))
X(r,E(x|p(x)) q(x)) ↔ E(x|p(x)) X(r,q(x))
V(r,A(x|p(x)) q(x)) ↔ A(x|p(x)) V(r,q(x))
E(x|p(x)) V(r, q(x)) → V(r,E(x|p(x)) q(x))
E(x) V(r,q(x)) ↔ V(r, E(x) q(x))
C(r,A(x|p(x)) q(x)) ↔ A(x|p(x)) C(r,q(x))
E(x|p(x)) C(r,q(x)) → C(r,E(x|p(x)) q(x))
E(x) C(r,q(x)) ↔ C(r,E(x) q(x))
```

Example:

E						+		+	
	T	A							+
			T	X					
					S		@		
						^		^	^

If we select the second quantifier (on position 3) and apply the inference operator, we obtain:

E				+				+	
	T	X							
			S		A				+
				^		T	@		
								^	^

If we now select the conjunction in position 3 and apply the operator again, we obtain:

E						+		+	
	T	A							+
			T	X					
					S		@		
						^		^	^

We can see that “Partial quantifier distributivity” works in both directions.

Restricted quantifier distributivity

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 disjunction or an existential quantifier.
- The selected form is implicable.

If the selected form is:

- A disjunction whose arguments are universal quantifiers having equal conditions, then VAcpAcq → AcVpq.
- An existential quantifier with a conjunction as its execution part, then EcXpq → XEcpEcq.

The operator is based on the following formulas, where c(x), p(x), q(x) are predicates by the variable x:

```
V(A(x|c(x)) p(x),A(x|c(x)) q(x)) → A(x|c(x)) V(p(x),q(x))
E(x|c(x)) X(p(x),q(x)) → X(E(x|c(x)) p(x),E(x|c(x)) q(x))
```

Example for the second case:

E				+		+	+
	T	X					
			E		P		
			m	^	a	^	^
			p		r		
			t		t		
			y				

Applying “Restricted quantifier distributivity” to the quantifier, we obtain:

X									
	E			+	E			+	+
		T	E			T	P		
			m	^			a	^	^
			p				r		
			t				t		
			y						

Quantifier distributivity

Finally, the real distributivity of quantifiers!

To use this inference operator, select:

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

If the selected form is:

- A universal quantifier having its second argument a conjunction, then AcXpq → XAcpAcq.
- An existential quantifier having its second argument a disjunction, then EcVpq → VEcpEcq.
- A conjunction whose arguments are universal quantifiers having the same condition, then XAcpAcq → AcXpq.

- A disjunction whose arguments are existential quantifiers having the same condition, then VEcpEcq $\rightarrow$ EcVpq.

The operator is based on the following formulas, where p(x), q(x), r(x) are predicates of variable x:

```
A(x|p(x)) X(q(x),r(x)) ↔ X(A(x|p(x)) q(x),A(y|p(y)) r(y))
E(x|p(x)) V(q(x),r(x)) ↔ V(E(x|p(x)) q(x),E(y|p(y)) r(y))
```

Example:

A		+					+				+
	S		X								
		^		E				+	P		
					T	@			a	0	^
							^	^	r		
									t		

If we select the first quantifier and call the inference operator, the result is:

X														
	A		+				+		A		+			+
		S		E				+		S		P		
			^		T	@					^	a	0	^
							^	^				r		
												t		

If we apply here the operator on the first position, we obtain the initial sentence.

Interchanging quantifiers

To use this inference operator, select:

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

The operator checks that one of the following conditions is met:

- The execution part of the destination is a quantifier of the same type, and the condition part of the second quantifier has no references to the destination.
- The destination is existential and implicable, its execution part is a universal quantifier, and the condition part of the second quantifier has no references to the destination.

This inference operator interchanges the quantifiers along with their conditions.

This inference operator is justified by the following formulas, where p(x) is a predicate of variable x, q(y) is a predicate of variable y, and r(x,y) a predicate of variables x and y:

```
A(x|p(x)) A(y|q(y)) r(x,y) ↔ A(y|q(y)) A(x|p(x)) r(x,y)
E(x|p(x)) E(y|q(y)) r(x,y) ↔ E(y|q(y)) E(x|p(x)) r(x,y)
E(x|p(x)) A(y|q(y)) r(x,y) → A(y|q(y)) E(x|p(x)) r(x,y)
```

Example:

A		+						+	
	S		A		+				+
		^		E		N			
				m	^		@		
				p				^	^
				t					
				y					

If we select the first quantifier and apply the operator “Interchanging quantifiers”, the result is:

A		+							+
	E		A		+			+	
	m	^		S		N			
	p				^		@		
	t							^	^
	y								

Particularization

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).

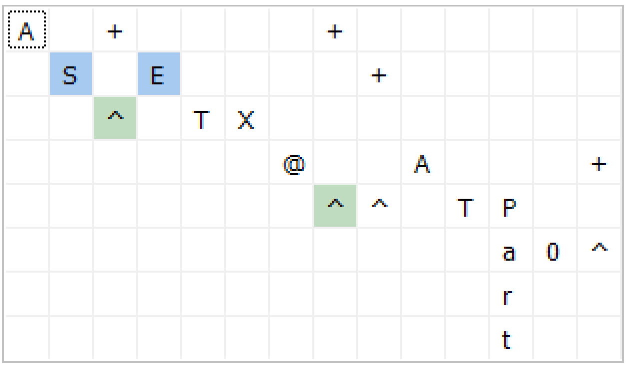
This operator verifies whether:

- If the source is a quantifier, it is an ancestor of the destination; if not, all the external references of the 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 destination with 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 destination with the execution part of the destination, where the links to the destination have been replaced by 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:

A		+					+				
	S		E					+			+
		^		T	X						
						@			P		
							^	^	a	0	^
									r		
									t		

Example: The source is a function.

A						+		+						
	T	A							+					
			T	X										
					P					A				+
					a	^	U				T	P		
					r			^	^			a	0	^
					t							r		
												t		

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

A						+		+					+	
	T	A							+					+
			T	X										
					P					P				
					a	^	U			a	0	U		
					r			^	^	r			^	^
					t					t				

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

The inference operators “Commutativity” and “Reflexivity” are also applicable for the equality of classes. They are applicable to equality because class equality is reflexive and symmetric. If we want to use the "Logic" system to create an axiomatic theory without "=" or in which "=" is used with other meanings, those operators will no longer work for "=". These operators are not absolutely necessary, but they are useful.

Commutativity

To use this inference operator, select:

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

This inference operator verifies whether the selected form is a conjunction, disjunction, double implication, or equality.

This inference operator interchanges the arguments of the selected form.

The operator is not absolutely necessary, but it is very useful. It should be noted that this inference operator can also be applied to equality.

Since for us, primitive relations are also logical operators, we will use in their cases names specific to functions, such as commutativity.

Reflexivity

To use this inference operator, select:

- a step of the proof, and
- a double implication or an equality from the step.

If the arguments of the selected form are equal, then this operator replaces the form with the logical operator "T".

This operator is not absolutely necessary. It was made only to simplify some demonstrations.

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 source references all the quantifiers of the pre-q-list of the source step.
- The source is equal to the destination.

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

A				+				+	
	T	C							
			S		E				+
				^		T	@		
								^	^

The destination step is:

A				+			+	+
	T	X						
			S		N			
				^		@		
							^	^

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

A					+				+				+	+
	T	X												
		X									N			
				S		E				+		@		
					^		T	@					^	^
									^	^				

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 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 destination is equal to the source.

The result of the operator consists of the destination step, in which the destination is replaced by 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:

A				+				+	
	T	D							
			S		E				+
				^		T	@		
								^	^

The destination step is

A				+			+	+
	T	X						
			S		N			
				^		@		
							^	^

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

A						+				+	+
	T	X									
			E				+	N			
				T	@				@		
						^	^			^	^

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 destination is replaced by the conjunction of the destination and the source step.

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

A		+			+	
	S		N			
		^		@		
					^	0

A		+						+		
	S		A			+			+	
		^		X						T
					S		@			
						^		^	^	

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

X																		
	A		+						+			A		+			+	
		S		A			+			+			S		N			
			^		X						T			^		@		
						S		@									^	0
							^		^	^								

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

A		+						+		
	S		A			+			+	
		^		X						T
					S		@			
						^		^	^	

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

A		+						+										
	S		A			+			+									
		^		X						X								
					S		@				T	A		+			+	
						^		^	^				S		N			
														^		@		
																	^	0

Formulas

On the "Formulas" page of the "Logic" program, some of the commonly used tautologies are defined. They begin with an implication or double implication. Some formulas have a name: Absorption, Injection, Associativity, Biconditionality, Contraposition, Disjunctive Syllogism, Exportation, Modus Ponens, etc. For other formulas, we have defined new names (short descriptions). The names of the formulas have no importance for the system. The user can also define new formulas as needed.

Using formulas

To use a formula, select:

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

If the formula we want to use is based on an implication, the selected form must be implicable.

We call the desired formula by double-clicking in the "Formulas" table.

The only difference between inference operators and formulas is that we choose the formula from the list of formulas. If the structure of the selected form corresponds to the first argument of the formula, the result of the operator is the selected step in which the selected form is replaced with the second argument of the formula.

Let us suppose that our proof contains the following sentence:

A							+		+					+		+				+
	T	A								+							+		+	
			T	X																
					X						C									
						S		@				X						X		
							^		^	^			S		@			Y	^	^
														^		^	^	Z		

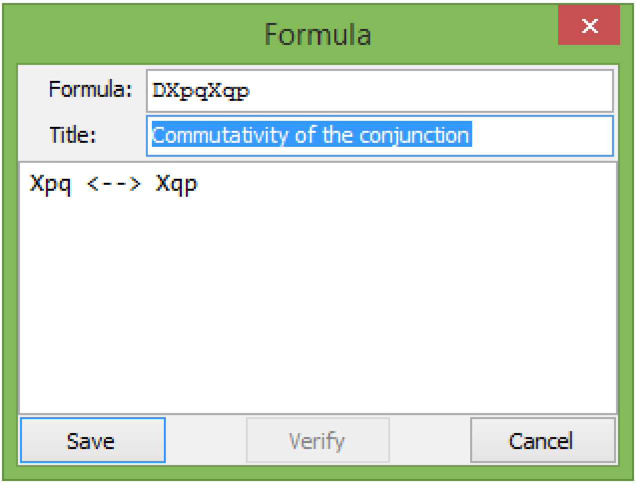
By applying “Modus ponens” for the conjunction on position 5, we obtain:

A						+
	T	A			+	
			T	X		
				Y	^	^
				Z		

Generating new formulas

Let us suppose we want to define a new formula for the commutativity of the conjunction. This is not necessary because we have the inference operator "Commutativity", but this is only an example. Using the button "Insert formula", we get the definition window. We must introduce the formula and its title (name/description). We must write the formula in Polish notation, using the variables p, q, r, and s (in this order). In our example, the formula is DXpqXqp.

After introducing the formula and the title, we can use the "Verify" button. If the formula is a tautology, the program writes a short description of the action of our formula in the memory field. We can complete this description and then use the button "Save":



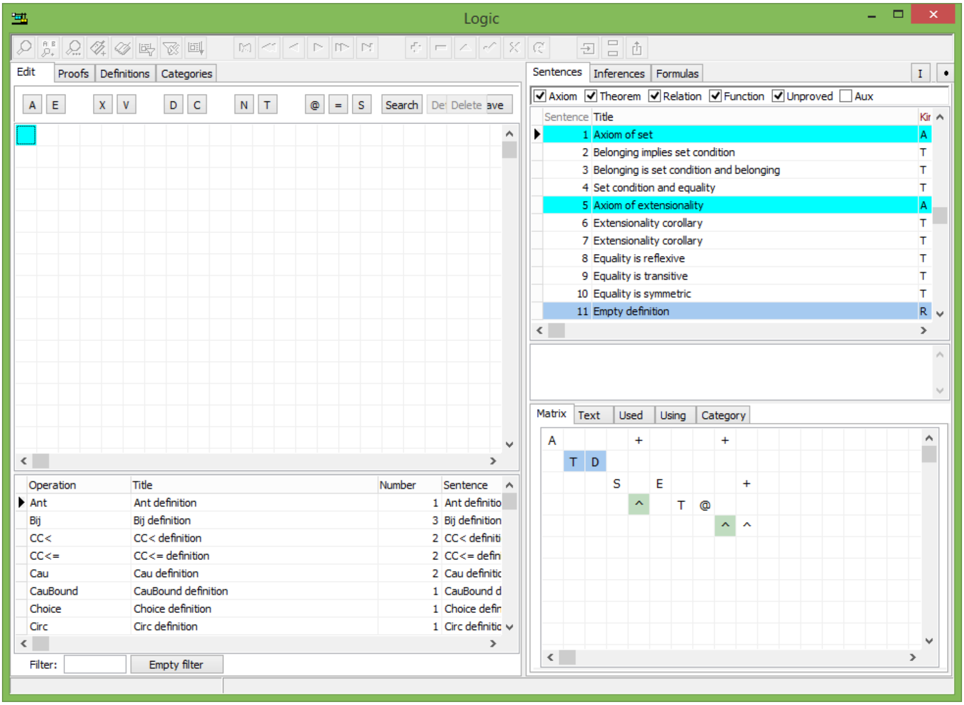
The title and the description of a formula can be changed at any time.

Verifying the proof

This is a pseudo-inference operator. It generates no step of the proof, but it verifies whether the last step of the proof is identical with the sentence we want to prove, or, in the case of using "Reductio ad absurdum", the last step of the proof is a contradiction (NT). In this case, the operator turns the kind of the sentence to "T", marking that the sentence is now a theorem. It is not possible to change or delete the proof of a theorem. If the theorem is not yet used to prove other theorems, and we want to change or delete the proof, we must first manually turn the kind of sentence to “U”.

The “Logic” program

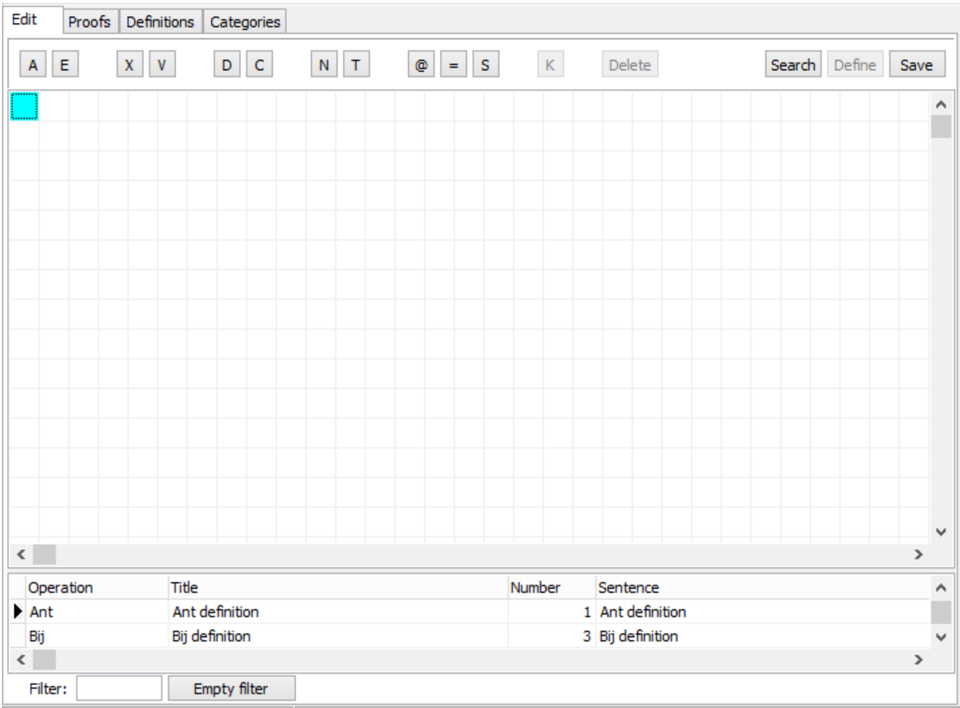
The “Logic” program is the user interface of our system:



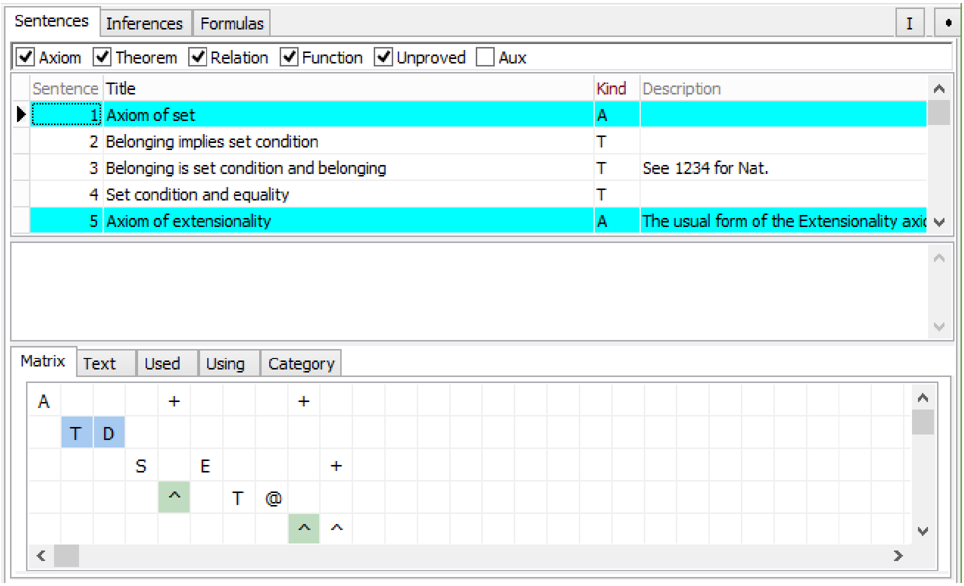
All the data in our system can be seen in tables: sentences, operators, etc. For navigation in tables, there is a navigation bar on the first line in the program:

With the navigation bar, we can do all we need to work with tables, but not everything is permitted. For example, it is not permitted to delete an axiom or a theorem if it is used in the proof of another theorem. When the system gives you an error message, simply cancel the last action and try to understand the problem. The navigation bar has a cancel button.

On the left side of the window we have the "Edit", "Proofs", "Definitions" and "Categories" tabs:



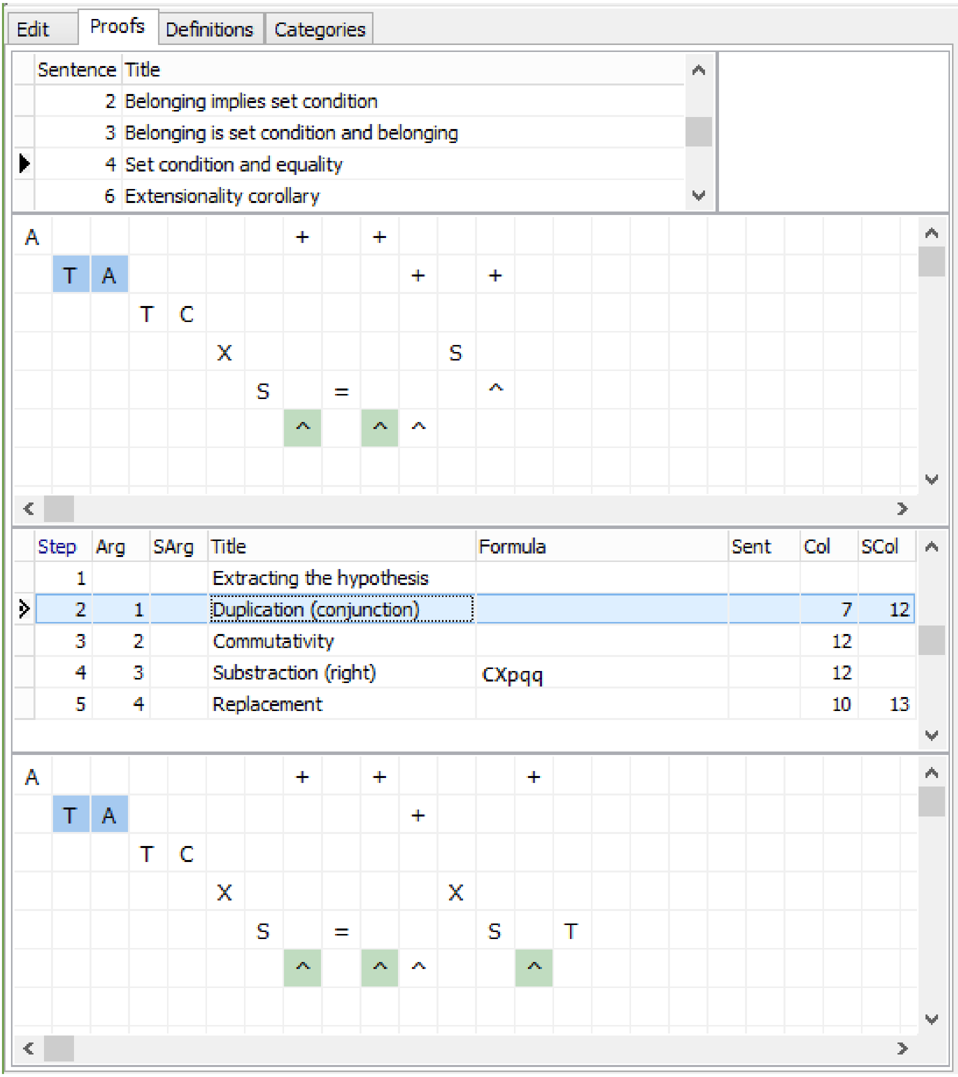
On the right side of the window we have the "Sentences", "Inferences", and "Formulas" tabs:



Edit

We can directly generate a sentence or start from an existing one. To load an existing sentence into the editor, double-click the sentence (or use Shift+Ctrl+C). We can use the active buttons and the list of user-defined (primitive) relations and functions from the table under the sentence. If necessary, we can delete logical operators with the “Delete” button. Finally, we can save the new sentence. We can save a sentence as an axiom, or we can leave it with the kind "U" (undefined).

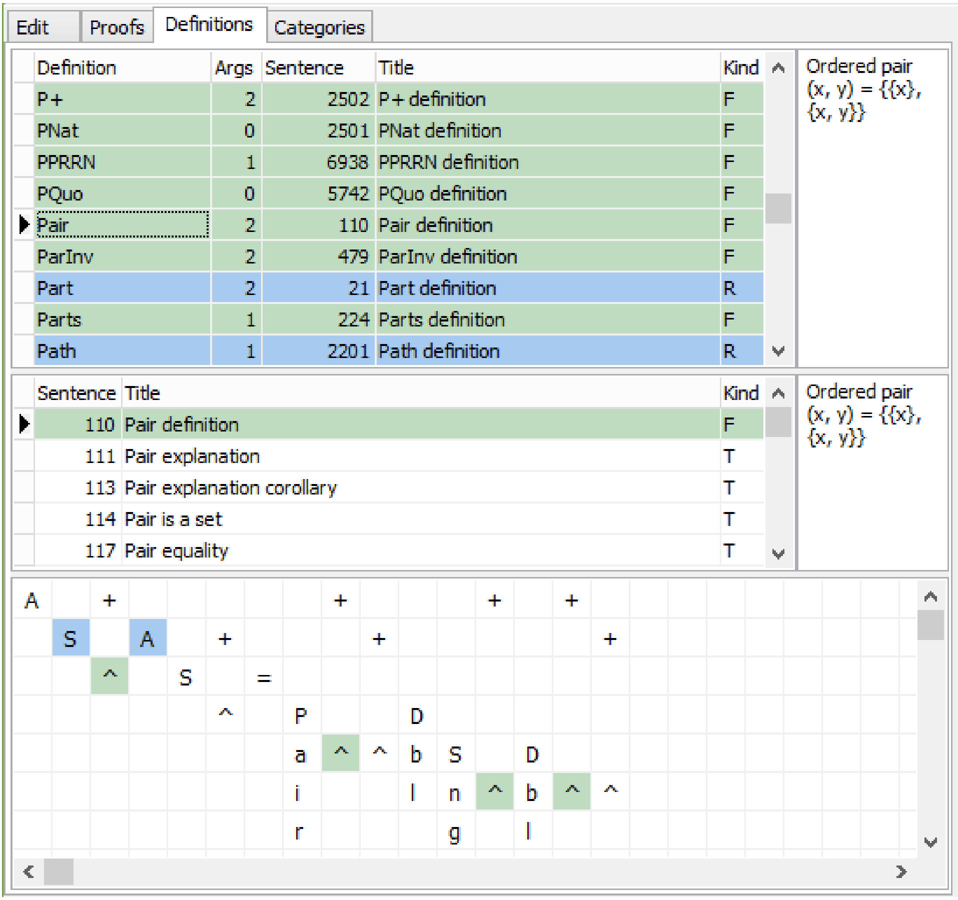
Proofs



In order to complete a proof, we load sentences from the “Sentences” page and call inference operators from the pages “Inferences” and “Formulas”.

For each step of the proof, we can see the result of the inference operator.

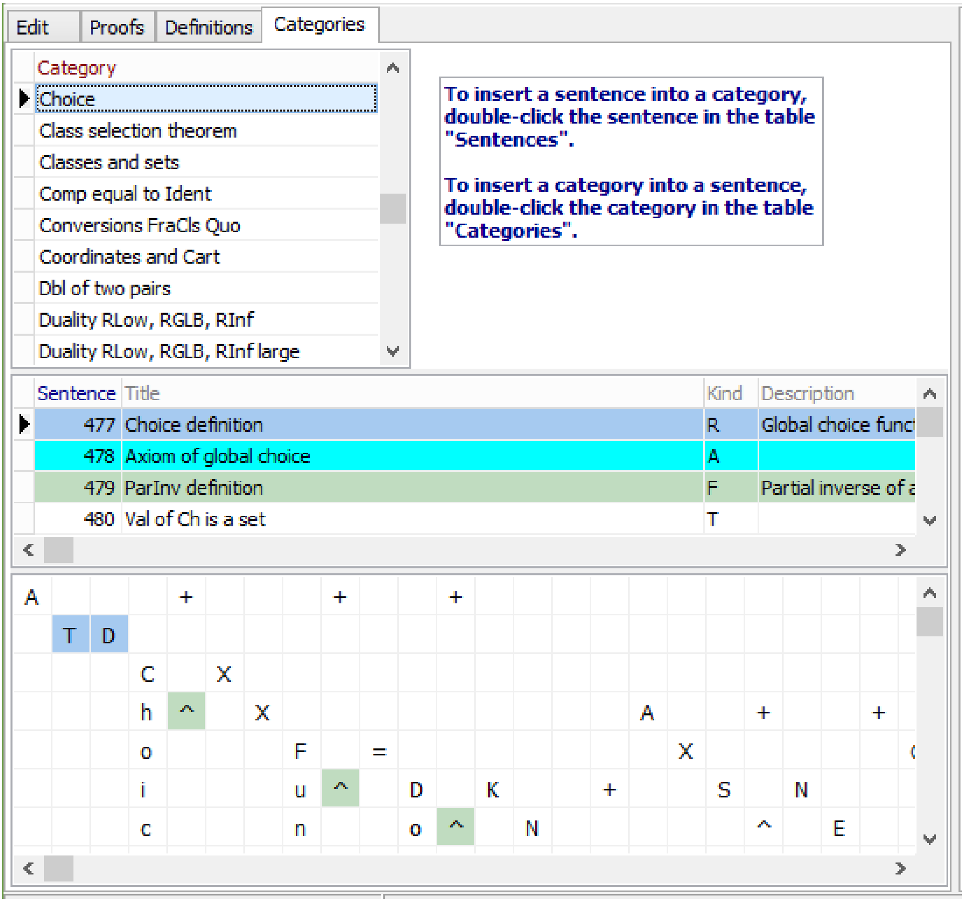
Definitions



The “Definitions” page contains the user-defined logical operators (relations and functions). We can see them in a table. In the next table, we can see the sentences containing the current relation or function.

On the “Sentences” page, we can also see the sentences used or using the current sentence. “Used sentences” means that the proof of our current sentence contains those sentences. “Using sentences” means that the proofs of those sentences contain the current sentence.

Categories



On this page, we can define categories to which we can attach sentences. We can also attach a category to a sentence. The meaning and role of a category depend only on the user’s intention. Defining, modifying, or deleting a category, as well as attaching sentences, has no influence on sentences or proofs. They are only auxiliary means of systematizing or searching for sentences.

An introduction to the axiomatic class theory

To verify our functional logic system for mathematics, we decided to use it for creating an axiomatic theory of classes.

Our concept does not allow the use of an axiom schema or a class existence theorem. Instead, we created the class generator as a logical operator. The class generator, together with the logical operator "S" and the inference operators "Class generator to form" and "Form to class generator", makes a class existence theorem unnecessary.

Mathematicians like to invent and use new notations and signs for new things. This can be beautiful, but it has nothing to do with the internal nature of things. Our system allows exclusively the definition of (primitive) relations and functions. The user gives names to the user-defined relation and function, and these names must be representative.

We also cannot adopt the convention that upper-case letter variables refer to classes and lower-case letter variables refer to sets. First of all, a statement does not depend on the names of its variables. Using something that does not influence a sentence in order to influence the sentence is more than interesting! To indicate that a class is a set, we always use the logical operator "S".

Our quantifiers are binary logical operators. They have a condition and an execution argument/part. That simplifies the writing of the set condition. If a quantifier does not need to be conditioned, then we use "T" for its condition.

We consider that a theory having the universe of discourse consisting of classes is natural to be called a class theory.

To prove a theorem in our system, we only use axioms, definitions, or theorems. We cannot use obvious things or helpful theorems left as exercises for the reader. For this reason, we have proved many secondary theorems, which are not very important in a presentation like this. For an exhaustive understanding, it is absolutely necessary to study our theorems in detail using the "Logic" program or the list of all sentences and proofs (an extract from the database). What follows is just a selection of some interesting things and new solutions to old problems.

In this presentation, the sentences are not numbered, but they receive the identification number from the database, followed by one of the letters A, T, F, R, U (axiom, theorem, definition of primitive function, definition of primitive relation, or unknown), and the title of the sentence.

The title and the description of a sentence are only auxiliary information and can be changed at any time.

For our purposes, we will use the following axioms:

- Axiom of set
- Axiom of extensionality
- Axiom of empty set
- Axiom of part
- Axiom of double
- Axiom of generalized union
- Axiom of power set
- Axiom of replacement
- Axiom of global choice
- Axiom of infinity
- Axiom of regularity

Classes and sets

At the beginning, we will use the following representations of sentences: the matrix form, the text form, the formalization with brackets, and the natural language. In this way, we want to make it easier for the reader to adapt to the sentence forms used by our proof system. For the text form, we have defined names for variables. We recommend that the reader recreate some theorems with their proofs using our system.

The space allows us to present only the important statements of this theory. For all the details, the reader can use our proof system or the text file with all the sentences and the proofs of the theorems.

Because in our system, "S" (set condition) is a predefined logical operator, we need an axiom to specify the connection between "S" and belonging.

1(A) Axiom of set

A				+				+	
	T	D							
			S		E				+
				^		T	@		
								^	^

```
A - - - + - - - + -
^ T D
X     S   E - - - +
        ^ ^ T @
        X Y     ^ ^
                X Y
```

`A(X) D(S(X),E(Y) @(X,Y))`

*Let **X** be a class.*
***X** is a set if and only if there exists **Y** such that **X** belongs to **Y**.*

The sentence below can be easily proven by reductio ad absurdum using the inference operators "Replacement" and "Reflexivity":

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

For this reason, instead of the extensionality axiom, we can use:

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

However, we have kept the axiom of extensionality in its original form because of its beauty!

5(A) Axiom of extensionality

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

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

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

*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**.*

Or, if we want, we can say:

***X** = **Y** if and only if for any set **u**, **u** belongs to **X** implies **u** belongs to **Y**, and **u** belongs to **Y** implies **u** belongs to **X**.*

In the case of using variables, what do we actually mean when we say that "X is equal to Y"? Is variable X equal to variable Y? No, because in this case we would not need two variables. If we think that in our case the variables represent classes, then "X is equal to Y" may mean that the class represented by X is equal to the class represented by Y. No, that's not true either! There are no equal classes; classes are unique. Maybe it would be correct to say that X and Y represent the same class, but no one says that...

6(T) Extensionality corollary

A						+							+			
	T	A					+									+
			T	D												
					=			A				+			+	
						^	^		T	D						
											@			@		
												^	^		^	^

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

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

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

Proof

1. Axiom of extensionality (Sentence: 5)

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8
A - - - - - + - - - - - - - + - - -
  T A - - - - + - - - - - - - - - +
      T D
          =     A - + - - + - - + -
            ^ ^   S   D
                    ^   @     @
                          ^ ^   ^ ^
```

2. Extracting the condition of the quantifier (Step: 1, Col: 9)

```
                  1                   2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
A - - - - - + - - - - - - - - - + - - -
  T A - - - - + - - - - - - - - - - - +
      T D
          =     A - - - + - - + - - + -
            ^ ^   T C
                      S   D
                        ^   @     @
                              ^ ^   ^ ^
```

3. DCpDqrDXpqXpr (Step: 2, Col: 11)

```
                  1                   2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
A - - - - - + - - - - - - - - - + - - - - - -
  T A - - - - + - - - - - - - - - - - - - - +
      T D
          =     A - - - - + - + - - - + - + -
            ^ ^   T D
                      X           X
                        S   @       S   @
                          ^   ^ ^     ^   ^ ^
```

4. Belonging is set condition and belonging (Sentence: 3)

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2 3 4
A - - - - - + - - - + - + -
  T A - - - - + - - - - - +
      T D
          @     X
            ^ ^   S   @
                    ^   ^ ^
```

5. Commutativity (Step: 4, Col: 5)

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2 3 4
A - - - - - - + - + - - + -
  T A - - - - - - - + - - +
      T D
          X           @
            S   @       ^ ^
              ^   ^ ^
```

6. Cross replacement (Step: 5, SStep: 3, Col: 12)

```
                  1                   2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
A - - - - - + - - - - - - + - - - - - -
  T A - - - - + - - - - - - - - - - - +
      T D
          =     A - - - + - - - + - + -
            ^ ^   T D
                      @     X
                        ^ ^   S   @
                                ^   ^ ^
```

7. Cross replacement (Step: 5, SStep: 6, Col: 15)

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
A - - - - - + - - - - - - + - - -
  T A - - - - + - - - - - - - - +
      T D
          =     A - - - + - - + -
            ^ ^   T D
                      @     @
                        ^ ^   ^ ^
```

8(T) Equality is reflexive

A			+	+
	T	=		
			^	^

```
A - - + +
^ T =
X     ^ ^
      X X
```

`A(X) =(X,X)`

Each class is equal to itself.

We will first present how the proof of this theorem looks in mathematics books:

The statement x ∈ X ⇒ x ∈ X and x ∈ X ⇒ x ∈ X is obviously true; thus, by theorem 6, X = X.

Anyone who wants to understand this "proof" will want to know where x comes from and why it belongs to X. If we follow the proof in our system, we will see that these strange elements are actually the content of the universal quantifier in position 9 of theorem 6 that has been taken out of context.

Proof

The first step of the proof is the loading of theorem 6:

1. Extensionality corollary (Sentence: 6)

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
A - - - - - + - - - - - - + - - -
  T A - - - - + - - - - - - - - +
      T D
          =     A - - - + - - + -
            ^ ^   T D
                      @     @
                        ^ ^   ^ ^
```

The second step is the call of the inference operator "Commutativity" for the result of the first step at the 5th position (the double implication "D"):

2. Commutativity (Step: 1, Col: 5)

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
A - - - - - - - - - + - - - - + -
  T A - - - - - - - - - - + - - +
      T D
          A - - - + - - + - =
            T D               ^ ^
                @     @
                  ^ ^   ^ ^
```

We now want to transform the double implication at position 5 into an implication. To do this, we call the "CDpqCpq" formula at position 5.

3. CDpqCpq (Step: 2, Col: 5)

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
A - - - - - - - - - + - - - - + -
  T A - - - - - - - - - - + - - +
      T C
          A - - - + - - + - =
            T D               ^ ^
                @     @
                  ^ ^   ^ ^
```

The result of the third step has a universal quantifier in the third position, which we want to particularize using the quantifier in the first position. Therefore, we call the "Particularization" operator for positions (columns) 1 and 3:

4. Particularization (Step: 3, Col: 1, SCol: 3)

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
A - - - - - - - + - - + - + +
  T C
      A - - - + - - + - =
        T D           ^ ^
            @     @
              ^ ^   ^ ^
```

At the 6th position we have the tautology "Dpp". Applying the "DDppT" formula to the 6th position, we obtain:

5. DDppT (Step: 4, Col: 6)

```
1 2 3 4 5 6 7 8 9
A - - - - - - + +
  T C
      A - - =
        T T   ^ ^
```

Now we can delete the quantifier from the 6th position:

6. Deleting a quantifier (Step: 5, Col: 4)

```
1 2 3 4 5 6 7
A - - - - + +
  T C
      T =
          ^ ^
```

By applying the "DCTpp" formula to the 3rd position, the proof is complete:

7. DCTpp (Step: 6, Col: 3)

```
1 2 3 4 5
A - - + +
  T =
      ^ ^
```

9(T) Equality is transitive

A									+						+	
	T	A								+		+				
			T	A									+			+
					T	C										
							X							=		
								=			=				^	^
									^	^		^	^			

```
A - - - - - - - - + - - - - - + -
^ T A - - - - - - - + - + - - - -
X   ^ T A - - - - - - - - + - - +
    Y   ^ T C
        Z     X             =
                =     =       ^ ^
                  ^ ^   ^ ^   X Z
                  X Y   Y Z
```

```
A(X) A(Y) A(Z) C(X(=(X,Y),=(Y,Z)),=(X,Z))
```

*Let **X**, **Y**, **Z** be classes. If **X** = **Y** and **Y** = **Z**, then **X** = **Z**.*

Proof

1. Extracting the hypothesis

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
A - - - - - - - - + - - - - -
  T A - - - - - - - + - + - -
      T A - - - - - - - - + -
          T C
              X             T
                =     =
                  ^ ^   ^ ^
```

2. DCpTCpp (Step: 1, Col: 7)

```
                  1                   2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
A - - - - - - - - + - - - - - - + - - - -
  T A - - - - - - - + - + - - - - + - + -
      T A - - - - - - - - + - - - - - - +
          T C
              X             X
                =     =       =     =
                  ^ ^   ^ ^     ^ ^   ^ ^
```

3. Replacement (Step: 2, Col: 20, SCol: 18)

```
                  1                   2
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
A - - - - - - - - + - - - - - - + - - - -
  T A - - - - - - - + - + - - - - - - + -
      T A - - - - - - - - + - - - + - - +
          T C
              X             X
                =     =       =     =
                  ^ ^   ^ ^     ^ ^   ^ ^
```

4. CXpqp (Step: 3, Col: 15)

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
A - - - - - - - - + - - - - - + -
  T A - - - - - - - + - + - - - -
      T A - - - - - - - - + - - +
          T C
              X             =
                =     =       ^ ^
                  ^ ^   ^ ^
```

10(T) Equality is symmetric

A						+				+
	T	A					+		+	
			T	D						
					=			=		
						^	^		^	^

```
A - - - - - + - - - +
^ T A - - - - + - + -
X   ^ T D
    Y     =     =
            ^ ^   ^ ^
            X Y   Y X
```

`A(X) A(Y) D(=(X,Y),=(Y,X))`

*Let **X**, **Y** be classes. **X** = **Y** if and only if **Y** = **X**.*

Proof

1. Extracting the hypothesis

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
A - - - - - - + - - - - - + -
  T A - - - - - + - - - + - -
      T X
          C         C
            =     T   =     T
              ^ ^       ^ ^
```

2. DCpTCpp (Step: 1, Col: 6)

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
A - - - - - - + - - + - - - - + -
  T A - - - - - + - - + - - + - -
      T X
          C             C
            =     =       =     T
              ^ ^   ^ ^     ^ ^
```

3. DCpTCpp (Step: 2, Col: 13)

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
A - - - - - - + - - + - - - - + - - +
  T A - - - - - + - - + - - + - - + -
      T X
          C             C
            =     =       =     =
              ^ ^   ^ ^     ^ ^   ^ ^
```

4. Replacement (Step: 3, Col: 9, SCol: 12)

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
A - - - - - - + - - + + - - - + - - +
  T A - - - - - + - - - - - + - - + -
      T X
          C             C
            =     =       =     =
              ^ ^   ^ ^     ^ ^   ^ ^
```

5. Replacement (Step: 4, Col: 8, SCol: 11)

```
                  1
```

```
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
A - - - - - - + - - - + - - - + - - +
  T A - - - - - + - + - - - + - - + -
      T X
          C             C
            =     =       =     =
              ^ ^   ^ ^     ^ ^   ^ ^
```

6. Replacement (Step: 5, Col: 15, SCol: 18)

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
A - - - - - - + - - - + - - - + - + +
  T A - - - - - + - + - - - + - - - -
      T X
          C             C
            =     =       =     =
              ^ ^   ^ ^     ^ ^   ^ ^
```

7. Replacement (Step: 6, Col: 16, SCol: 19)

```
                  1
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
A - - - - - - + - - - + - - - + - + -
  T A - - - - - + - + - - - + - - - +
      T X
          C             C
            =     =       =     =
              ^ ^   ^ ^     ^ ^   ^ ^
```

8. DXCpqCqpDpq (Step: 7, Col: 5)

```
                  1
1 2 3 4 5 6 7 8 9 0 1
A - - - - - + - - - +
  T A - - - - + - + -
      T D
          =     =
            ^ ^   ^ ^
```

We have proved theorems 9 and 10 without using the extensionality axiom. However, both theorems can also be proven using this axiom. In other words, this axiom implies the transitivity and the symmetry of the class equality.

We say that a class is empty if it has no elements. The existence of an empty class is guaranteed by an axiom, and its uniqueness is given by a theorem. For the empty class, we will use the notation "0". By an axiom, we establish that "0" is a set, so we can call it "the empty set".

11(R) Empty class definition

A				+							+
	T	D									
			E		A		+			+	
			m	^		S		N			
			p				^		@		
			t							^	^
			y								

```
A - - - + - - - - - - - +
^ T D
X     E   A - + - - + -
      m ^ ^ S   N
      p X u   ^   @
      t       u     ^ ^
      y             u X
```

```
A(X) D(Empty(X),A(u|S(u)) N(@(u,X)))
```

*A class **X** is empty (Empty(**X**)) if and only if no set **u** belongs to **X**.*

12(F) 0 definition

=				
	0	K		
			N	
				T

```
=
 0 K - -
     N
       T
```

```
=(0,{u|N(T)})
```

0 is equal to {u|N(T)}

13(T) Empty class corollary

A				+						+
	T	D								
			E		A				+	
			m	^		T	N			
			p					@		
			t						^	^
			y							

```
A - - - + - - - - - - +
^ T D
X     E   A - - - + -
      m ^ ^ T N
      p X u     @
      t           ^ ^
      y           u X
```

```
A(X) D(Empty(X),A(u) N(@(u,X)))
```

*A class **X** is empty (Empty(**X**)) if and only if no class **u** belongs to **X**.*

Proof

1. Reductio ad absurdum
2. Empty class definition (Sentence: 11)
3. Cross replacement (Step: 2, SStep: 1, Col: 5)
4. Negation (Step: 3, Col: 3)
5. Negation (Step: 4, Col: 12)
6. Negation (Step: 5, Col: 14)
7. Negation (Step: 6, Col: 18)
8. Negation (Step: 7, Col: 21)
9. Commutativity (Step: 8, Col: 17)
10. Partial quantifier distributivity (Step: 9, Col: 17)
11. Particularization (Step: 10, Col: 17, SCol: 21)
12. DXNppNT (Step: 11, Col: 20)
13. Deleting a quantifier (Step: 12, Col: 17)
14. DVpNTp (Step: 13, Col: 3)
15. Partial quantifier distributivity (Step: 14, Col: 3)
16. Belonging implies set condition (Sentence: 2)
17. Inserting a sentence into another sentence (Step: 16, SStep: 15, Col: 13)
18. Particularization (Step: 17, Col: 15, SCol: 17)
19. Particularization (Step: 18, Col: 16, SCol: 17)
20. Particularization (Step: 19, Col: 18, SCol: 6)
21. CXpqp (Step: 20, Col: 10)
22. DXNppNT (Step: 21, Col: 5)
23. Deleting quantifiers from right to left (Step: 22, Col: 3)

14(T) Empty class uniqueness

A		+					+	
	E		A		+			+
	m	^		E		=		
	p			m	^		^	^
	t			p				
	y			t				
				y				

```
A - + - - - - + -
^ E   A - + - - +
X m ^ ^ E   =
  p X Y m ^   ^ ^
  t     p Y   X Y
  y     t
        y
```

A(X|Empty(X)) A(Y|Empty(Y)) =(X,Y)

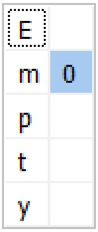
If two classes are empty, then they are equal.

Proof

1. Extracting the hypothesis
2. Empty class definition (Sentence: 11)
3. CDpqCpq (Step: 2, Col: 3)
4. Inserting the condition into the quantifier (Step: 3, Col: 1)
5. Inserting a sentence into another sentence (Step: 4, SStep: 1, Col: 7)
6. CXpqq (Step: 5, Col: 7)

```
7. Particularization (Step: 6, Col: 1, SCol: 7)
8. Inserting a sentence into another sentence (Step: 4, SStep: 7, Col: 7)
9. Particularization (Step: 8, Col: 4, SCol: 15)
10. Quantifier distributivity (Step: 9, Col: 7)
11. CXpqDpq (Step: 10, Col: 10)
12. DDpqDNpNq (Step: 11, Col: 10)
13. Negation (Step: 12, Col: 11)
14. Negation (Step: 13, Col: 14)
15. Axiom of extensionality (Sentence: 5)
16. Commutativity (Step: 15, Col: 5)
17. Cross replacement (Step: 16, SStep: 14, Col: 7)
```

15(T) 0 theorem




```
E
m 0
p
t
y
```

```
Empty(0)
```

0 is empty.

Proof

1. Reductio ad absurdum

```
 1 2 3
 N
   E
   m 0
   p
   t
   y
```

2. Empty class corollary (Sentence: 13)

```
                   1
 1 2 3 4 5 6 7 8 9 0 1
 A - - - + - - - - - +
   T D
       E   A - - - + -
       m ^   T N
       p         @
       t           ^ ^
       y
```

3. Inserting a sentence into another sentence (Step: 2, SStep: 1, Col: 1)

```
                   1
 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
 X
   N     A - - - + - - - - - - +
     E     T D
     m 0       E   A - - - + -
     p         m ^   T N
     t         p       @
     y         t         ^ ^
               y
```

4. Particularization (Step: 3, Col: 4, SCol: 5)

```
                   1
 1 2 3 4 5 6 7 8 9 0 1 2 3
 X
   N     D
     E     E   A - - - + -
     m 0   m 0   T N
     p     p         @
     t     t           ^ 0
     y     y
```

5. Replacement (Step: 4, Col: 6, SCol: 3)

```
                   1
 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
 X
   N             D
     A - - - + -   E   A - - - + -
       T N         m 0   T N
           @       p         @
             ^ 0   t           ^ 0
                   y
```

6. CXpqp (Step: 5, Col: 1)

```
 1 2 3 4 5 6 7
 N
   A - - - + -
     T N
         @
           ^ 0
```

7. Negation (Step: 6, Col: 1)

```
1 2 3 4 5 6 7
E - - - - + -
  T N
      N
        @
          ^ 0
```

8. Negation (Step: 7, Col: 3)

```
1 2 3 4 5
E - - + -
  T @
      ^ 0
```

9. 0 definition (Sentence: 12)

```
1 2 3 4 5
=
  0 K - -
      N
        T
```

10. Cross replacement (Step: 9, SStep: 8, Col: 5)

```
1 2 3 4 5 6 7
E - - + - - -
  T @
      ^ K - -
          N
            T
```

11. Class generator to form (Step: 10, Col: 3)

```
1 2 3 4 5 6 7
E - - - + - -
  T X
      S   N
        ^   T
```

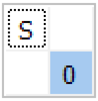
12. CXpqq (Step: 11, Col: 3)

```
1 2 3 4
E - - -
  T N
      T
```

13. Deleting a quantifier (Step: 12, Col: 1)

```
1 2
N
  T
```

16(A) Axiom of empty set (0 is a set.)



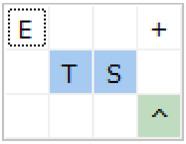

```
S
  0
```

```
S(0)
```

0 is a set.

According to this axiom, "0" is a set, so we can say that "0" is the empty set. The axiom actually guarantees the existence of sets.

17(T) Set existence




```
E - - +
^ T S
X     ^
      X
```

```
E(X) S(X)
```

There exists a class that is a set.

Proof

1. Axiom of empty set (Sentence: 16)

```
1 2
S
  0
```

2. Existential generalization (using equality) (Step: 1, Col: 1, SCol: 2)

```
1 2 3 4 5 6
E - + - - +
  =     S
    ^ 0   ^
```

3. Extracting the condition of the quantifier (Step: 2, Col: 1)

```
1 2 3 4 5 6 7 8
E - - - + - - +
  T X
      =     S
        ^ 0   ^
```

4. CXpqq (Step: 3, Col: 3)

```
1 2 3 4
E - - +
  T S
      ^
```

18(T) No set belongs to 0

A		+			+	
	S		N			
		^		@		
					^	0

```
A - + - - + -
^ S   N
x   ^   @
    x     ^ 0
          x
```

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

No set belongs to 0.

The proof of this theorem was explained in detail in the "Proofs" chapter.

Since the proofs of the theorems can be understood much more easily using the "Logic" system or the text file containing all the sentences and proofs, we will give up presenting the proofs of the following theorems.

19(T) Nothing belongs to 0

A				+	
	T	N			
			@		
				^	0

```
A - - - + -
^ T N
X     @
        ^ 0
        X
```

```
A(X) N(@(X,0))
```

20(T) Empty is equality to 0

A				+			+
	T	D					
			=			E	
				^	0	m	^
						p	
						t	
						y	

```
A - - - + - - +
^ T D
X     =     E
        ^ 0 m ^
        X   p X
            t
            y
```

```
A(X) D(=(X,0),Empty(X))
```

21(R) Part definition (inclusion)

A					+						+			
	T	A				+								+
			T	D										
					P			A		+			+	
					a	^	^		@			@		
					r					^	^		^	^
					t									

```
A - - - - - + - - - - + - - -
^ T A - - - - + - - - - - - +
X   ^ T D
    Y     P     A - + - - + -
          a ^ ^ ^ @     @
          r X Y u   ^ ^   ^ ^
          t         u X   u Y
```

```
A(X) A(Y) D(Part(X,Y),A(u|@(u,X)) @(u,Y))
```

*Let **X**, **Y** be two classes. We say that **X** is included in **Y** (**X** is a part of **Y** or Part of **X**, **Y**) if and only if, for any **u** belonging to **X**, **u** belongs to **Y**.*

22(A) Axiom of part

A		+				+		
	S		A		+			+
		^		P			S	
				a	^	^		^
				r				
				t				

```
A - + - - - + - -
^ S   A - + - - +
A   ^ ^ P     S
    A X a ^ ^   ^
        r X A   X
        t
```

```
A(A|S(A)) A(X|Part(X,A)) S(X)
```

A class included in a set is a set.

23(T) Part is implied by equality

A				+			+	
	T	A			+			+
			=			P		
				^	^	a	^	^
						r		
						t		

```
A - - - + - - + -
^ T A - - + - - +
X   ^ =     P
    Y   ^ ^ a ^ ^
        X Y r X Y
            t
```

`A(X) A(Y|=(X,Y)) Part(X,Y)`

*Let **X**, **Y** be classes. If **X** = **Y**, then **X** is included in **Y**.*

24(T) Part is reflexive

A			+	+
	T	P		
		a	^	^
		r		
		t		

```
A - - + +
^ T P
X   a ^ ^
    r X X
    t
```

`A(X) Part(X,X)`

Any class is included in itself.

The text form contains all the information of the matrix form but also may contain variables, so it is the best form for this presentation. For reasons of space, we will gradually abandon the presentation of the matrix form of sentences.

25(T) Part is transitive

```
A - - - - - - - - + - - - - - + -
^ T A - - - - - - - + - + - - - -
X   ^ T A - - - - - - - - + - - +
    Y   ^ T C
        Z     X             P
                P     P     a ^ ^
                a ^ ^ a ^ ^ r X Z
                r X Y r Y Z t
                t     t
```

`A(X) A(Y) A(Z) C(X(Part(X,Y),Part(Y,Z)),Part(X,Z))`

*Let **X**, **Y**, **Z** be classes. If **X** is included in **Y** and **Y** is included in **Z**, then **X** is included in **Z**.*

26(T) Equality is double Part

```
A - - - - - + - - - + - - - +
^ T A - - - - + - - - + - + -
X   ^ T D
    Y     =     X
            ^ ^   P     P
            X Y   a ^ ^ a ^ ^
                  r X Y r Y X
                  t     t
```

`A(X) A(Y) D(=(X,Y),X(Part(X,Y),Part(Y,X)))`

*Let **X**, **Y** be classes. **X** = **Y** if and only if **X** is included in **Y** and **Y** is included in **X**.*

In proofs, for processing the sentences that contain the logical operator "K" (class generator), we will use the inference operator "Class generator to form" and its inverse ("Form to class generator").

27(F) Dbl definition (double, doubleton, unordered pair)

```
A - + - - - - - + - - - - - + - - -
^ S   A - + - - - + - - - - - - - - +
x   ^ ^ S   =
    x y   ^   D     K - - + - - + -
          y   b ^ ^ ^ V
              l x y u   =     =
                          ^ ^   ^ ^
                          u x   u y
```

`A(x|S(x)) A(y|S(y)) =(Dbl(x,y),{u|V(=(u,x),=(u,y))})`

*Let **x**, **y** be two sets. {**x**, **y**} = {**u** | **u** = **x** ∨ **u** = **y**}*

28(A) Axiom of double

A		+						+	
	S		A		+				+
		^		S		S			
					^		D		
							b	^	^
							l		

```
A - + - - - - - + -
^ S   A - + - - - +
x   ^ ^ S   S
    x y   ^   D
          y   b ^ ^
              l x y
```

`A(x|S(x)) A(y|S(y)) S(Dbl(x,y))`

*If **x**, **y** are sets, then {**x**, **y**} is also a set.*

30(T) Dbl explanation

```
A - + - - - - - - - - - + - - - - + - - -
^ S   A - + - - - - - - - + - - - - - - +
x   ^ ^ S   A - - - + - - - - - + - - + -
    x y   ^ ^ T D
          y u     @         V
                    ^ D       =     =
                    u b ^ ^     ^ ^   ^ ^
                      l x y     u x   u y
```

`A(x|S(x)) A(y|S(y)) A(u) D(@(u,Dbl(x,y)),V(=(u,x),=(u,y)))`

*Let **x**, **y** be sets and **u** a class. **u** belongs to {**x**, **y**} if and only if **u** = **x** or **u** = **y**.*

31(T) The first argument of Dbl belongs to Dbl

A		+					+		+	
	S		A		+					+
		^		S		@				
					^		^	D		
								b	^	^
								l		

```
A - + - - - - + - + -
^ S   A - + - - - - +
x   ^ ^ S   @
    x y   ^   ^ D
          y   x b ^ ^
                l x y
```

`A(x|S(x)) A(y|S(y)) @(x,Dbl(x,y))`

32(T) The second argument of Dbl belongs to Dbl

A		+							+	
	S		A		+		+			+
		^		S		@				
					^		^	D		
								b	^	^
								l		

```
A - + - - - - - - + -
^ S   A - + - + - - +
x   ^ ^ S   @
    x y   ^   ^ D
          y   y b ^ ^
                l x y
```

`A(x|S(x)) A(y|S(y)) @(y,Dbl(x,y))`

33(T) Only the arguments of Dbl belong to Dbl

```
A - + - - - - - - - - - - - + - - - - - - - - + -
^ S   A - + - - - - - - - - - - - - + - - - - - +
x   ^ ^ S   A - - + - - - + - - - + - - - + - - -
    x y   ^ ^ X                       N
          y u   S   X                   @
                  ^   N       N           ^ D
                  u     =       =         u b ^ ^
                          ^ ^     ^ ^       l x y
                          u x     u y
```

```
A(x|S(x)) A(y|S(y)) A(u|X(S(u),X(N(=(u,x)),N(=(u,y))))) N(@(u,Dbl(x,y)))
```

*Let **x**, **y**, **u** be sets. If a set **u** is not equal to either **x** or **y**, then **u** does not belong to {**x**, **y**}.*

34(T) Dbl is commutative

```
A - + - - - - - + - - - +
^ S   A - + - - + - + -
x   ^ ^ S   =
    x y   ^   D     D
          y   b ^ ^ b ^ ^
              l x y l y x
```

```
A(x|S(x)) A(y|S(y)) =(Dbl(x,y),Dbl(y,x))
```

*For any **x**, **y** sets, {**x**, **y**} = {**y**, **x**}.*

38(T) Dbl equality

```
A - + - - - - - - - - - - - - + - - - - - - - + - - - - - - + - - - -
^ S   A - + - - - - - - - - - - + - - - - - - - - - + - - - - - - + -
x   ^ ^ S   A - + - - - - - - - - - + - - - - - + - - - - - - - - - +
    x y   ^ ^ S   A - + - - - - - - - + - - - - - - - + - - - + - - -
          y a   ^ ^ S   D
                a b   ^   =             V
                      b     D     D       X             X
                            b ^ ^ b ^ ^     =     =       =     =
                            l x y l a b       ^ ^   ^ ^     ^ ^   ^ ^
                                              x a   y b     x b   y a
```

```
A(x|S(x)) A(y|S(y)) A(a|S(a)) A(b|S(b))
D(=(Dbl(x,y),Dbl(a,b)),V(X(=(x,a),=(y,b)),X(=(x,b),=(y,a))))
```

*Let **x**, **y**, **a**, **b** be sets. {**x**, **y**} = {**a**, **b**} if and only if **x** = **a** and **y** = **b**, or **x** = **b** and **y** = **a**.*

39(F) Sng definition (singleton)

A		+			+				+
	S		=						
		^		S		K		+	
				n	^		=		
				g				^	^

```
A - + - - + - - - +
^ S   =
x   ^   S   K - + -
    x   n ^ ^ =
        g x u   ^ ^
                u x
```

`A(x|S(x)) =(Sng(x),{u|=(u,x)})`

*Let **x** be a set. {**x**} = {**u** | **u** = **x**}*

40(T) Sng explanation

```
A - + - - - - - - + - - +
^ S   A - - - + - - - + -
x   ^ ^ T D
    x u     @       =
              ^ S     ^ ^
              u n ^   u x
                g x
```

`A(x|S(x)) A(u) D(@(u,Sng(x)),=(u,x))`

*Let **x** be a set and **u** be a class. **u** belongs to {**x**} if and only if **u** = **x**.*

41(T) Sng is Dbl

A		+			+		+	+
	S		=					
		^		S		D		
				n	^	b	^	^
				g		l		

```
A - + - - + - + +
^ S   =
x   ^   S   D
    x   n ^ b ^ ^
        g x l x x
```

`A(x|S(x)) =(Sng(x),Dbl(x,x))`

*For any set **x**, {**x**} = {**x**, **x**}.*

42(T) Sng is a set

A		+			+
	S		S		
		^		S	
				n	^
				g	

```
A - + - - +
^ S   S
x   ^   S
    x   n ^
        g x
```

`A(x|S(x)) S(Sng(x))`

*For any set **x**, {**x**} is a set.*

{x, y} is called a doubleton because it contains two elements, right? If x = y, then {x, y} = {x, x} = {x}, so it has only one element. We intentionally used a sequence of equalities, something loved by mathematicians but also more than problematic (when we think that equality is a binary relation).

Using the class generator with the condition "T" (i.e., no condition), we will define the universal class (Univ). This means that Univ is the class of all elements (i.e., of all sets). Every set belongs to the universal class, and every class is included in the universal class. Belonging to the universal class is equivalent to the set condition. In a way, we can consider the universal class as the opposite of the empty set.

43(F) Univ definition (universal class)

=			
	U	K	
	n		T
	i		
	v		

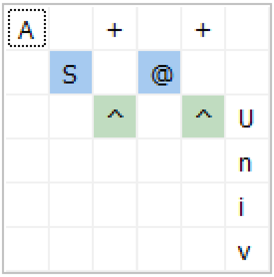
```
=
  U K -
  n ^ T
  i u
  v
```

`=(Univ,{u|T})`

*Univ is the class of those elements **u** with the condition 'T' (i.e., without any condition).*

From this definition, it does not follow that any class belongs to Univ.

44(T) Every set belongs to the universal class

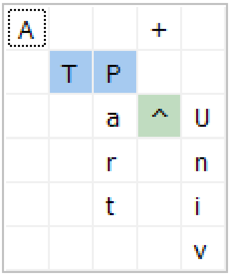


```
A - + - + -
^ S   @
x   ^   ^ U
    x   x n
          i
          v
```

`A(x|S(x)) @(x,Univ)`

Every set belongs to the universal class.

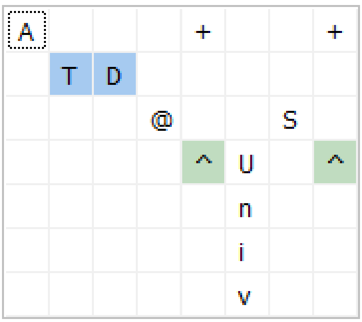
45(T) Every class is included in the universal class



```
A - - + -
^ T P
X   a ^ U
    r X n
    t   i
        v
```

`A(X) Part(X,Univ)`

46(T) Belonging to Univ is set condition



```
A - - - + - - +
^ T D
X     @     S
        ^ U   ^
        X n   X
          i
          v
```

`A(X) D(@(X,Univ),S(X))`

*Let **X** be a class. **X** belongs to Univ if and only if **X** is a set.*

So, we can say that Univ is the class of all the sets.

To study the algebra of classes, we will define the union and intersection of classes and the complement of a class. Among other things, we will show the absorption and DeMorgan's laws. Two theorems will show that Russell's paradoxes no longer exist.

47(F) U definition (union)

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

`A(X) A(Y) =(U(X,Y),{u|V(@(u,X),@(u,Y))})`

*Let **X**, **Y** be classes. The union of **X** and **Y** is the class {**u** | **u** $\in$ **X** $\vee$ **u** $\in$ **Y**}.*

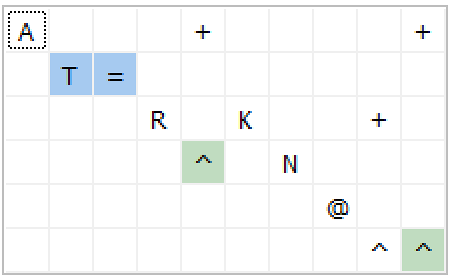
48(F) I definition (intersection)

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

`A(X) A(Y) =(I(X,Y),{u|X(@(u,X),@(u,Y))})`

*Let **X**, **Y** be classes. The intersection of **X** and **Y** is the class {**u** | **u** $\in$ **X** $\wedge$ **u** $\in$ **Y**}.*

49(F) R definition (complement/remainder)




```
A - - - + - - - - +
^ T =
X     R   K - - + -
        ^ ^ N
        X u   @
                ^ ^
                u X
```

`A(X) =(R(X),{u|N(@(u,X))})`

*Let **X** be a class. The complement of **X** is the class {**u** | N(**u** $\in$ **X**)}.*

53(T) U explanation corollary

```
 A - - - - - - - - - + - - - - + - - -
 ^ T A - - - - - - - - + - - - - - - - +
 X   ^ T A - - - + - - - - - - + - - - + -
     Y   ^ T D
         u     @         V
                 ^ U       @     @
                 u   ^ ^     ^ ^   ^ ^
                     X Y     u X   u Y
```

```
A(X) A(Y) A(u) D(@(u,U(X,Y)),V(@(u,X),@(u,Y)))
```

*Let **X**, **Y**, **u** be classes. **u** belongs to the union of **X** and **Y** if and only if **u** $\in$ **X** $\vee$ **u** $\in$ **Y**.*

Instead of the union of X and Y, we will briefly say the union of X, Y. Theorems 54 and 55 are similar explanations for intersection and complement.

56(R) Dis definition (disjoint classes)

```
 A - - - - - + - - - + - -
 ^ T A - - - - + - - - + -
 X   ^ T D
     Y     D     =
           i ^ ^   I     0
           s X Y     ^ ^
                     X Y
A(X) A(Y) D(Dis(X,Y),=(I(X,Y),0))
```

*Let **X**, **Y** be classes. **X** and **Y** are disjoint if and only if their intersection is 0.*

57(T) Dis explanation with set condition

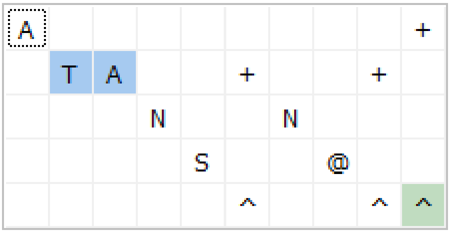
```
 A - - - - - + - - - - - - - - - + - - - -
 ^ T A - - - - + - - - - - - - - - - - - +
 X   ^ T D
     Y     D     A - + - - - + - - - + -
           i ^ ^ ^ S   V
           s X Y u   ^   N       N
                     u     @       @
                             ^ ^     ^ ^
                             u X     u Y
```

```
A(X) A(Y) D(Dis(X,Y),A(u|S(u)) V(N(@(u,X)),N(@(u,Y))))
```

*Let **X**, **Y** be two classes. **X** and **Y** are disjoint if and only if, for any set **u**, **u** does not belong to **X** or **u** does not belong to **Y**.*

We say that a class is a proper class if it is not a set. A proper class does not belong to any class.

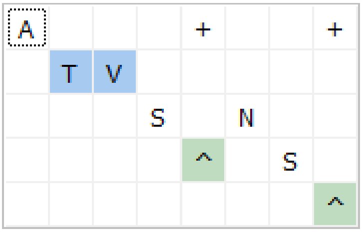
58(T) A proper class belongs to no class




```
A - - - - - - - - +
^ T A - - + - - + -
X   ^ N     N
    U   S     @
          ^     ^ ^
          U     U X
```

`A(X) A(U|N(S(U))) N(@(U,X))`

59(T) A class is a set or a proper class




```
A - - - + - - +
^ T V
X     S   N
        ^   S
        X     ^
              X
```

`A(X) V(S(X),N(S(X)))`

62(T) Dis explanation

```
A - - - - - + - - - - - - - + - - - -
^ T A - - - - + - - - - - - - - - - +
X   ^ T D
    Y     D     A - - - - + - - - + -
          i ^ ^ ^ T V
          s X Y u     N       N
                        @       @
                          ^ ^     ^ ^
                          u X     u Y
```

`A(X) A(Y) D(Dis(X,Y),A(u) V(N(@(u,X)),N(@(u,Y))))`

*Let **X**, **Y** be classes. **X**, **Y** are disjoint if and only if, for any **u**, **u** does not belong to **X** or **u** does not belong to **Y**.*

63(T) Inclusion and union

```
A - - - - - - - - - - + - - - - - - + - - - -
^ T A - - - - - - - - - + - - - - - - - - + -
X   ^ T A - - - - - - - - - + - - - - + - - -
    Y   ^ T A - - - - - - - - + - - - - - - +
        A   ^ T C
            B     X             P
                    P     P     a U     U
                    a ^ ^ a ^ ^ r   ^ ^   ^ ^
                    r X Y r A B t   X A   Y B
                    t     t
```

`A(X) A(Y) A(A) A(B) C(X(Part(X,Y),Part(A,B)),Part(U(X,A),U(Y,B)))`

*Let **X**, **Y**, **A**, **B** be classes. If **X** is included in **Y** and **A** is included in **B**, then the union of **X** and **A** is included in the union of **Y** and **B**.*

64(T) Inclusion and intersection

```
A - - - - - - - - - - + - - - - - - + - - - -
^ T A - - - - - - - - - + - - - - - - - - + -
X   ^ T A - - - - - - - - - + - - - - + - - -
    Y   ^ T A - - - - - - - - + - - - - - - +
        A   ^ T C
            B     X             P
                    P     P     a I     I
                    a ^ ^ a ^ ^ r   ^ ^   ^ ^
                    r X Y r A B t   X A   Y B
                    t     t
```

`A(X) A(Y) A(A) A(B) C(X(Part(X,Y),Part(A,B)),Part(I(X,A),I(Y,B)))`

*Let **X**, **Y**, **A**, **B** be classes. If **X** is included in **Y** and **A** is included in **B**, then the intersection of **X** and **A** is included in the intersection of **Y** and **B**.*

65(T) Inclusion implies transposed inclusion of remainders

```
A - - - - - + - - - - - +
^ T A - - - - + - - + - -
X   ^ T C
    Y     P     P
          a ^ ^ a R   R
          r X Y r   ^   ^
          t     t   Y   X
```

`A(X) A(Y) C(Part(X,Y),Part(R(Y),R(X)))`

*Let **X**, **Y** be classes. If **X** is included in **Y**, then the complement of **Y** is included in the complement of **X**.*

66(T) No Russell's paradox 1

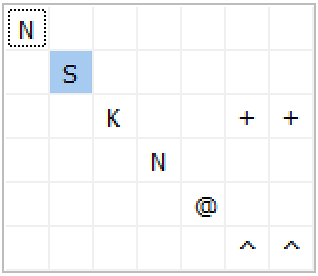
N											
	@										
		K			+	+	K			+	+
			N					N			
				@					@		
					^	^				^	^

```
N
  @
    K - - + + K - - + +
    ^ N       ^ N
    u   @     v   @
          ^ ^       ^ ^
          u u       v v
```

`N(@({u|N(@(u,u))},{v|N(@(v,v))}))`

The class of non-self-contained elements does not belong to itself.

67(T) No Russell's paradox 2

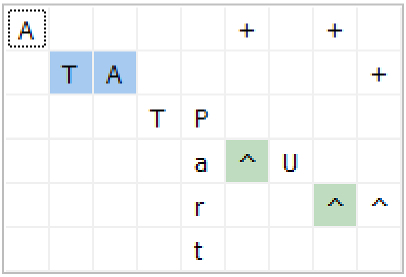



```
N
  S
    K - - + +
    ^ N
    u   @
          ^ ^
          u u
```

```
N(S({u|N(@(u,u))}))
```

The class of non-self-contained elements is not a set.

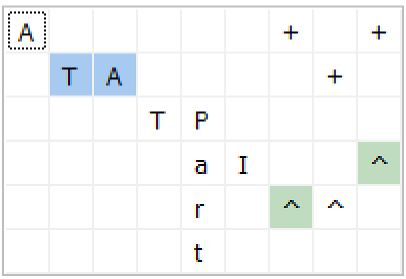
68(T) A class is included in a union of the class




```
A - - - - + - + -
^ T A - - - - - +
X   ^ T P
    Y   a ^ U
        r X   ^ ^
        t     X Y
```

```
A(X) A(Y) Part(X,U(X,Y))
```

69(T) An intersection is included in its first argument




```
A - - - - - + - +
^ T A - - - - + -
X   ^ T P
    Y   a I     ^
        r   ^ ^ X
        t   X Y
```

A(X) A(Y) Part(I(X,Y),X)

*Let **X**, **Y** be classes. The intersection of **X**, **Y** is included in **X**.*

70(T) Union of X, Y included in Y implies X included in Y

```
A - - - - - - + - - - + -
^ T A - - - - - + + - - +
X   ^ T C
    Y     P         P
          a U     ^ a ^ ^
          r   ^ ^ Y r X Y
          t   X Y   t
```

A(X) A(Y) C(Part(U(X,Y),Y),Part(X,Y))

*Let **X**, **Y** be classes. If the union of **X**, **Y** is included in **Y**, then **X** is included in **Y**.*

72(T) U is commutative

A						+				+
	T	A					+		+	
			T	=						
					U			U		
						^	^		^	^

```
A - - - - - + - - - +
^ T A - - - - + - + -
X   ^ T =
    Y     U     U
            ^ ^   ^ ^
            X Y   Y X
```

A(X) A(Y) =(U(X,Y),U(Y,X))

*Let **X**, **Y** be classes. The union of **X**, **Y** is equal to the union of **Y**, **X**.*

73(F) I is commutative

A						+				+
	T	A					+		+	
			T	=						
					I			I		
						^	^		^	^

```
A - - - - - + - - - +
^ T A - - - - + - + -
X   ^ T =
    Y     I     I
            ^ ^   ^ ^
            X Y   Y X
```

A(X) A(Y) =(I(X,Y),I(Y,X))

*Let **X**, **Y** be classes. The intersection of **X**, **Y** is equal to the intersection of **Y**, **X**.*

78(T) X is included in Y if and only if the union is included in Y

```
A - - - - - + - - - + - -
^ T A - - - - + - - - + +
X   ^ T D
    Y     P     P
          a ^ ^ a U     ^
          r X Y r   ^ ^ Y
          t     t   X Y
```

A(X) A(Y) D(Part(X,Y),Part(U(X,Y),Y))

*Let **X**, **Y** be classes. **X** is included in **Y** if and only if the union of **X**, **Y** is included in **Y**.*

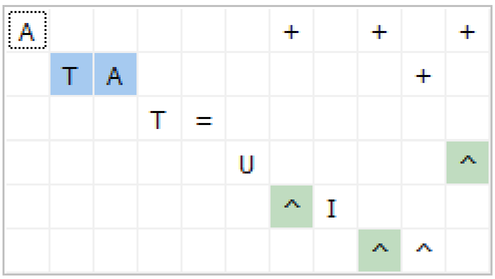
79(T) X is included in Y if and only if X is included in the intersection

```
A - - - - - + - - + - + -
^ T A - - - - + - - - - - +
X   ^ T D
    Y     P     P
          a ^ ^ a ^ I
          r X Y r X   ^ ^
          t     t     X Y
```

`A(X) A(Y) D(Part(X,Y),Part(X,I(X,Y)))`

*Let **X**, **Y** be classes. **X** is included in **Y** if and only if **X** is included in the intersection of **X**, **Y**.*

86(T) Absorption law

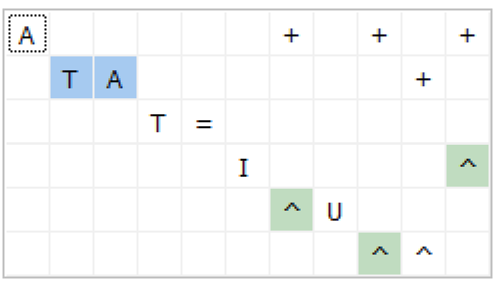



```
A - - - - - + - + - +
^ T A - - - - - - + -
X   ^ T =
    Y     U         ^
            ^ I     X
            X   ^ ^
                X Y
```

`A(X) A(Y) =(U(X,I(X,Y)),X)`

*Let **X**, **Y** be classes. The union of **X** and the intersection of **X** and **Y** is equal to **X**.*

87(T) Absorption law




```
A - - - - - + - + - +
^ T A - - - - - - + -
X   ^ T =
    Y     I         ^
            ^ U     X
            X   ^ ^
                X Y
```

`A(X) A(Y) =(I(X,U(X,Y)),X)`

*Let **X**, **Y** be classes. The intersection of **X** and the union of **X** and **Y** is equal to **X**.*

89(T) DeMorgan's Law

```
A - - - - - - + - - - + - -
^ T A - - - - - + - - - - - +
X   ^ T =
    Y     R       I
            U       R   R
              ^ ^     ^   ^
              X Y     X   Y
```

`A(X) A(Y) =(R(U(X,Y)),I(R(X),R(Y)))`

The complement of a union is the intersection of the complements.

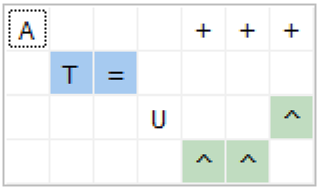
90(T) DeMorgan's Law

```
A - - - - - - + - - - + - -
^ T A - - - - - + - - - - - +
X   ^ T =
    Y     R       U
            I       R   R
              ^ ^     ^   ^
              X Y     X   Y
```

`A(X) A(Y) =(R(I(X,Y)),U(R(X),R(Y)))`

The complement of an intersection is the union of the complements.

91(T) U idempotency

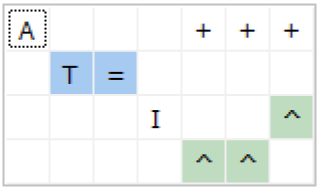



```
A - - - + + +
^ T =
X     U     ^
        ^ ^ X
        X X
```

`A(X) =(U(X,X),X)`

The union of ***X*** *and* ***X*** *is equal to* ***X****.*

92(T) I idempotency




```
A - - - + + +
^ T =
X     I     ^
        ^ ^ X
        X X
```

`A(X) =(I(X,X),X)`

The intersection of ***X*** *and* ***X*** *is equal to* ***X****.*

93(T) U associativity

```
A - - - - - - - + - - - - - + - -
^ T A - - - - - - - + - - - - + -
X   ^ T A - - - - - - + - - - - +
    Y   ^ T =
        Z     U         U
                ^ U       U     ^
                X   ^ ^     ^ ^ Z
                    Y Z     X Y
```

```
A(X) A(Y) A(Z) =(U(X,U(Y,Z)),U(U(X,Y),Z))
```

*Let **X**, **Y**, **Z** be classes. Then **X** U (**Y** U **Z**) = (**X** U **Y**) U **Z**.*

94(T) I associativity

```
A - - - - - - - + - - - - - + - -
^ T A - - - - - - - + - - - - + -
X   ^ T A - - - - - - + - - - - +
    Y   ^ T =
        Z     I         I
                ^ I       I     ^
                X   ^ ^     ^ ^ Z
                    Y Z     X Y
```

```
A(X) A(Y) A(Z) =(I(X,I(Y,Z)),I(I(X,Y),Z))
```

*Let **X**, **Y**, **Z** be classes. The intersection of **X** with the intersection of **Y** and **Z** is equal to the intersection of the intersection of **X** and **Y** with **Z**.*

95(T) U distributivity for I

```
A - - - - - - - + - - - - - + - - + -
^ T A - - - - - - - + - - - - + - - -
X   ^ T A - - - - - - + - - - - - - +
    Y   ^ T =
        Z     U         I
                ^ I       U     U
                X   ^ ^     ^ ^   ^ ^
                    Y Z     X Y   X Z
```

```
A(X) A(Y) A(Z) =(U(X,I(Y,Z)),I(U(X,Y),U(X,Z)))
```

*Let **X**, **Y**, **Z** be classes. The union of **X** and the intersection of **Y**, **Z** is equal to the intersection of the union of **X**, **Y** and the union of **X**, **Z**.*

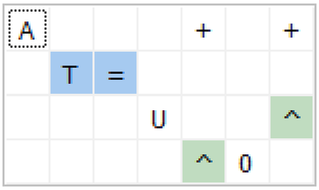
96(T) I distributivity for U

```
A - - - - - - - + - - - - - + - - + -
^ T A - - - - - - - + - - - - + - - -
X   ^ T A - - - - - - + - - - - - - +
    Y   ^ T =
        Z     I         U
                ^ U       I     I
                X   ^ ^     ^ ^   ^ ^
                    Y Z     X Y   X Z
```

```
A(X) A(Y) A(Z) =(I(X,U(Y,Z)),U(I(X,Y),I(X,Z)))
```

*Let **X**, **Y**, **Z** be classes. The intersection of **X** and the union of **Y**, **Z** is equal to the union of the intersection of **X**, **Y** and the intersection of **X**, **Z**.*

97(T) The union of X and 0 is X

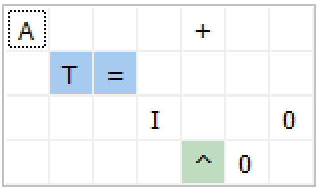



```
A - - - + - +
^ T =
X     U     ^
        ^ 0 X
        X
```

```
A(X) =(U(X,0),X)
```

*The union of **X** and 0 is equal to **X**.*

98(T) The intersection of X and 0 is 0

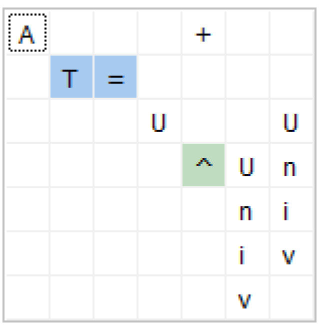



```
A - - - + - -
^ T =
X     I     0
        ^ 0
        X
```

```
A(X) =(I(X,0),0)
```

*The intersection of **X** and 0 is equal to 0.*

99(T) The union of X and Univ (universal class) is Univ




```
A - - - + - -
^ T =
X     U     U
        ^ U n
        X n i
          i v
          v
```

```
A(X) =(U(X,Univ),Univ)
```

The union of a class with the universal class is equal to the universal class.

100(T) The intersection of X and Univ (universal class) is X

A			+		+	
	T	=				
			I			^
				^	U	
					n	
					i	
					v	

```
A - - - + - +
^ T =
X     I     ^
        ^ U X
        X n
          i
          v
```

`A(X) =(I(X,Univ),X)`

*The intersection of **X** with the universal class is equal to **X**.*

101(T) The complement of the universal class is the empty class

=			
	R		0
		U	
		n	
		i	
		v	

```
=
  R   0
    U
    n
    i
    v
```

`=(R(Univ),0)`

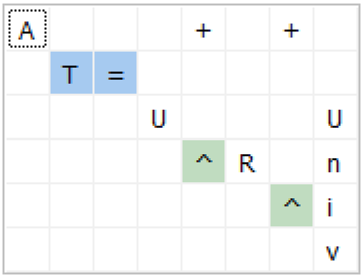
102(T) The complement of the empty class is the universal class

=			
	R		U
		0	n
			i
			v

```
=
  R   U
    0 n
      i
      v
```

`=(R(0),Univ)`

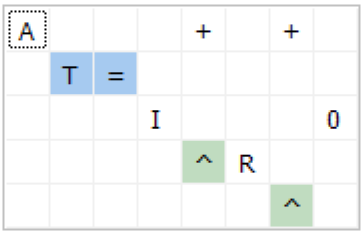
103(T) The union of X and the remainder of X is the universal class



```
A - - - + - + -
^ T =
X     U       U
        ^ R   n
        X   ^ i
            X v
```

A(X) =(U(X,R(X)),Univ)

104(T) The intersection of X and the remainder of X is the empty class



```
A - - - + - + -
^ T =
X     I       0
        ^ R
        X   ^
            X
```

A(X) =(I(X,R(X)),0)

108(T) Sng equality corollary

```
A - + - - - - - - + - - - + -
^ S   A - + - - - - - + - - +
x   ^ ^ S   D
    x y   ^   =         =
          y     S   S     ^ ^
                n ^ n ^   x y
                g x g y
```

A(x|S(x)) A(y|S(y)) D(=(Sng(x),Sng(y)),=(x,y))

*Let **x**, **y** be sets. {**x**} = {**y**} if and only if **x** = **y**.*

109(T) Sng to Dbl equality

```
A - + - - - - - - - - - + - - - - - + - - + -
^ S   A - + - - - - - - - - + - - - - + - - -
a   ^ ^ S   A - + - - - - - - + - - - - - - +
    a x   ^ ^ S   D
          x y   ^   =           X
                y     S   D       =     =
                      n ^ b ^ ^     ^ ^   ^ ^
                      g a l x y     a x   a y
```

A(a|S(a)) A(x|S(x)) A(y|S(y)) D(=(Sng(a),Dbl(x,y)),X(=(a,x),=(a,y)))

*Let **a**, **x**, **y** be sets. {**a**} = {**x**, **y**} if and only if **a** = **x** and **a** = **y**.*

The notion of function is often "defined" using the words assignment, mapping, rule, law, or other similar words, which give the impression of depth but actually say nothing.

An important notion in mathematics is the ordered pair. To define it, we will use the notions of singleton and doubleton. Based on the notion of ordered pair, we will define the Cartesian product, graphs, graph functions, and functions.

What mysterious forces are hidden in the notion of ordered pair, which can replace all kinds of expressions dear to mathematicians when explaining the notion of function? The answer is simple: the notion of ordered pair represents exactly that; it is a connection (assignment, mapping, rule, law, ...) between x and y, with the difference that there is nothing mysterious and obscure anymore. A simple, elegant thing; maybe we should even say it's beautiful!

110(F) Pair definition (ordered pair)

```
A - + - - - - - + - - - + - + -
^ S   A - + - - - + - - - - - +
x     ^ ^ S   =
      x y   ^   P     D
            y   a ^ ^ b S   D
                i x y l n ^ b ^ ^
                r         g x l x y
```

```
A(x|S(x)) A(y|S(y)) =(Pair(x,y),Dbl(Sng(x),Dbl(x,y)))
```

*Let **x**, **y** be sets. (**x**, **y**) = {{**x**}, {**x**, **y**}}*

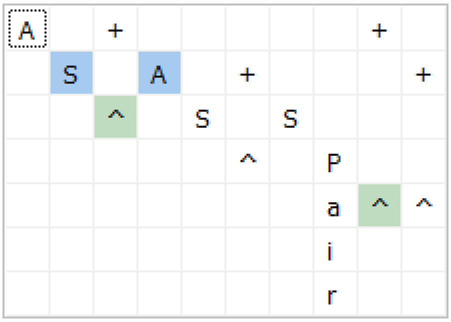
111(T) Pair explanation

```
A - + - - - - - - - - - - + - - - - - + - - - + -
^ S   A - + - - - - - - - - + - - - - - - - - - +
x     ^ ^ S   A - + - - + - - - - - + - - - + - - -
      x y   ^ ^ S   D
            y u   ^   @           V
                  u     ^ P         =       =
                        u a ^ ^       ^ S     ^ D
                          i x y     u n ^   u b ^ ^
                          r           g x     l x y
```

```
A(x|S(x)) A(y|S(y)) A(u|S(u)) D(@(u,Pair(x,y)),V(=(u,Sng(x)),=(u,Dbl(x,y))))
```

*Let **x**, **y**, **u** be sets. **u** belongs to Pair of **x**, **y** if and only if **u** = {**x**} or **u** = {**x**, **y**}.*

114(T) Pair is a set




```
A - + - - - - - + -
^ S   A - + - - - +
x   ^ ^ S   S
    x y   ^   P
          y   a ^ ^
              i x y
              r
```

`A(x|S(x)) A(y|S(y)) S(Pair(x,y))`

A pair of two sets is a set.

Two pairs are equal if and only if the first argument of the first pair is equal to the first argument of the second pair, and the second argument of the first pair is equal to the second argument of the second pair.

117(T) Pair equality

```
A - + - - - - - - - - - - - - + - - - - - - + - - - -
^ S   A - + - - - - - - - - - - + - - - - - - - - + -
x   ^ ^ S   A - + - - - - - - - - - + - - - - + - - -
    x y   ^ ^ S   A - + - - - - - - - + - - - - - - +
          y a   ^ ^ S   D
                a b   ^   =             X
                      b     P     P       =     =
                            a ^ ^ a ^ ^     ^ ^   ^ ^
                            i x y i a b     x a   y b
                            r     r
```

```
A(x|S(x)) A(y|S(y)) A(a|S(a)) A(b|S(b))
D(=(Pair(x,y),Pair(a,b)),X(=(x,a),=(y,b)))
```

*Let **x**, **y**, **a**, **b** be sets. The pair of **x**, **y** is equal to the pair of **a**, **b** if and only if **x** = **a** and **y** = **b**.*

Cartesian product

118(F) Cart definition (Cartesian product)

```
A - - - - - + - - - - - + - - - - - - - - -
^ T A - - - - + - - - - - - - - + - - - - -
X   ^ T =
    Y     C     K - - - - - - - - - + - - -
          a ^ ^ ^ E - + - - - - - - - - + -
          r X Y u ^ @     E - + - - - - - +
          t       x   ^ ^ ^ @     =
                      x X y   ^ ^   ^ P
                              y Y   u a ^ ^
                                      i x y
                                      r
```

```
A(X) A(Y) =(Cart(X,Y),{u|E(x|@(x,X)) E(y|@(y,Y)) =(u,Pair(x,y))})
```

*Let **X**, **Y** be classes. Their Cartesian product is the class of the elements **u** such that there exists **x** belonging to **X** and **y** belonging to **Y** such that **u** is the ordered pair of **x**, **y**.*

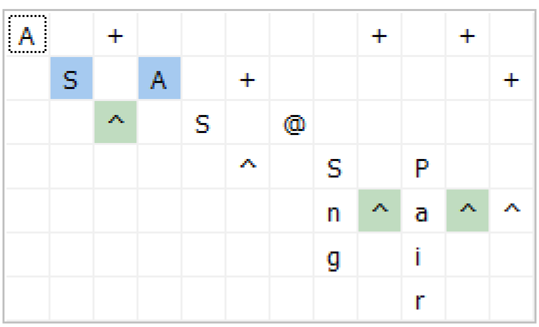
119(T) Cart explanation

```
A - - - - - - - - - + - - - - + - - - - - - - - -
^ T A - - - - - - - - + - - - - - - - + - - - - -
X   ^ T A - - - + - - - - - - - - - - - - + - - -
    Y   ^ T D
        u     @         E - + - - - - - - - - + -
                ^ C     ^ @     E - + - - - - - +
                u a ^ ^ x   ^ ^ ^ @     =
                  r X Y     x X y   ^ ^   ^ P
                  t                 y Y   u a ^ ^
                                            i x y
                                            r
```

```
A(X) A(Y) A(u) D(@(u,Cart(X,Y)),E(x|@(x,X)) E(y|@(y,Y)) =(u,Pair(x,y)))
```

*Let **X**, **Y**, **u** be classes. **u** belongs to Cart(**X**, **Y**) if and only if there exists **x** belonging to **X** and **y** belonging to **Y** such that **u** is the ordered pair of **x**, **y**.*

121(T) Sng of x belongs to Pair of x, y



```
A - + - - - - - + - + -
^ S   A - + - - - - - +
x   ^ ^ S   @
    x y   ^   S   P
          y   n ^ a ^ ^
              g x i x y
                  r
```

```
A(x|S(x)) A(y|S(y)) @(Sng(x),Pair(x,y))
```

*Let **x**, **y** be sets. Sng(**x**) belongs to Pair(**x**, **y**).*

Since the work in our system is done by using the matrix form of the statements, we have so far presented this form together with the text form. We consider that the reader has become familiar enough with the two forms. From here, we will only use the text form with variable names. This form is the best connection between the usual expression and the matrix form.

122(T) Dbl of x, y belongs to Pair of x, y

```
A - + - - - - - + - - + -
^ S   A - + - - - + - - +
x     ^ ^ S   @
      x y   ^   D     P
            y   b ^ ^ a ^ ^
                l x y i x y
                      r
```

```
A(x|S(x)) A(y|S(y)) @(Dbl(x,y),Pair(x,y))
```

*Let **x**, **y** be sets. Dbl(**x**, **y**) belongs to Pair(**x**, **y**).*

123(T) Cart of Univ, Univ corollary

```
A - + - - - - - - - - - - + - - -
^ @         E - + - - - - - - + -
u   ^ C     ^ S   E - + - - - - +
    u a U U x   ^ ^ S   =
      r n n     x y   ^   ^ P
      t i i           y   u a ^ ^
        v v                 i x y
                            r
```

```
A(u|@(u,Cart(Univ,Univ))) E(x|S(x)) E(y|S(y)) =(u,Pair(x,y))
```

*For any **u** belonging to the Cartesian product of Univ and Univ, there exist the sets **x** and **y** such that **u** is the ordered pair of **x** and **y**.*

Any element of the Cartesian product of Univ and Univ is an ordered pair having a first and a second coordinate. This allows us to define two (primitive) functions, Coor1 and Coor2.

124(T) Existence for Coor1

```
A - + - - - - - - - - - - - +
^ @         E - - - + - - + -
u   ^ C     ^ T X
    u a U U x     S   @
      r n n         ^   S   ^
      t i i         x   n ^ u
        v v             g x
```

```
A(u|@(u,Cart(Univ,Univ))) E(x) X(S(x),@(Sng(x),u))
```

*For any **u** belonging to the Cartesian product of Univ and Univ, there exists **x** such that **x** is a set and Sng(**x**) belongs to **u**.*

126(T) Uniqueness for Coor1

```
A - + - - - - - - - - - - - + - - - - - - - - + - - -
^ @         A - - + - - + - - - - - - - - - - - + -
u   ^ C     ^ X             A - - + - - + - - - - +
    u a U U x   S   @       ^ X                 =
      r n n 1     ^   S   ^ x   S   @         ^ ^
      t i i       x   n ^ u 2     ^   S   ^   x x
        v v       1   g x         x   n ^ u   1 2
                        1         2   g x
                                        2
```

A(u|@(u,Cart(Univ,Univ))) A(x1|X(S(x1),@(Sng(x1),u))) A(x2|X(S(x2),@(Sng(x2),u))) =(x1,x2)

*For any **u** belonging to the Cartesian product of Univ and Univ, and for any set $\boldsymbol{x_1}$ with Sng($\boldsymbol{x_1}$) in u, and for any set $\boldsymbol{x_2}$ with Sng($\boldsymbol{x_2}$) in **u**, $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ are equal.*

127(F) Coor1 definition

```
A - + - - - - - - - + - - - + +
^ @         X
u   ^ C       S     @
    u a U U     C     S     ^
      r n n     o ^   n C   u
      t i i     o u   g o ^
        v v     r       o u
                1       r
                        1
```

A(u|@(u,Cart(Univ,Univ))) X(S(Coor1(u)),@(Sng(Coor1(u)),u))

*For any **u** belonging to the Cartesian product of Univ and Univ, Coor1(**u**) is a set, and the set formed by Coor1(**u**) belongs to **u**.*

129(T) Existence for Coor2

```
A - + - - - - - - - - - + - - + -
^ @         E - - - + - - - - - +
u   ^ C     ^ T X
    u a U U y     S   =
      r n n         ^   ^ P
      t i i         y   u a C   ^
        v v               i o ^ y
                          r o u
                            r
                            1
```

A(u|@(u,Cart(Univ,Univ))) E(y) X(S(y),=(u,Pair(Coor1(u),y)))

*For any **u** belonging to the Cartesian product of Univ and Univ, there exists a set **y** such that **u** is equal to the ordered pair of Coor1(**u**) and **y**.*

130(T) Uniqueness for Coor2

```
A - + - - - - - - - - + - - + - - - - - - + - - + - - - -
^ @         A - - + - - - - - + - - - - - - - - - - - + -
u   ^ C     ^ X                 A - - + - - - - - + - - +
    u a U U y   S   =           ^ X                 =
      r n n 1     ^   ^ P       y   S   =             ^ ^
      t i i       y   u a C   ^ 2     ^   ^ P         y y
        v v       1     i o ^ y       y   u a C   ^   1 2
                        r o u 1       2     i o ^ y
                          r                 r o u 2
                          1                   r
                                              1
```

A(u|@(u,Cart(Univ,Univ))) A(y1|X(S(y1),=(u,Pair(Coor1(u),y1))))
A(y2|X(S(y2),=(u,Pair(Coor1(u),y2)))) =(y1,y2)

*For any **u** belonging to the Cartesian product of Univ and Univ, and for any set* $\boldsymbol{y_1}$ *with **u** equal to the pair of Coor1(**u**) and* $\boldsymbol{y_1}$*, and for any set* $\boldsymbol{y_2}$ *with **u** equal to the pair of Coor1(**u**) and* $\boldsymbol{y_2}$*,* $\boldsymbol{y_1}$ *is equal to* $\boldsymbol{y_2}$*.*

131(F) Coor2 definition

```
A - + - - - - - - + - + - - + - +
^ @         X
u   ^ C       S     =
    u a U U     C     ^ P
      r n n     o ^   u a C   C
      t i i     o u     i o ^ o ^
        v v     r       r o u o u
                2         r   r
                          1   2
```

A(u|@(u,Cart(Univ,Univ))) X(S(Coor2(u)),=(u,Pair(Coor1(u),Coor2(u))))

*For any **u** belonging to the Cartesian product of Univ and Univ, Coor2(**u**) is a set, and **u** is equal to the pair of Coor1(**u**) and Coor2(**u**).*

If a set belongs to a class that is included in a second class, then the set also belongs to the second class. Since the situation appears in many proofs, we considered it useful to dedicate a theorem to it.

132(T) Belonging and inclusion

```
A - - - - - - - - - + - + - - - -
^ T A - - - - - - - - - - + - - +
X   ^ T A - - - - + - - - - - + -
    Y   ^ T C
        x     X             @
                @     P       ^ ^
                  ^ ^ a ^ ^   x Y
                  x X r X Y
                      t
```

A(X) A(Y) A(x) C(X(@(x,X),Part(X,Y)),@(x,Y))

*Let **X**, **Y**, **x** be classes. If **x** belongs to **X** and **X** is included in **Y**, then **x** belongs to **Y**.*

136(T) Cart explanation

```
A - - - - - - - - - - - - - - - + - - - - + - - -
^ T A - - - - - - - - - - - - - - + - - - - - - +
X   ^ T A - + - - - - - - + - - - - - - + - - - -
    Y   ^ S   A - + - - - - + - - - - - - - - + -
        x   ^ ^ S   D
            x y   ^   @             X
                  y     P     C       @     @
                        a ^ ^ a ^ ^     ^ ^   ^ ^
                        i x y r X Y     x X   y Y
                        r     t
```

A(X) A(Y) A(x|S(x)) A(y|S(y)) D(@(Pair(x,y),Cart(X,Y)),X(@(x,X),@(y,Y)))

*Let **X**, **Y** be classes and **x**, **y** sets. The pair of **x**, **y** belongs to the Cartesian product of **X**, **Y** if and only if **x** belongs to **X** and **y** belongs to **Y**.*

140(T) Cart inclusion

```
A - - - - - - - - - - + - - - - - - + - - - -
^ T A - - - - - - - - - - - + - - - - + - - -
X   ^ T A - - - - - - - + - - - - - - - - + -
    Y   ^ T A - - - - - - - - + - - - - - - +
        A   ^ T C
            B     X             P
                    P     P     a C     C
                    a ^ ^ a ^ ^ r a ^ ^ a ^ ^
                    r X A r Y B t r X Y r A B
                    t     t       t     t
```

A(X) A(Y) A(A) A(B) C(X(Part(X,A),Part(Y,B)),Part(Cart(X,Y),Cart(A,B)))

*Let **X**, **Y**, **A**, **B** be classes. If **X** is included in **A** and **Y** is included in **B**, then the Cartesian product of **X**, **Y** is included in the Cartesian product of **A**, **B**.*

142(T) Coordinates theorem

```
A - + - - - - - - - + - - + - + - - + - - +
^ @         X
u   ^ C       X             =
    u a U U     S     S       ^ P
      r n n       C     C     u a C   C
      t i i       o ^   o ^     i o ^ o ^
        v v       o u   o u     r o u o u
                  r     r         r   r
                  1     2         1   2
```

A(u|@(u,Cart(Univ,Univ)))
X(X(S(Coor1(u)),S(Coor2(u))),=(u,Pair(Coor1(u),Coor2(u))))

*Let **u** be an element of the Cartesian product of Univ and Univ. The coordinates of **u** are sets, and **u** is equal to the ordered pair of its coordinates.*

In addition to the union and intersection of two classes, we can also define the difference of two classes.

144(F) Difference of two classes (X\Y)

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

```
A(X) A(Y) =(\(X,Y),{u|X(@(u,X),N(@(u,Y)))})
```

*Let **X**, **Y** be classes. The difference of **X** and **Y** (do not say “between **X** and **Y**“!) is the class of all **u** belonging to **X** but not to **Y**.*

146(T) \ explanation without set condition

```
A - - - - - - - - - + - - - - + - - - -
^ T A - - - - - - - - + - - - - - - - +
X   ^ T A - - - + - - - - - + - - - + -
    Y   ^ T D
        u     @         X
                ^ \       @     N
                u   ^ ^     ^ ^   @
                    X Y     u X     ^ ^
                                    u Y
```

```
A(X) A(Y) A(u) D(@(u,\(X,Y)),X(@(u,X),N(@(u,Y))))
```

*Let **X**, **Y**, **u** be classes. **u** belongs to the difference of **X**, **Y** if and only if **u** belongs to **X** but not to **Y**.*

147(T) U of \ and I

```
A - - - - + - - + - - + -
^ T A - - - - - - - + - - +
X   ^ T =
    Y     ^ U
          X   \     I
                ^ ^   ^ ^
                X Y   X Y
```

```
A(X) A(Y) =(X,U(\(X,Y),I(X,Y)))
```

*Let **X**, **Y** be classes. The union of \(**X**,**Y**) and I(**X**,**Y**) is equal to **X**.*

148(T) Dis of \ and I

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

```
A(X) A(Y) Dis(\(X,Y),I(X,Y))
```

*Let **X**, **Y** be classes. The difference of **X** and **Y** and the intersection of **X** and **Y** are disjoint.*

153(T) Inclusion and inequality implies \ is not null

```
A - - - - - - - + - - + - - - - - - - + -
^ T A - - - - + - - - - + - - - - - - - +
X   ^ T C
    Y     X               E - - + - - -
            P     N       ^ T @
            a ^ ^   =     x     ^ \
            r Y X     ^ ^       x   ^ ^
            t         X Y           X Y
```

A(X) A(Y) C(X(Part(Y,X),N(=(X,Y))),E(x) @(x,\(X,Y)))

*Let **X**, **Y** be classes. If **Y** is included in **X** and **X**, **Y** are not equal, then there exists an **x** belonging to the difference of **X** and **Y**.*

156(T) \(X, Y) and Y are disjoint

```
A - - - - - + - -
^ T A - - - - + +
X   ^ T D
    Y   i \     ^
        s   ^ ^ Y
            X Y
```

A(X) A(Y) Dis(\(X,Y),Y)

*Let **X**, **Y** be classes. \(**X**, **Y**) and **Y** are disjoint.*

Graphs

One of the most important definitions, which uses the notion of ordered pair, is the notion of a graph. A graph is simply a class of ordered pairs. We will define the inverse, the domain, and the range of a graph, as well as the composite of two graphs, and will study their properties. In addition to these well-known properties, we will define at the level of graphs properties specific to functions (direct image, reverse image).

161(R) Graph definition

```
A - - - + - + - - -
^ T D
G     G   P
      r ^ a ^ C
      a G r G a U U
      p   t   r n n
      h       t i i
                v v
```

A(G) D(Graph(G),Part(G,Cart(Univ,Univ)))

A graph is a part of the Cartesian product of Univ and Univ.

In mathematics, the definition of a class can be done in the following way: {f(x) | p(x)}, where f is a function and p is a predicate of variable x. In our system, the logical operator "K" (class generator) allows only the simpler form: {x | p(x)}. This is not a lack but a consistent application of the principles of our system and contributes to simplifying the action of the inference operators "Class generator to form" and "Form to class generator".

162(T) Graph corollary

```
A - + - - - + - - - - - - - - - - -
^ G   A - + - - - - - - - - + - - -
G r ^ ^ @     E - + - - - - - - + -
  a G u   ^ ^ ^ S   E - + - - - - +
  p       u G x   ^ ^ S   =
  h               x y   ^   ^ P
                        y   u a ^ ^
                              i x y
                              r
```

A(G|Graph(G)) A(u|@(u,G)) E(x|S(x)) E(y|S(y)) =(u,Pair(x,y))

*For any graph **G** and any element **u** of **G**, there exist the sets **x** and **y** such that **u** is the ordered pair of **x**, **y**.*

Expressing the inclusion of graphs using ordered pairs simplifies many proofs.

167(T) Graph inclusion

```
A - + - - - - - + - - - - - - - - - - - - + - - - - -
^ G   A - + - - - + - - - - - - - - - - - - - - - - +
G r ^ ^ G   D
  a G H r ^   P     A - + - - - - - - + - - - - + - -
  p     a H   a ^ ^ ^ S   A - + - - - - + - - - - + -
  h     p     r G H x   ^ ^ S   C
        h     t         x y   ^   @         @
                              y     P     ^   P     ^
                                    a ^ ^ G   a ^ ^ H
                                    i x y     i x y
                                    r         r
```

A(G|Graph(G)) A(H|Graph(H)) D(Part(G,H),A(x|S(x)) A(y|S(y))
C(@(Pair(x,y),G),@(Pair(x,y),H)))

*Let **G**, **H** be graphs. **G** is included in **H** if and only if, for any sets **x** and **y**, the belonging of Pair(**x**, **y**) to **G** implies the belonging of Pair(**x**, **y**) to **H**.*

168(T) Graph equality

```
A - + - - - - - + - - - - - - - - - - - - + - - - - -
^ G   A - + - - - + - - - - - - - - - - - - - - - - - +
G r ^ ^ G   D
  a G H r ^   =     A - + - - - - - - - + - - - - + - -
  p     a H     ^ ^ ^ S   A - + - - - - + - - - - - + -
  h     p       G H x   ^ ^ S   D
        h               x y   ^   @         @
                              y     P     ^   P     ^
                                    a ^ ^ G   a ^ ^ H
                                    i x y     i x y
                                    r         r
```

```
A(G|Graph(G)) A(H|Graph(H)) D(=(G,H),A(x|S(x)) A(y|S(y))
D(@(Pair(x,y),G),@(Pair(x,y),H)))
```

*Let **G**, **H** be graphs. **G** = **H** if and only if, for any sets **x** and **y**, the belonging of Pair(**x**, **y**) to **G** is equivalent to the belonging of Pair(**x**, **y**) to **H**.*

169(T) Cart is Graph

```
A - - - - - + -
^ T A - - - - +
X   ^ T G
    Y   r C
        a a ^ ^
        p r X Y
        h t
```

```
A(X) A(Y) Graph(Cart(X,Y))
```

The Cartesian product of two classes is a graph.

170(T) Part of Graph

```
A - - - - - - + - - +
^ T A - + - - - + - -
X   ^ G   C
    G r ^   P     G
      a G   a ^ ^ r ^
      p     r X G a X
      h     t     p
                  h
```

```
A(X) A(G|Graph(G)) C(Part(X,G),Graph(X))
```

A part of a graph is a graph.

171(T) The intersection of a graph with a class is a graph

```
A - + - - - - + -
^ G   A - - - - +
G r ^ ^ T G
  a G X   r I
  p       a   ^ ^
  h       p   G X
          h
```

```
A(G|Graph(G)) A(X) Graph(I(G,X))
```

176(R) Inv definition (Definition of the inverse of a graph)

```
 A - + - - + - - - - - - - - - - - - - + - - - - -
 ^ G   =
 G r ^   I   K - - - - - - - - - - - - - - - + - - -
   a G   n ^ ^ E - + - - - - - - - + - - - - - - - +
   p     v G u ^ S   E - + - - - - - + - - - - - + -
   h           x   ^ ^ S   X
                   x y   ^   @         =
                         y     P     ^   ^ P
                               a ^ ^ G   u a ^ ^
                               i x y       i y x
                               r           r
```

```
A(G|Graph(G)) =(Inv(G),{u|E(x|S(x)) E(y|S(y)) X(@(Pair(x,y),G),=(u,Pair(y,x)))})
```

*The inverse of a graph consists of the pairs of **y**, **x**, where the pair of **x**, **y** belongs to the graph.*

178(T) Inv explanation

```
 A - + - - - - - - - - - - - - - + - - - - - +
 ^ G   A - + - - - - - - - + - - - - - - - + -
 G r ^ ^ S   A - + - - - - - + - - - - - + - -
   a G x   ^ ^ S   D
   p       x y   ^   @           @
   h             y     P     I     P       ^
                       a ^ ^ n ^   a ^ ^ G
                       i x y v G   i y x
                       r           r
```

```
A(G|Graph(G)) A(x|S(x)) A(y|S(y)) D(@(Pair(x,y),Inv(G)),@(Pair(y,x),G))
```

*Let **G** be a graph and **x**, **y** sets. The pair of **x**, **y** belongs to Inv(**G**) if and only if the pair of **y**, **x** belongs to **G**.*

180(T) Inv of Graph is Graph

```
 A - + - - +
 ^ G   G
 G r ^ r I
   a G a n ^
   p   p v G
   h   h
```

```
A(G|Graph(G)) Graph(Inv(G))
```

The inverse of a graph is a graph.

181(T) Inv of Inv of a graph is the graph

```
A - + - - - + +
^ G   =
G r ^   I     ^
  a G   n I   G
  p     v n ^
  h       v G
```

```
A(G|Graph(G)) =(Inv(Inv(G)),G)
```

The inverse of the inverse of a graph is the graph.

182(F) Composite of graphs definition

```
A - + - - - - - + - - - - - - - - - - - - - - - - - - - - - - + - - - - -
^ G   A - + - - - + - - - - - - - - - - - - - - - - - - - - - - - - - - +
G r ^ ^ G   =
  a G H r ^   o     K - - - - - - - - - - - + - - - - - - - - - - - - - -
  p     a H     ^ ^ ^ E - + - - - - - - - - - - + - - - - + - - - - - - -
  h     p       G H u ^ S   E - + - - - - - - - - - - - - - + - - - + - -
        h             x   ^ ^ S   E - + - - - - - + - - - - - - - - - + -
                          x y   ^ ^ S   X
                                y z   ^   =         X
                                      z     ^ P       @         @
                                            u a ^ ^     P     ^   P     ^
                                              i x z     a ^ ^ G   a ^ ^ H
                                              r         i x y     i y z
                                                        r         r
```

```
A(G|Graph(G)) A(H|Graph(H)) =(o(G,H),{u|E(x|S(x)) E(y|S(y)) E(z|S(z))
X(=(u,Pair(x,z)),X(@(Pair(x,y),G),@(Pair(y,z),H)))})
```

*Let **G**, **H** be graphs. The composite of **G** and **H** is the set of all elements **u** for which there exist the sets **x**, **y**, **z**, such that **u** is the pair of **x**, **z**, and the pair of **x**, **y** belongs to **G**, and the pair of **y**, **z** belongs to **H**.*

The usual notation of o(G, H) is H o G.

186(T) Composite of graphs is a graph

```
A - + - - - - - + -
^ G   A - + - - - +
G r ^ ^ G   G
  a G H r ^ r o
  p     a H a   ^ ^
  h     p   p   G H
        h   h
```

```
A(G|Graph(G)) A(H|Graph(H)) Graph(o(G,H))
```

The composite of graphs is a graph.

188(T) Composite associativity

```
A - + - - - - - - - - - + - - - + - - -
^ G   A - + - - - - - - - + - - - - + -
X r ^ ^ G   A - + - - - - - + - - - - +
  a X Y r ^ ^ G   =
  p     a Y Z r ^   o         o
  h     p     a Z     o     ^   ^ o
        h     p         ^ ^ Z   X   ^ ^
              h         X Y         Y Z
```

```
A(X|Graph(X)) A(Y|Graph(Y)) A(Z|Graph(Z)) =(o(o(X,Y),Z),o(X,o(Y,Z)))
```

*In the usual notation: **Z** o (**Y** o **X**) = (**Z** o **Y**) o **X**.*

189(T) Inv of composite is composite of Inv

```
A - + - - - - - - + - - - - - +
^ G   A - + - - - - + - - + - -
G r ^ ^ G   =
  a G H r ^   I       o
  p     a H   n o       I   I
  h     p     v   ^ ^   n ^ n ^
        h         G H   v H v G
```

```
A(G|Graph(G)) A(H|Graph(H)) =(Inv(o(G,H)),o(Inv(H),Inv(G)))
```

190(F) Dom definition (Domain of a graph)

```
A - + - - + - - - - - - - - +
^ G   =
G r ^   D   K - - - - - + - -
  a G   o ^ ^ E - + - - - + -
  p     m G x ^ S   @
  h           y   ^   P     ^
                  y   a ^ ^ G
                      i x y
                      r
```

```
A(G|Graph(G)) =(Dom(G),{x|E(y|S(y)) @(Pair(x,y),G)})
```

*Let **G** be a graph. Dom of **G** is the class of all elements **x** for which there exists a set **y** such that the pair of **x**, **y** belongs to **G**.*

191(F) Ran definition (Range of a graph)

```
A - + - - + - - - - - - - - +
^ G   =
G r ^   R   K - - - - - - + -
  a G   a ^ ^ E - + - - + - -
  p     n G y ^ S   @
  h           x   ^   P     ^
                  x   a ^ ^ G
                      i x y
                      r
```

```
A(G|Graph(G)) =(Ran(G),{y|E(x|S(x)) @(Pair(x,y),G)})
```

*Let **G** be a graph. Ran of **G** is the class of all elements **y** for which there exists a set **x** such that the pair of **x, y** belongs to **G**.*

195(T) Dom is Ran of Inv

```
A - + - - + - - +
^ G   =
G r ^   D   R
  a G   o ^ a I
  p     m G n n ^
  h           v G
```

```
A(G|Graph(G)) =(Dom(G),Ran(Inv(G)))
```

196(T) Ran is Dom of Inv

```
A - + - - + - - +
^ G   =
G r ^   R   D
  a G   a ^ o I
  p     n G m n ^
  h           v G
```

```
A(G|Graph(G)) =(Ran(G),Dom(Inv(G)))
```

198(T) Dom of comp of two graphs is included in Dom of the first graph

```
A - + - - - - - - + - - +
^ G   A - + - - - - + - -
G r ^ ^ G   P
  a G H r ^ a D       D
  p     a H r o o     o ^
  h     p   t m   ^ ^ m G
        h         G H
```

```
A(G|Graph(G)) A(H|Graph(H)) Part(Dom(o(G,H)),Dom(G))
```

200(T) Ran of comp of two graphs is included in Ran of the second graph

```
A - + - - - - - - + - - -
^ G   A - + - - - - + - +
G r ^ ^ G   P
  a G H r ^ a R       R
  p     a H r a o     a ^
  h     p   t n   ^ ^ n H
        h         G H
```

```
A(G|Graph(G)) A(H|Graph(H)) Part(Ran(o(G,H)),Ran(H))
```

208(T) Dom, Ran corollary for composite

```
A - + - - - - - - - + - - - - + - - + -
^ G   A - + - - - - - - + - - - - - - - +
G r ^ ^ G   C
  a G H r ^   P         P
  p     a H   a R   D   a D   D
  h     p     r a ^ o ^ r o ^ o o
        h     t n G m H t m G m   ^ ^
                                  G H
```

```
A(G|Graph(G)) A(H|Graph(H)) C(Part(Ran(G),Dom(H)),Part(Dom(G),Dom(o(G,H))))
```

*Let **G**, **H** be graphs. If the Ran of **G** is included in the Dom of **H**, then the Dom of **G** is included in the Dom of the composite of **G** and **H**.*

209(T) Part of Ran, Dom for composite

```
A - + - - - - - - - + - - - - - - + - - +
^ G   A - + - - - - - - + - - - - - + - -
G r ^ ^ G   C
  a G H r ^   P         =
  p     a H   a R   D     D         D
  h     p     r a ^ o ^   o o       o ^
        h     t n G m H   m     ^ ^ m G
                                G H
```

```
A(G|Graph(G)) A(H|Graph(H)) C(Part(Ran(G),Dom(H)),=(Dom(o(G,H)),Dom(G)))
```

*Let **G**, **H** be graphs. If the Ran of **G** is included in the Dom of **H**, then the Dom of the composite of **G** and **H** is equal to the Dom of **G**.*

211(T) Inv of U is U of Inv

```
A - + - - - - - - - + - - - + - -
^ G   A - + - - - - + - - - - - +
G r ^ ^ G   =
  a G H r ^   I       U
  p     a H   n U       I   I
  h     p     v   ^ ^   n ^ n ^
        h         G H   v G v H
```

```
A(G|Graph(G)) A(H|Graph(H)) =(Inv(U(G,H)),U(Inv(G),Inv(H)))
```

The inverse of the union of two graphs is the union of the inverses of the graphs.

Generalized union and intersection

We now will present the definitions of the power set (Parts), the generalized union (GU), and the generalized intersection (GI), to which we add the axiom of the power set and the axiom of the generalized union. It immediately follows that the Cartesian product of two sets is a set, and the union of two sets is a set.

224(F) Parts definition (power set)

```
A - + - - + - - - +
^ S   =
X   ^   P   K - + -
    X   a ^ ^ P
        r X Y a ^ ^
        t     r Y X
        s     t
```

```
A(X|S(X)) =(Parts(X),{Y|Part(Y,X)})
```

*Let **X** be a set. The power set of **X** is the class of all **Y** included in **X**.*

225(T) Parts explanation

```
A - + - - - - - - - + - - +
^ S   A - - - + - - - + -
X   ^ ^ T D
    X Y     @       P
              ^ P   a ^ ^
              Y a ^ r Y X
                r X t
                t
                s
```

```
A(X|S(X)) A(Y) D(@(Y,Parts(X)),Part(Y,X))
```

*Let **X** be a set and **Y** a class. **Y** belongs to the power set of **X** if and only if **Y** is included in **X**.*

228(T) 0 belongs to Parts of every set

```
A - + - - - +
^ S   @
X   ^   0 P
    X     a ^
          r X
          t
          s
```

```
A(X|S(X)) @(0,Parts(X))
```

229(T) A set belongs to Parts of itself

```
A - + - + - +
^ S   @
X   ^   ^ P
    X   X a ^
          r X
          t
          s
```

```
A(X|S(X)) @(X,Parts(X))
```

230(F) GU definition (generalised union)

```
A - - - + - - - - - + - - -
^ T =
X     G   K - - - - - - + -
      U ^ ^ E - + - - - +
        X u ^ @       @
            x   ^ ^   ^ ^
                x X   u x
```

`A(X) =(GU(X),{u|E(x|@(x,X)) @(u,x)})`

*Let **X** be a class. The generalized union of **X** is the class of all **u** for which there exists an **x** belonging to **X** such that **u** belongs to **x**.*

232(F) GI definition (generalized intersection)

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

`A(X) =(GI(X),{u|A(x|@(x,X)) @(u,x)})`

*Let **X** be a class. The generalized intersection of **X** is the class of all **u** such that for any **x** belonging to **X**, **u** belongs to **x**.*

234(A) Axiom of generalized union

```
A - + - - +
^ S   S
X   ^   G
    X   U ^
          X
```

`A(X|S(X)) S(GU(X))`

The generalized union of a set is a set.

235(A) Axiom of power set

```
A - + - - +
^ S   S
X   ^   P
    X   a ^
        r X
        t
        s
```

`A(X|S(X)) S(Parts(X))`

The power set of a set is a set.

236(T) The union of two sets is gen. union of Dbl of these sets

```
A - + - - - - - - + - - - - + -
^ S   A - + - - - - + - - - - +
X     ^ ^ S     =
      X Y     ^     U     G
              Y         ^ ^ U D
                        X Y   b ^ ^
                              l X Y
```

```
A(X|S(X)) A(Y|S(Y)) =(U(X,Y),GU(Dbl(X,Y)))
```

237(T) The union of two sets is a set

```
A - + - - - - - + -
^ S   A - + - - - +
x     ^ ^ S   S
      x y   ^   U
            y     ^ ^
                  x y
```

```
A(x|S(x)) A(y|S(y)) S(U(x,y))
```

241(T) Pair inclusion

```
A - + - - - - - - - - + - - - - - + - - - + - -
^ S   A - + - - - - - - - + - - - - - - - - - - + -
x     ^ ^ S   A - - - - - - - + - - - - + - - - - - +
      x y   ^ ^ T D
            y U     P         X
                    a P     ^   @       @
                    r a ^ ^ U     S   ^   D     ^
                    t i x y       n ^ U   b ^ ^ U
                      r           g x     l x y
```

```
A(x|S(x)) A(y|S(y)) A(U) D(Part(Pair(x,y),U),X(@(Sng(x),U),@(Dbl(x,y),U)))
```

*Let **x**, **y** be sets and **U** a class. The pair of **x**, **y** is included in **U** if and only if Sng of **x** and Dbl of **x**, **y** belong to **U**.*

245(T) Pair belongs to Parts of Parts of U

```
A - + - - - - - - - - - - - - - - + - - - - - - - - - - - + -
^ S   A - + - - - - - - - - - - - - - - + - - - - - - - - - +
X     ^ ^ S   A - + - - - - - - - + - - - - - - - + - - - - - -
      X Y   ^ ^ S   A - + - - - - - - - + - - - - - + - - - - -
            Y x   ^ ^ S   C
                  x y   ^   X             @
                        y     @     @       P     P
                                ^ ^   ^ ^   a ^ ^ a P
                                x X   y Y   i x y r a U
                                            r     t r   ^ ^
                                                  s t   X Y
                                                    s
```

```
A(X|S(X)) A(Y|S(Y)) A(x|S(x)) A(y|S(y))
C(X(@(x,X),@(y,Y)),@(Pair(x,y),Parts(Parts(U(X,Y)))))
```

254(T) Dis is symmetric

```
A - - - - - + - - - +
^ T A - - - - + - + -
X   ^ T D
    Y     D     D
          i ^ ^ i ^ ^
          s X Y s Y X
```

```
A(X) A(Y) D(Dis(X,Y),Dis(Y,X))
```

The inference operators "Cross attachment" and "Cross replacement" are easy to use and facilitate demonstrations. In some cases, however, they cannot be used directly. With the help of the "ZE" form, the desired definitions can be adapted so that they become usable with the mentioned inference operators.

260(R) ZE definition

```
A - - - + -
^ T D
x     Z   T
      E ^
        x
```

```
A(x) D(ZE(x),T)
```

261(T) Every class is ZE

```
A - - +
^ T Z
x   E ^
      x
```

```
A(x) ZE(x)
```

*For any **x**, ZE of **x**.*

267(T) GI corollary

```
A - - - - + - - + -
^ T A - + - - - - +
X   ^ @     P
    x   ^ ^ a G   ^
        x X r I ^ x
            t   X
```

```
A(X) A(x|@(x,X)) Part(GI(X),x)
```

*Let **X** be a class and **x** be an element of this class. Then GI(**X**) is included in **x**.*

282(R) Tra definition (transitive graph)

```
A - + - - + - - - - - - - - - - - - - - - + - - - - + - - - - +
^ G   D
G r ^   T   A - + - - - - - - - - - - + - - - - - - - - - + - -
  a G   r ^ ^ S   A - + - - - - - - - - + - - - + - - - - - - -
  p     a G x   ^ ^ S   A - + - - - - - - - - - - + - - - - + -
  h             x y   ^ ^ S   C
                      y z   ^   X                     @
                            z     @         @           P     ^
                                    P     ^   P     ^   a ^ ^ G
                                    a ^ ^ G   a ^ ^ G   i x z
                                    i x y     i y z     r
                                    r         r
```

```
A(G|Graph(G)) D(Tra(G),A(x|S(x)) A(y|S(y)) A(z|S(z))
C(X(@(Pair(x,y),G),@(Pair(y,z),G)),@(Pair(x,z),G)))
```

*Let **G** be a graph. **G** is transitive if and only if, for any sets **x**, **y**, **z**, if Pair(**x**, **y**) and Pair(**y**, **z**) belong to **G**, then Pair(**x**, **z**) belongs to **G**.*

283(R) Sym definition (symmetric graph)

```
A - + - - + - - - - - - - - - - - + - - - - +
^ G   D
G r ^   S   A - + - - - - - - + - - - - - + -
  a G   y ^ ^ S   A - + - - - - + - - - + - -
  p     m G x   ^ ^ S   C
  h             x y   ^   @         @
                      y     P     ^   P     ^
                            a ^ ^ G   a ^ ^ G
                            i x y     i y x
                            r         r
```

```
A(G|Graph(G)) D(Sym(G),A(x|S(x)) A(y|S(y)) C(@(Pair(x,y),G),@(Pair(y,x),G)))
```

*Let **G** be a graph. **G** is symmetric if and only if, for any sets **x** and **y**, the condition that the pair of **x**, **y** belongs to **G** implies that the pair of **y**, **x** belongs to **G**.*

284(T) Sym corollary

```
A - + - - + - - - - - - - - - - - + - - - - +
^ G   D
G r ^   S   A - + - - - - - - + - - - - - + -
  a G   y ^ ^ S   A - + - - - - + - - - + - -
  p     m G x   ^ ^ S   D
  h             x y   ^   @         @
                      y     P     ^   P     ^
                            a ^ ^ G   a ^ ^ G
                            i x y     i y x
                            r         r
```

```
A(G|Graph(G)) D(Sym(G),A(x|S(x)) A(y|S(y)) D(@(Pair(x,y),G),@(Pair(y,x),G)))
```

*Let **G** be a graph. **G** is symmetric if and only if, for any sets **x** and **y**, the pair of **x**, **y** belongs to **G** if and only if the pair of **y**, **x** belongs to **G**.*

285(R) Ant definition (antisymmetric graph)

```
A - + - - + - - - - - - - - - - - - + - - - - + - - -
^ G   D
G r ^   A   A - + - - - - - - - - + - - - - - + - - + -
  a G   n ^ ^ S   A - + - - - - - - + - - - + - - - - +
  p     t G x   ^ ^ S   C
  h             x y   ^   X                     =
                      y     @         @           ^ ^
                              P     ^   P     ^   x y
                              a ^ ^ G   a ^ ^ G
                              i x y     i y x
                              r         r
```

```
A(G|Graph(G)) D(Ant(G),A(x|S(x)) A(y|S(y))
C(X(@(Pair(x,y),G),@(Pair(y,x),G)),=(x,y)))
```

*Let **G** be a graph. **G** is antisymmetric if and only if, for any sets **x** and **y**, the condition that the pair of **x**, **y** and the pair of **y**, **x** belong to G implies the equality of **x** and **y**.*

288(T) Sym is the equality between a graph and its inverse

```
A - + - - + - + - +
^ G   D
G r ^   S   =
  a G   y ^   ^ I
  p     m G   G n ^
  h             v G
```

```
A(G|Graph(G)) D(Sym(G),=(G,Inv(G)))
```

A graph is symmetric if and only if it is equal to its inverse.

289(T) Tra is composite included

```
A - + - - + - - + + +
^ G   D
G r ^   T   P
  a G   r ^ a o     ^
  p     a G r   ^ ^ G
  h         t   G G
```

```
A(G|Graph(G)) D(Tra(G),Part(o(G,G),G))
```

A graph is transitive if and only if the composite with itself is included in it.

290(F) Dia definition (diagonal of a graph)

```
A - + - - + - - - - + - - - - -
^ G   =
G r ^   D   K - - + - - - + - +
  a G   i ^ ^ X
  p     a G u   @     =
  h               ^ ^   C   C
                  u G   o ^ o ^
                        o u o u
                        r   r
                        1   2
```

```
A(G|Graph(G)) =(Dia(G),{u|X(@(u,G),=(Coor1(u),Coor2(u)))})
```

292(T) Dia is a graph

```
A - + - - +
^ G   G
G r ^ r D
  a G a i ^
  p   p a G
  h   h
```

```
A(G|Graph(G)) Graph(Dia(G))
```

296(T) Ant is inclusion of I(G, Inv(G)) and Dia(G)

```
A - + - - + - - + - + - +
^ G   D
G r ^   A   P
  a G   n ^ a I       D
  p     t G r   ^ I   i ^
  h         t   G n ^ a G
                  v G
```

```
A(G|Graph(G)) D(Ant(G),Part(I(G,Inv(G)),Dia(G)))
```

A graph is antisymmetric if and only if its intersection with its inverse is included in its diagonal.

Graph functions

The main feature of a function is the ability to uniquely link an element from the domain with one element of the codomain. But this feature can very well be defined at the graph level and can be later used for functions. This is what we mean by a graph function, which we name “Func”. The fundamental difference between graph functions and functions is that graph functions are classes, while functions are only relations between three classes: domain, codomain, and graph.

In addition, we will define for graphs several notions usually used for functions but having the same meaning for graphs:

- DomInt (restriction)
- RanInt (restriction)
- DImg (direct image)
- RImg (reverse image)

301(R) Func definition (graph function)

```
A - - - + - - + - - - - - - - - - - - + - - - - - - - - + - - -
^ T D
F     F   X
      u ^   G   A - + - - - - - - + - - - - - - - - + - - - - -
      n F   r ^ ^ S   A - - + - - - + - - - - - - - - - - - + -
      c     a F x   ^ ^ X               A - - + - - - + - - - +
            p       x y   S   @         ^ X               =
            h         1     ^   P     ^ y   S   @           ^ ^
                            y   a ^ ^ F 2     ^   P     ^   y y
                            1   i x y         y   a ^ ^ F   1 2
                                r   1         2   i x y
                                                  r   2
```

A(F) D(Func(F),X(Graph(F),A(x|S(x))
A(y1|X(S(y1),@(Pair(x,y1),F)))
A(y2|X(S(y2),@(Pair(x,y2),F))) =(y1,y2)))

*A class **F** is a graph function if and only if it is a graph, and for any sets **x**, **y1**, **y2**, the belonging of the pair of **x**, **y1** and the belonging of the pair of **x**, **y2** to **F**, **y1** is equal to **y2**.*

303(T) Func is Graph

```
A - + - +
^ F   G
F u ^ r ^
  n F a F
  c   p
      h
```

A(F|Func(F)) Graph(F)

304(T) Existence for Val

```
A - + - - - - + - - - - - - - - - +
^ F   A - + - - - - - - - - - + - -
F u ^ ^ @       E - - - + - - - + -
  n F x   ^ D   ^ T X
  c       x o ^ y     S   @
            m F       ^   P     ^
                        y   a ^ ^ F
                            i x y
                            r
```

A(F|Func(F)) A(x|@(x,Dom(F))) E(y) X(S(y),@(Pair(x,y),F))

*For any graph function **F** and for any **x** belonging to Dom of **F**, there exists **y** such that **y** is a set and the pair of **x**, **y** belongs to **F**.*

305(T) Uniqueness for Val

```
A - + - - - - + - - - - - - - - + - - - - - - - - + - - -
^ F   A - + - - - - - - - - + - - - - - - - - + - - - - -
F u ^ ^ @       A - - + - - - + - - - - - - - - - - - + -
  n F x   ^ D   ^ X               A - - + - - - + - - - +
  c       x o ^ y   S   @         ^ X               =
            m F 1     ^   P     ^ y   S   @           ^ ^
                      y   a ^ ^ F 2     ^   P     ^   y y
                      1   i x y         y   a ^ ^ F   1 2
                          r   1         2   i x y
                                            r   2
```

```
A(F|Func(F))
A(x|@(x,Dom(F)))
A(y1|X(S(y1),@(Pair(x,y1),F)))
A(y2|X(S(y2),@(Pair(x,y2),F))) =(y1,y2)
```

*For any graph function **F**, for any **x** belonging to Dom of **F**, for any set **y1** with the pair of **x**, **y1** belonging to **F**, and for any set **y2** with the pair of **x**, **y2** belonging to **F**, **y1** is equal to **y2**.*

306(F) Val definition

```
A - + - - - - + - - - + - - - - - + - +
^ F   A - + - - - - - - - + - - + - - + -
F u ^ ^ @       X
  n F x   ^ D     S       @
  c       x o ^     V       P         ^
            m F     a ^ ^   a ^ V     F
                    l F x   i x a ^ ^
                            r   l F x
```

```
A(F|Func(F)) A(x|@(x,Dom(F))) X(S(Val(F,x)),@(Pair(x,Val(F,x)),F))
```

*For any graph function **F**, and for any **x** belonging to Dom of **F**, Val(**F**, **x**) is a set, and the pair of **x** and Val(**F**, **x**) belongs to **F**.*

Usually (and abusively) the notation F(x) is used for Val(F, x). The "Logic" system does not allow this.

308(T) Val belongs to Ran

```
A - + - - - - + - - + - - +
^ F   A - + - - - - - - + - -
F u ^ ^ @       @
  n F x   ^ D     V     R
  c       x o ^   a ^ ^ a ^
            m F   l F x n F
```

```
A(F|Func(F)) A(x|@(x,Dom(F))) @(Val(F,x),Ran(F))
```

309(T) Val Inv corollary

```
A - + - - - - + - - - + - - - - + - + -
^ F   A - + - - - - - - - + - - - - - - +
F u ^ ^ @       C
  n F x   ^ D     F     =
  c       x o ^   u I     ^ V
            m F   n n ^   x a I   V
                  c v F     l n ^ a ^ ^
                              v F l F x
```

```
A(F|Func(F)) A(x|@(x,Dom(F))) C(Func(Inv(F)),=(x,Val(Inv(F),Val(F,x))))
```

*Let **F** be a graph function and **x** an element of Dom of **F**. If the inverse of **F** is a graph function, then **x** is equal to Val(Inv(**F**), Val(**F**, **x**)).*

312(T) Pair and Val for Func corollary 1

```
A - + - - - - - - - - - - - + - - - - + - - + - -
^ F   A - + - - - - - - - + - - - - + - - - - - - + -
F u ^ ^ S   A - + - - - - + - - - - - - - - - - - +
  n F x   ^ ^ S   D
  c       x y   ^   @         X
                y     P     ^   @       =
                      a ^ ^ F     ^ D     V     ^
                      i x y       x o ^   a ^ ^ y
                      r             m F   l F x
```

```
A(F|Func(F)) A(x|S(x)) A(y|S(y)) D(@(Pair(x,y),F),X(@(x,Dom(F)),=(Val(F,x),y)))
```

*Let **F** be a graph function and **x**, **y** be sets. The pair of **x**, **y** belongs to **F** if and only if **x** belongs to Dom of F and Val(F, **x**) = **y**.*

318(T) Val of composite

```
A - + - - - - - - - - - + - - - - + - - - - - - + -
^ F   A - + - - - - - - - + - - - - + - - + - - -
F u ^ ^ F   A - + - - - - - - - - - - + - - - - - +
  n F G u ^ ^ @           =
  c     n G x   ^ D         V         V
        c       x o o       a o     ^ a ^ V
                  m   ^ ^   l   ^ ^ x l G a ^ ^
                      F G       F G       l F x
```

```
A(F|Func(F)) A(G|Func(G)) A(x|@(x,Dom(o(F,G)))) =(Val(o(F,G),x),Val(G,Val(F,x)))
```

*Let **F**, **G** be graph functions and **x** be an element of Dom of o(**F**,**G**). Then Val(o(**F**, **G**), **x**) = Val(**G**, Val(**F**, **x**)).*

Because of the theorem above, the notation with inverted arguments for composite functions is usually used.

319(T) Pair and Val for Func corollary 2

```
A - + - - - - + - - - - - - - - + - - + - -
^ F   A - + - - - - - - - - + - - - - - + -
F u ^ ^ @       A - + - - - - + - - - - - +
  n F x   ^ D   ^ S   D
  c       x o ^ y   ^   @         =
            m F     y     P     ^   V     ^
                          a ^ ^ F   a ^ ^ y
                          i x y     l F x
                          r
```

`A(F|Func(F)) A(x|@(x,Dom(F))) A(y|S(y)) D(@(Pair(x,y),F),=(Val(F,x),y))`

*Let **F** be a graph function, **x** an element of the Dom of **F**, and **y** a set. The pair of **x**, **y** belongs to **F** if and only if Val(**F**, **x**) = **y**.*

If F is a graph function, instead of Pair(x, y) ∈ F, we have now y = Val(F, x), and we say that y is the value of F for x. This corresponds to the usual notation y = f(x).

320(T) 0 is Graph

```
G
r 0
a
p
h
```

`Graph(0)`

321(T) 0 is Func

```
F
u 0
n
c
```

`Func(0)`

322(T) Func existence

```
E - - +
^ T F
F   u ^
    n F
    c
```

`E(F) Func(F)`

323(T) Func equality

```
A - + - - - - - + - - - - - - - - - - - - - + - - - - -
^ F   A - + - - - + - - - - - - - - - - - - - - - - - - +
F u ^ ^ F   D
  n F G u ^   =     A - + - - - - - - - + - - - - + - -
  c     n G     ^ ^ ^ S   A - + - - - - + - - - - - + -
        c       F G x   ^ ^ S   D
                        x y   ^   @         @
                              y     P     ^   P     ^
                                    a ^ ^ F   a ^ ^ G
                                    i x y     i x y
                                    r         r
```

```
A(F|Func(F)) A(G|Func(G)) D(=(F,G),A(x|S(x)) A(y|S(y))
D(@(Pair(x,y),F),@(Pair(x,y),G)))
```

*Let **F**, **G** be graph functions. **F** is equal to **G** if and only if, for any sets **x** and **y**, the belonging of Pair(**x**, **y**) to **F** is equivalent to the belonging of Pair(**x**, **y**) to **G**.*

333(F) Ident definition (identic graph function)

```
A - - - + - - - - + - - - - -
^ T =
X     I   K - - - - - + - - -
      d ^ ^ E - + - - - - + +
      e X u ^ @     =
      n     x   ^ ^   ^ P
      t         x X   u a ^ ^
                        i x x
                        r
```

```
A(X) =(Ident(X),{u|E(x|@(x,X)) =(u,Pair(x,x))})
```

335(T) Ident is Graph

```
A - - - +
^ T G
X   r I
    a d ^
    p e X
    h n
      t
```

```
A(X) Graph(Ident(X))
```

340(T) Dom of Ident

```
A - - - - + +
^ T =
X     D     ^
      o I   X
      m d ^
        e X
        n
        t
```

```
A(X) =(Dom(Ident(X)),X)
```

*For any **X**, Dom(Ident(**X**)) = **X**.*

342(T) Ident is Func

```
A - - - +
^ T F
X   u I
    n d ^
    c e X
      n
      t
```

343(T) Val of Ident

```
A - - - - + - - - + - -
^ T A - + - - - - - + +
X   ^ @     =
    x   ^ ^   V       ^
        x X   a I   ^ x
              l d ^ x
                e X
                n
                t
```

A(X) A(x|@(x,X)) =(Val(Ident(X),x),x)

*Let **X** be a class. For any **x** belonging to **X**, Val(Ident(**X**), **x**) = **x**.*

344(T) Ran of Ident

```
A - - - - + +
^ T =
X     R     ^
      a I   X
      n d ^
        e X
        n
        t
```

A(X) =(Ran(Ident(X)),X)

Simple graph functions

Functions can be injective, surjective, or bijective. For graph functions, surjectivity has no meaning. By simple graph functions, we will mean injective/bijective graph functions.

345(R) SFunc definition (simple graph function)

```
A - - - + - - + - - - - - - - - - - - + - - - - - - - - + - - -
^ T D
F     S   X
      F ^   F   A - + - - - - - - - + - - - - - - - - + - - - -
      u F   u ^ ^ S   A - - + - - + - - - - - - - - - - - - + -
      n     n F y   ^ ^ X               A - - + - - + - - - - +
      c     c       y x   S   @         ^ X               =
                      1     ^   P     ^ x   S   @           ^ ^
                            x   a ^ ^ F 2     ^   P     ^   x x
                            1   i x y         x   a ^ ^ F   1 2
                                r 1           2   i x y
                                                  r 2
```

```
A(F) D(SFunc(F),X(Func(F),A(y|S(y)) A(x1|X(S(x1),@(Pair(x1,y),F)))
A(x2|X(S(x2),@(Pair(x2,y),F))) =(x1,x2)))
```

348(T) SFunc corollary

```
A - - - + - - + - - - - + - - - - + - - - + - - + - - - -
^ T D
F     S   X
      F ^   F   A - + - - - - - - - - - - - + - - - - + -
      u F   u ^ ^ @       A - + - - - - - - - - - + - - +
      n     n F x   ^ D   ^ @       C
      c     c   1   x o ^ x   ^ D     =             =
                    1 m F 2   x o ^     V     V       ^ ^
                              2 m F     a ^ ^ a ^ ^   x x
                                        l F x l F x   1 2
                                            1     2
```

```
A(F) D(SFunc(F),X(Func(F),A(x1|@(x1,Dom(F))) A(x2|@(x2,Dom(F)))
C(=(Val(F,x1),Val(F,x2)),=(x1,x2))))
```

*Let **F** be a class. **F** is a simple graph function if and only if **F** is a graph function, and for any $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ belonging to the Dom of **F**, Val(**F**, $\boldsymbol{x_1}$) = Val(**F**, $\boldsymbol{x_2}$) implies $\boldsymbol{x_1} = \boldsymbol{x_2}$.*

350(T) SFunc and Inv

```
A - + - - + - - - +
^ F   D
F u ^   S   F
  n F   F ^ u I
  c     u F n n ^
        n   c v F
        c
```

```
A(F|Func(F)) D(SFunc(F),Func(Inv(F)))
```

*Let **F** be a graph function. **F** is a simple graph function if and only if its inverse is a graph function.*

352(T) Inv of SFunc is SFunc

```
A - + - - +
^ S   S
F F ^ F I
  u F u n ^
  n   n v F
  c   c
```

```
A(F|SFunc(F)) SFunc(Inv(F))
```

The inverse of a simple graph function is a simple graph function.

355(T) The composite of simple graph functions is a simple graph function

```
A - + - - - - - + -
^ S   A - + - - - +
F F ^ ^ S   S
  u F G F ^ F o
  n     u G u   ^ ^
  c     n   n   F G
        c   c
```

```
A(F|SFunc(F)) A(G|SFunc(G)) SFunc(o(F,G))
```

357(T) Ident is SFunc

```
A - - - +
^ T S
X   F I
    u d ^
    n e X
    c n
      t
```

```
A(X) SFunc(Ident(X))
```

358(F) Const definition (constant graph function)

```
A - - - - - - + - - - - - + - - - - -
^ T A - + - - - + - - - - - - - - - +
X   ^ S   =
    a   ^   C     K - - - - - + - - -
        a   o ^ ^ ^ E - + - - - - + -
            n X a u ^ @     =
            s       x   ^ ^   ^ P
            t           x X   u a ^ ^
                                i x a
                                r
```

```
A(X) A(a|S(a)) =(Const(X,a),{u|E(x|@(x,X)) =(u,Pair(x,a))})
```

361(T) Const is Graph

```
A - - - - - - + -
^ T A - + - - - +
X   ^ S   G
    a   ^ r C
        a a o ^ ^
          p n X a
          h s
            t
```

A(X) A(a|S(a)) Graph(Const(X,a))

365(T) Dom of Const

```
A - - - - - - - + - +
^ T A - + - - - - + -
X   ^ S   =
    a   ^   D       ^
        a   o C     X
            m o ^ ^
              n X a
              s
              t
```

A(X) A(a|S(a)) =(Dom(Const(X,a)),X)

367(T) Const is Func

```
A - - - - - - + -
^ T A - + - - - +
X   ^ S   F
    a   ^ u C
        a n o ^ ^
          c n X a
            s
            t
```

A(X) A(a|S(a)) Func(Const(X,a))

368(T) Val of Const

```
A - - - - - - - + - - - + - - -
^ T A - + - - - - - - - - - + - +
X   ^ S   A - + - - - - - - - + -
    a   ^ ^ @     =
        a x   ^ ^   V         ^
              x X   a C     ^ a
                    l o ^ ^ x
                      n X a
                      s
                      t
```

A(X) A(a|S(a)) A(x|@(x,X)) =(Val(Const(X,a),x),a)

374(T) Cart explanation

```
A - - - - - - - - - + - - - - - - - - - - - + - - - -
^ T A - - - - - - - - + - - - - - - - - - - - - - - +
X   ^ T A - - - + - - - - - + - - - - - - + - - - + -
    Y   ^ T D
        p     @         X
                ^ C       @         X
                p a ^ ^     ^ C       @       @
                  r X Y     p a U U     C   ^   C   ^
                  t           r n n     o ^ X   o ^ Y
                              t i i     o p     o p
                                v v     r       r
                                        1       2
```

```
A(X) A(Y) A(p)
D(@(p,Cart(X,Y)),X(@(p,Cart(Univ,Univ)),X(@(Coor1(p),X),@(Coor2(p),Y))))
```

*Let **X**, **Y**, **p** be classes. **p** belongs to Cart of **X**, **Y** if and only if **p** belongs to Cart of Univ, Univ, Coor1(**p**) belongs to **X**, and Coor2(**p**) belongs to **Y**.*

401(F) DomInt definition

```
A - + - - - - + - - + - - -
^ G   A - - - - + - - - + -
G r ^ ^ T =
  a G X     D     I
  p         o ^ ^   ^ C
  h         m G X   G a ^ U
            I         r X n
            n         t   i
            t             v
```

```
A(G|Graph(G)) A(X) =(DomInt(G,X),I(G,Cart(X,Univ)))
```

*DomInt(**G**, **X**) is the intersection of **G** and Cart(**X**, Univ).*

402(F) RanInt definition

```
A - + - - - - + - - + - - -
^ G   A - - - - + - - - - +
G r ^ ^ T =
  a G Y     R     I
  p         a ^ ^   ^ C
  h         n G Y   G a U ^
            I         r n Y
            n         t i
            t           v
```

```
A(G|Graph(G)) A(Y) =(RanInt(G,Y),I(G,Cart(Univ,Y)))
```

*RanInt(**G**, **Y**) is the intersection of **G** and Cart(Univ, **Y**).*

405(T) DomInt is Graph

```
A - + - - - - + -
^ G   A - - - - +
G r ^ ^ T G
  a G X   r D
  p       a o ^ ^
  h       p m G X
          h I
            n
            t
```

```
A(G|Graph(G)) A(X) Graph(DomInt(G,X))
```

406(T) RanInt is Graph

```
A - + - - - - + -
^ G   A - - - - +
G r ^ ^ T G
  a G Y   r R
  p       a a ^ ^
  h       p n G Y
          h I
            n
            t
```

```
A(G|Graph(G)) A(Y) Graph(RanInt(G,Y))
```

411(T) DomInt of a Func is Func

```
A - + - - - - + -
^ F   A - - - - +
F u ^ ^ T F
  n F X   u D
  c       n o ^ ^
          c m F X
            I
            n
            t
```

```
A(F|Func(F)) A(X) Func(DomInt(F,X))
```

413(T) Ran of DomInt is included in Ran

```
A - + - - - - - + - - +
^ G   A - - - - - + - -
G r ^ ^ T P
  a G X   a R       R
  p       r a D     a ^
  h       t n o ^ ^ n G
              m G X
              I
              n
              t
```

```
A(G|Graph(G)) A(X) Part(Ran(DomInt(G,X)),Ran(G))
```

417(T) Func explanation using Val

```
A - + - + - - - - - - + - - - - - - + -
^ F   =
F u ^   ^ K - - - - - - - + - - - - - -
  n F   F ^ E - + - - - - - - + - - +
  c       u ^ @       =
            x   ^ D     ^ P
                x o ^   u a ^ V
                  m F     i x a ^ ^
                          r   l F x
```

```
A(F|Func(F)) =(F,{u|E(x|@(x,Dom(F))) =(u,Pair(x,Val(F,x)))})
```

*If **F** is a graph function, then **F** is equal to the class of all **u** for which there exists an **x** belonging to Dom of **F**, such that **u** is equal to the pair of **x** and Val(**F**, **x**).*

418(T) Val of DomInt

```
A - + - - - - - - - + - - - - + - - - + -
^ F   A - - - - - - - + - - - - + - - - -
F u ^ ^ T A - + - - - - - - - - - - + - - +
  n F X   ^ @           =
  c       x   ^ I         V         V
              x   D   ^   a D     ^ a ^ ^
                  o ^ X   l o ^ ^ x l F x
                  m F       m F X
                            I
                            n
                            t
```

```
A(F|Func(F)) A(X) A(x|@(x,I(Dom(F),X))) =(Val(DomInt(F,X),x),Val(F,x))
```

424(T) Dis of Dom implies Dis of Graph

```
A - + - - - - - - - + - - - + -
^ G   A - + - - - - - + - - +
F r ^ ^ G   C
  a F G r ^   D         D
  p     a G   i D   D   i ^ ^
  h     p     s o ^ o ^ s F G
        h       m F m G
```

```
A(F|Graph(F)) A(G|Graph(G)) C(Dis(Dom(F),Dom(G)),Dis(F,G))
```

*Let **F** and **G** be graphs. If Dom of **F** and Dom of **G** are disjoint, then **F** and **G** are also disjoint.*

430(T) Union of graph functions with disjoint domains is a graph function

```
A - + - - - - - - - + - - - - - + -
^ F   A - + - - - - - - + - - - - +
F u ^ ^ F   C
  n F G u ^   D         F
  c     n G   i D   D   u U
        c     s o ^ o ^ n   ^ ^
                m F m G c   F G
```

A(F|Func(F)) A(G|Func(G)) C(Dis(Dom(F),Dom(G)),Func(U(F,G)))

*Let **F** and **G** be two graph functions. If Dom of **F** and Dom of **G** are disjoint, then the union of **F** and **G** is a graph function.*

438(T) Val equality is double implication

```
A - + - - - - - + - - - - - + - - - + - - + - - - - - - - - - - + - - - - - +
^ F   A - + - - - - - - - - - - - - + - - - - - - - - - - + - - - - - - -
F u ^ ^ @       A - + - - - - - - - - - - + - - - - - - - - - - - - - + - -
  n F x   ^ D   ^ @       D
  c   1   x o ^ x   ^ D     =             A - + - - - - - + - - - - - + -
          1 m F 2   x o ^     V     V     ^ S   D
                    2 m F     a ^ ^ a ^ ^ y   ^   @         @
                              l F x l F x     y     P     ^   P     ^
                                  1     2           a ^ ^ F   a ^ ^ F
                                                    i x y     i x y
                                                    r 1       r 2
```

A(F|Func(F)) A(x1|@(x1,Dom(F))) A(x2|@(x2,Dom(F)))
D(=(Val(F,x1),Val(F,x2)),A(y|S(y)) D(@(Pair(x1,y),F),@(Pair(x2,y),F)))

*Let **F** be a graph function and **x1**, **x2** be elements of Dom of **F**. The value of **F** for **x1** is equal to the value of **F** for **x2** if and only if, for any set **y**, the belonging of Pair(**x1**, **y**) to **F** is equivalent to the belonging of Pair(**x2**, **y**) to **F**.*

439(T) Val of Inv

```
A - - + - - + - - - - + - - - + - + - -
^ X         A - + - - - - - - - - + +
F   F   F     ^ @       =
    u ^ u I   x   ^ D     V           ^
    n F n n ^     x o ^   a I   V     x
    c   c v F       m F   l n ^ a ^ ^
                            v F l F x
```

A(F|X(Func(F),Func(Inv(F)))) A(x|@(x,Dom(F))) =(Val(Inv(F),Val(F,x)),x)

*Let **F** be a class such that **F** and Inv of **F** are graph functions. For any element **x** of Dom of **F**, we have the equality Val(Inv(**F**), Val(**F**, **x**)) = **x**.*

440(T) Val of Inv

```
A - - + - - + - - - - + - - + - - + - -
^ X         A - + - - - - - - - - + +
F   F   F     ^ @       =
    u ^ u I   y   ^ R     V           ^
    n F n n ^     y a ^   a ^ V       y
    c   c v F       n F   l F a I   ^
                              l n ^ y
                                v F
```

A(F|X(Func(F),Func(Inv(F)))) A(y|@(y,Ran(F))) =(Val(F,Val(Inv(F),y)),y)

*Let **F** be a class such that **F** and Inv of **F** are graph functions. For any element **y** of Ran of **F**, we have the equality Val(**F**, Val(**F**, **y**)) = **y**.*

445(T) SFunc corollary

```
A - - + - - + - - +
^ X           S
F   F   F     F I
    u ^ u I   u n ^
    n F n n ^ n v F
    c   c v F c
```

`A(F|X(Func(F),Func(Inv(F)))) SFunc(Inv(F))`

*If **F** and Inv of **F** are graph functions, then Inv of **F** is a simple graph function.*

451(F) DImg definition (direct image of X under G)

```
A - + - - - - + - - - - - - - - - - - +
^ G   A - - - - + - - - - + - - - - -
G r ^ ^ T =
  a G X     D     K - - - - - - - - + -
  p         I ^ ^ ^ E - + - - - - + - -
  h         m G X y ^ @     @
            g       x   ^ ^   P     ^
                        x X   a ^ ^ G
                              i x y
                              r
```

`A(G|Graph(G)) A(X) =(DImg(G,X),{y|E(x|@(x,X)) @(Pair(x,y),G)})`

452(F) RImg definition (reverse image of Y under G)

```
A - + - - - - + - - - - - - - - - - - +
^ G   A - - - - + - - - - + - - - - -
G r ^ ^ T =
  a G Y     R     K - - - - - - - + - -
  p         I ^ ^ ^ E - + - - - - - + -
  h         m G Y x ^ @     @
            g       y   ^ ^   P     ^
                        y Y   a ^ ^ G
                              i x y
                              r
```

`A(G|Graph(G)) A(Y) =(RImg(G,Y),{x|E(y|@(y,Y)) @(Pair(x,y),G)})`

459(T) Part of DImg and Ran

```
A - + - - - - + - - +
^ G   A - - - - + - -
G r ^ ^ T P
  a G X   a D     R
  p       r I ^ ^ a ^
  h       t m G X n G
            g
```

`A(G|Graph(G)) A(X) Part(DImg(G,X),Ran(G))`

*Let **G** be a graph and **X** a class. DImg of **G** and **X** is included in Ran of **G**.*

460(T) Part of RImg and Dom

```
 A - + - - - - + - - +
 ^ G   A - - - - + - -
 G r ^ ^ T P
   a G Y   a R     D
   p       r I ^ ^ o ^
   h       t m G Y m G
               g
```

`A(G|Graph(G)) A(Y) Part(RImg(G,Y),Dom(G))`

*Let **G** be a graph and **Y** a class. RImg of **G** and **Y** is included in Dom of **G**.*

463(A) Axiom of replacement

```
 A - + - - - + - - +
 ^ F   C
 F u ^   S     S
   n F     D     R
   c       o ^   a ^
           m F   n F
```

`A(F|Func(F)) C(S(Dom(F)),S(Ran(F)))`

*Let **F** be a graph function. If Dom of **F** is a set, then Ran of **F** is a set.*

466(T) If Dom of a graph function F is a set, then F is a set

```
 A - + - - - + - +
 ^ F   C
 F u ^   S     S
   n F     D     ^
   c       o ^   F
           m F
```

`A(F|Func(F)) C(S(Dom(F)),S(F))`

467(R) OneToOne (one-to-one) definition

```
 A - - - - - + - - - - - - - - + - - - -
 ^ T A - - - - + - - - - - - - - - - - +
 X   ^ T D
     Y     O     E - + - - - + - - - + -
           n ^ ^ ^ S   X
           e X Y F F ^   =       =
           T       u F     D   ^   R   ^
           o       n       o ^ X   a ^ Y
           O       c       m F     n F
           n
           e
```

`A(X) A(Y) D(OneToOne(X,Y),E(F|SFunc(F)) X(=(Dom(F),X),=(Ran(F),Y)))`

*Let **X**, **Y** be classes. We say that **X** and **Y** are OneToOne if and only if there exists a simple graph function **F** such that Dom of **F** is equal to **X** and Ran of **F** is equal to **Y**.*

468(T) Replacement corollary for OneToOne

```
A - - - - - - + - + - - -
^ T A - - - - - - - + - +
X   ^ T C
    Y     X           S
            S   O       ^
              ^ n ^ ^   Y
              X e X Y
                T
                o
                O
                n
                e
```

`A(X) A(Y) C(X(S(X),OneToOne(X,Y)),S(Y))`

*Let **X**, **Y** be classes. If **X** is a set and **X**, **Y** are OneToOne, then **Y** is a set.*

473(T) RImg of Func, Sng is included in Dom

```
A - + - - - - + - - + - - - +
^ F   A - + - - - - - - + - -
F u ^ ^ @       P
  n F y   ^ R   a R       D
  c       y a ^ r I ^ S   o ^
            n F t m F n ^ m F
                  g   g y
```

`A(F|Func(F)) A(y|@(y,Ran(F))) Part(RImg(F,Sng(y)),Dom(F))`

*Let **F** be a graph function and **y** in Ran(**F**). Then RImg(**F**, Sng(**y**)) is included in Dom(**F**).*

Axiom of global choice

In the following, we will present an extremely important principle, often implicitly used in mathematical demonstrations, although it does not result from the previous postulates of mathematics or logic. This principle was called by Zermelo the axiom of choice.

477(R) Choice definition (choice function)

```
A - - - + - - - + - - + - - - - - - - - - - - - - + - -
^ T D
C     C   X
h     h ^   X                   A - - + - - + - - - + +
      o C     F   =             ^ X         @
      i h     u ^   D   K - - + x   S   N       V     ^
      c       n C   o ^ ^ N           ^   E     a ^ ^ x
      e       c h   m C A   E         x   m ^   l C x
                      h     m ^         p x     h
                            p A         t
                            t           y
                            y
```

```
A(Ch)
D(Choice(Ch),X(X(Func(Ch),=(Dom(Ch),{A|N(Empty(A))})),A(x|X(S(x),N(Empty(x))))
@(Val(Ch,x),x)))
```

*Let **Ch** be a class. **Ch** is a choice function if and only if **Ch** is a graph function having the domain equal to the class of not empty sets, and for any not empty set **x**, Val(**Ch**, **x**) belongs to **x**.*

478(A) Axiom of global choice

```
E - - +
^ T C
C   h ^
h   o C
    i h
    c
    e
```

```
E(Ch) Choice(Ch)
```

There exists a choice function.

479(F) ParInv definition (partial inverse of a Func defined by a choice function)

```
A - - + - - + - - - - - + - - - - - - + - - - - - - - + - -
^ X           A - + - - - + - - - - - - - - - - - + - - - -
F   F   S     ^ C   =
    u ^   D   C h ^   P     K - - - - - - + - - - - - - - -
    n F   o ^ h o C   a ^ ^ ^ E - + - - - - - + - - - - - +
    c     m F   i h   r F C u ^ @       =
                c     I   h   y   ^ R     ^ P
                e     n           y a ^   u a ^ V
                      v             n F     i y a ^ R
                                            r   l C I ^ S
                                                  h m F n ^
                                                    g   g y
```

```
A(F|X(Func(F),S(Dom(F)))) A(Ch|Choice(Ch)) =(ParInv(F,Ch),{u|E(y|@(y,Ran(F)))
=(u,Pair(y,Val(Ch,RImg(F,Sng(y))))))})
```

*Let **F** be a graph function with its domain a set, and **Ch** a choice function. Then ParInv(**F**, **Ch**) = {**u** | there exists **y** in Ran(**F**) such that **u** = (**y**, Val(**Ch**, RImg(**F**, {**y**}))}.*

480(T) Val of Ch is a set

```
A - + - - - - - - - - - + -
^ C   A - - + - - + - - - +
C h ^ ^ X           S
h o C x   S   N       V
  i h       ^   E     a ^ ^
  c         x   m ^   l C x
  e             p x     h
                t
                y
```

```
A(Ch|Choice(Ch)) A(x|X(S(x),N(Empty(x)))) S(Val(Ch,x))
```

481(T) Val of Ch and RImg of Sng is a set

```
A - - + - - + - - - - - - - - + - - - - + - -
^ X           A - + - - - - - - - + - - - -
F   F   S     ^ C   A - + - - - - - - - - - +
    u ^   D   C h ^ ^ @         S
    n F   o ^ h o C y   ^ R       V
    c     m F   i h     y a ^     a ^ R
                c         n F     l C I ^ S
                e                   h m F n ^
                                      g   g y
```

```
A(F|X(Func(F),S(Dom(F)))) A(Ch|Choice(Ch)) A(y|@(y,Ran(F)))
S(Val(Ch,RImg(F,Sng(y))))
```

485(T) ParInv is Graph

```
A - - + - - + - - - - - - + -
^ X           A - + - - - - +
F   F   S     ^ C   G
    u ^   D   C h ^ r P
    n F   o ^ h o C a a ^ ^
    c     m F   i h p r F C
                c   h I   h
                e     n
                      v
```

```
A(F|X(Func(F),S(Dom(F)))) A(Ch|Choice(Ch)) Graph(ParInv(F,Ch))
```

492(T) Dom of ParInv is Ran

```
A - - + - - + - - - - - - - + - - - +
^ X           A - + - - - - - + - -
F   F   S     ^ C   =
    u ^   D   C h ^   D       R
    n F   o ^ h o C   o P     a ^
    c     m F   i h   m a ^ ^ n F
                c       r F C
                e       I   h
                        n
                        v
```

```
A(F|X(Func(F),S(Dom(F)))) A(Ch|Choice(Ch)) =(Dom(ParInv(F,Ch)),Ran(F))
```

*Let **F** be a graph function with Dom(**F**) a set, and let **Ch** be a choice function. Then Dom(ParInv(**F**, **Ch**)) = Ran(**F**).*

494(T) ParInv is Func

```
A - - + - - + - - - - - + -
^ X           A - + - - - +
F   F   S     ^ C   F
    u ^   D   C h ^ u P
    n F   o ^ h o C n a ^ ^
    c     m F   i h c r F C
                c     I   h
                e     n
                      v
```

```
A(F|X(Func(F),S(Dom(F)))) A(Ch|Choice(Ch)) Func(ParInv(F,Ch))
```

495(T) ParInv Val

```
A - - + - - + - - - - - - - + - - - + - - - - - + - -
^ X           A - + - - - - - - - - - + - - + - - - -
F   F   S     ^ C   A - + - - - - - - - + - - - - - +
    u ^   D   C h ^ ^ @       =
    n F   o ^ h o C y   ^ R     V         V
    c     m F   i h     y a ^   a P     ^ a ^ R
                c         n F   l a ^ ^ y l C I ^ S
                e                 r F C     h m F n ^
                                  I   h       g   g y
                                  n
                                  v
```

```
A(F|X(Func(F),S(Dom(F)))) A(Ch|Choice(Ch)) A(y|@(y,Ran(F)))
=(Val(ParInv(F,Ch),y),Val(Ch,RImg(F,Sng(y))))
```

*Let **F** be a graph function whose domain is a set, **Ch** a choice function, and **y** an element of Ran of **F**. Then Val(ParInv(**F**,**Ch**),**y**) is equal to Val(**Ch**,RImg(**F**,Sng(**y**)).*

501(T) Ran of ParInv is included in Dom

```
A - - + - - + - - - - - - + - - +
^ X           A - + - - - - + - -
F   F   S     ^ C   P
    u ^   D   C h ^ a R       D
    n F   o ^ h o C r a P     o ^
    c     m F   i h t n a ^ ^ m F
                c       r F C
                e       I   h
                        n
                        v
```

```
A(F|X(Func(F),S(Dom(F)))) A(Ch|Choice(Ch)) Part(Ran(ParInv(F,Ch)),Dom(F))
```

*If **F** is a graph function whose domain is a set and **Ch** is a choice function, then Ran(ParInv(**F**,**Ch**)) is included in Dom(**F**).*

505(T) Dom of composite

```
A - + - - - - - - - + - - + - -
^ G   A - + - - - - - + - - - - +
G r ^ ^ G   =
  a G H r ^   D       R
  p     a H   o o     I ^ D
  h     p     m   ^ ^ m G o ^
        h         G H g   m H
```

`A(G|Graph(G)) A(H|Graph(H)) =(Dom(o(G,H)),RImg(G,Dom(H)))`

*If **F** and **H** are graph functions, then Dom(o(**G**,**H**)) is equal to RImg(**G**,Dom(**H**)).*

506(T) Ran of composite

```
A - + - - - - - - - + - - - - - +
^ G   A - + - - - - - + - + - -
G r ^ ^ G   =
  a G H r ^   R       D
  p     a H   a o     I ^ R
  h     p     n   ^ ^ m H a ^
        h         G H g   n G
```

`A(G|Graph(G)) A(H|Graph(H)) =(Ran(o(G,H)),DImg(H,Ran(G)))`

*If **F** and **H** are graph functions, then Ran(o(**G**,**H**)) is equal to DImg(**G**,Ran(**H**)).*

508(T) Dom of composite of ParInv

```
A - - + - - + - - - - - - - - + - + - +
^ X           A - + - - - - - + - - -
F   F   S     ^ C   =
    u ^   D   C h ^   D         R
    n F   o ^ h o C   o o       a ^
    c     m F   i h   m   P   ^ n F
                c         a ^ ^ F
                e         r F C
                          I   h
                          n
                          v
```

`A(F|X(Func(F),S(Dom(F)))) A(Ch|Choice(Ch)) =(Dom(o(ParInv(F,Ch),F)),Ran(F))`

*If **F** is a graph function whose domain is a set and **Ch** is a choice function, then Dom(o(ParInv(**F**,**Ch**),**F**)) is equal to Ran(**F**).*

510(T) Composite of ParInv and F is Ident of Ran of F

```
A - - + - - + - - - - - - - + - + - - +
^ X         A - + - - - - + - - - -
F   F   S   ^ C   =
    u ^   D   C h ^   o         I
    n F   o ^ h o C     P     ^ d R
    c     m F   i h     a ^ ^ F e a ^
                c       r F C   n n F
                e       I   h   t
                        n
                        v
```

A(F|X(Func(F),S(Dom(F)))) A(Ch|Choice(Ch)) =(o(ParInv(F,Ch),F),Ident(Ran(F)))

*If **F** is a graph function whose domain is a set and **Ch** is a choice function, then o(ParInv(**F**,**Ch**),**F**) is equal to Ident(Ran(**F**)).*

511(T) Partial inversion existence

```
A - - + - - + - - - - - - - - + - - + - - - - - - + - - - - - +
^ X         E - + - - - - + - - - - - - - - + - - - - - + - -
F   F   S   ^ F   X
    u ^   D   G u ^   =             X
    n F   o ^   n G     o     I       =         P
    c     m F   c         ^ ^ d R       D   R   a R   D
                          G F e a ^     o ^ a ^ r a ^ o ^
                              n n F     m G n F t n G m F
                              t
```

A(F|X(Func(F),S(Dom(F)))) E(G|Func(G))
X(=(o(G,F),Ident(Ran(F))),X(=(Dom(G),Ran(F)),Part(Ran(G),Dom(F))))

*For any graph function **F** with Dom(**F**) being a set, there exists a graph function **G** such that o(**G**, **F**) = Ident(Ran(**F**)), Dom(**G**) = Ran(**F**), and Ran(**G**) is included in Dom(**F**).*

Various properties of graph functions

514(T) Func definition corollary 1

```
A - + - - + - - - - + - - - - - - - - - + - - - - - - - - - + - - -
^ G   D
F r ^   F   A - + - - - - - - - - - + - - - - - - - - - + - - - - -
  a F   u ^ ^ @       A - - + - - - + - - - - - - - - - - - - + -
  p     n F x     ^ D   ^ X               A - - + - - - - + - - - - +
  h     c         x o ^ y   S   @         ^ X                 =
                    m F 1     ^   P     ^ y   S   @             ^ ^
                              y   a ^ ^ F 2     ^   P     ^   y y
                              1   i x y         y   a ^ ^ F   1 2
                                  r   1         2   i x y
                                                    r   2
```

A(F|Graph(F)) D(Func(F),A(x|@(x,Dom(F))) A(y1|X(S(y1),@(Pair(x,y1),F)))
A(y2|X(S(y2),@(Pair(x,y2),F))) =(y1,y2))

518(T) Func definition corollary 3

```
A - + - - + - - - - + - - - - - - - - + - - - - - - - - + - - -
^ G   D
F r ^   F   A - + - - - - - - - - + - - - - - - - - + - - - - -
  a F   u ^ ^ @       E - - + - - - + - - - - - - - - - - - + -
  p     n F x   ^ D   ^ X               A - - + - - - + - - - +
  h     c       x o ^ y   S   @         ^ X               =
                  m F       ^   P     ^ z   S   @           ^ ^
                            y   a ^ ^ F       ^   P     ^   y z
                                i x y         z   a ^ ^ F
                                r                 i x z
                                                  r
```

A(F|Graph(F)) D(Func(F),A(x|@(x,Dom(F))) E(y|X(S(y),@(Pair(x,y),F)))
A(z|X(S(z),@(Pair(x,z),F))) =(y,z))

519(T) SFunc is Func and Func of Inv

```
A - - - + - - + - - - +
^ T D
F     S   X
      F ^   F   F
      u F   u ^ u I
      n     n F n n ^
      c     c   c v F
```

A(F) D(SFunc(F),X(Func(F),Func(Inv(F))))

*For any **F**, **F** is a simple graph function if and only if **F** and Inv(**F**) are graph functions.*

520(T) Val of Inv

```
A - + - - - - + - - - + - + - -
^ S   A - + - - - - - - - - + +
F F ^ ^ @       =
  u F x   ^ D     V           ^
  n       x o ^   a I   V     x
  c         m F   l n ^ a ^ ^
                    v F l F x
```

A(F|SFunc(F)) A(x|@(x,Dom(F))) =(Val(Inv(F),Val(F,x)),x)

*For any simple graph function **F** and for any **x** in Dom(**F**), Val(Inv(**F**),Val(**F**,**x**)) is equal to **x**.*

521(T) Val of Inv

```
A - + - - - - + - - + - - + - -
^ S   A - + - - - - - - - - + +
F F ^ ^ @       =
  u F y   ^ R     V           ^
  n       y a ^   a ^ V       y
  c         n F   l F a I   ^
                      l n ^ y
                        v F
```

A(F|SFunc(F)) A(y|@(y,Ran(F))) =(Val(F,Val(Inv(F),y)),y)

*For any simple graph function **F** and for any **y** in Ran(**F**), Val(**F**,Val(Inv(**F**),**y**)) is equal to **y**.*

523(T) Dom of composite with Inv is Dom

```
A - + - - - + - + - +
^ G   =
G r ^   D         D
  a G   o o       o ^
  p     m   ^ I   m G
  h         G n ^
              v G
```

```
A(G|Graph(G)) =(Dom(o(G,Inv(G))),Dom(G))
```

*If **G** is a graph, then Dom(o(**G**,Inv(**G**))) is equal to Dom(**G**).*

524(T) Composite of F and Inv is Ident of Dom

```
A - + - - + - + - - +
^ S   =
F F ^   o       I
  u F     ^ I   d D
  n       F n ^ e o ^
  c         v F n m F
                t
```

```
A(F|SFunc(F)) =(o(F,Inv(F)),Ident(Dom(F)))
```

*If **F** is a simple graph function, then o(**F**,Inv(**F**)) is equal to Ident(Dom(**F**)).*

525(T) Composite of Inv and F is Ident of Ran

```
A - + - - - + + - - +
^ S   =
F F ^   o       I
  u F     I   ^ d R
  n       n ^ F e a ^
  c       v F   n n F
                t
```

```
A(F|SFunc(F)) =(o(Inv(F),F),Ident(Ran(F)))
```

*If **F** is a simple graph function, then o(Inv(**F**),**F**) is equal to Ident(Ran(**F**)).*

530(T) Composite equal to Ident implies SFunc

```
A - + - - - - - - + - - - + - +
^ F   A - - + - - - + - - - - -
F u ^ ^ X                 S
  n F G   F   =           F ^
  c       u ^   o     I   u F
          n G     ^ ^ d D   n
          c       F G e o ^ c
                      n m F
                      t
```

```
A(F|Func(F)) A(G|X(Func(G),=(o(F,G),Ident(Dom(F))))) SFunc(F)
```

*If **F** and **G** are graph functions and o(**F**,**G**) is equal to Ident(Dom(**F**)), then **F** is a simple graph function.*

536(T) Composite equal to Ident implies Ran equal to Dom

```
A - + - - - - - - - + - - - + - - - + - - - - - + - -
^ F   A - + - - - - - + - - - - - + - - - + - - - - +
F u ^ ^ F   C
  n F G u ^   X                             =
  c     n G     =             =               R   D
        c         o     I       o     I       a ^ o ^
                    ^ ^ d D       ^ ^ d D     n F m G
                    F G e o ^     G F e o ^
                        n m F         n m G
                        t             t
```

A(F|Func(F)) A(G|Func(G))
C(X(=(o(F,G),Ident(Dom(F))),=(o(G,F),Ident(Dom(G)))),=(Ran(F),Dom(G)))

*Let **F** and **G** be graph functions. If o(**F**,**G**) is equal to Ident(Dom(**F**)) and o(**G**,**F**) is equal to Ident(Dom(**G**)), then Ran(**F**) is equal to Dom(**G**).*

539(T) SFunc explanation with composite and Ident

```
A - + - - + - - - - - + - - - +
^ F   D
F u ^   S   E - + - - - + - - -
  n F   F ^ ^ F   =
  c     u F G u ^   o     I
        n     n G     ^ ^ d D
        c     c       F G e o ^
                          n m F
                          t
```

A(F|Func(F)) D(SFunc(F),E(G|Func(G)) =(o(F,G),Ident(Dom(F))))

*Let **F** be a graph function. **F** is a simple graph function if and only if there exists a graph function **G** such that o(**F**, **G**) = Ident(Dom(**F**)).*

540(T) Func definition corollary 4

```
A - - - + - - + - - - - + - - - - - - - - + - - - - - - - - + - - -
^ T D
F     F   X
      u ^   G   A - + - - - - - - - - + - - - - - - - - + - - - - -
      n F   r ^ ^ @       A - - + - - - + - - - - - - - - - - - + -
      c     a F x   ^ D   ^ X               A - - + - - - + - - - +
            p       x o ^ y   S   @         ^ X               =
            h         m F 1     ^   P     ^ y   S   @           ^ ^
                                y   a ^ ^ F 2     ^   P     ^   y y
                                1   i x y         y   a ^ ^ F   1 2
                                    r   1         2   i x y
                                                      r   2
```

A(F) D(Func(F),X(Graph(F),A(x|@(x,Dom(F))) A(y1|X(S(y1),@(Pair(x,y1),F)))
A(y2|X(S(y2),@(Pair(x,y2),F))) =(y1,y2)))

*For any class **F**, **F** is a graph function iff for any **x** in Dom(**F**), and for any set **y1** with Pair of **x**, **y1** in **F**, and for any set **y2** with Pair of **x**, **y2** in **F**, **y1** = **y2**.*

541(T) Graph corollary

```
A - - - + - - - + - - - - - - - - - - -
^ T D
G     G   A - + - - - - - - - - + - - -
      r ^ ^ @     E - + - - - - - - + -
      a G u   ^ ^ ^ S   E - + - - - - +
      p       u G x   ^ ^ S   =
      h               x y   ^   ^ P
                            y   u a ^ ^
                                  i x y
                                  r
```

A(G) D(Graph(G),A(u|@(u,G)) E(x|S(x)) E(y|S(y)) =(u,Pair(x,y)))

701(R) IsSng definition (class is a singleton)

```
A - - - + - - - - - - + - -
^ T D
X     I   E - - - + - - - +
      s ^ ^ T X
      S X x     S   =
      n           ^   ^ S
      g           x   X n ^
                        g x
```

A(X) D(IsSng(X),E(x) X(S(x),=(X,Sng(x))))

*Let **X** be a class. **X** is a singleton if and only if there exists **x** such that **x** is a set and **X** is equal to Sng(**x**).*

702(T) Existence for Cnt (the content of a singleton)

```
A - + - - - - - - + - -
^ I   E - - - + - - - +
X s ^ ^ T X
  S X x     S   =
  n           ^   ^ S
  g           x   X n ^
                    g x
```

A(X|IsSng(X)) E(x) X(S(x),=(X,Sng(x)))

*For any singleton **X**, there exists a set **x** such that **X** is equal to Sng(**x**).*

703(T) Uniqueness for Cnt (the content of a singleton)

```
A - + - - - - - + - - - - - - - + - - - - -
^ I   A - - + - - - + - - - - - - - - - + -
t s ^ ^ X             A - - + - - - + - - +
  S t x   S   =       ^ X           =
  n   1     ^   ^ S   x   S   =         ^ ^
  g         x   t n ^ 2     ^   ^ S     x x
            1     g x       x   t n ^   1 2
                    1       2     g x
                                    2
```

```
A(t|IsSng(t)) A(x1|X(S(x1),=(t,Sng(x1)))) A(x2|X(S(x2),=(t,Sng(x2)))) =(x1,x2)
```

704(F) Cnt definition (the content of a singleton)

```
A - + - - - + - + - - +
^ I   X
X s ^   S     =
  S X     C     ^ S
  n       n ^   X n C
  g       t X     g n ^
                    t X
```

```
A(X|IsSng(X)) X(S(Cnt(X)),=(X,Sng(Cnt(X))))
```

For any singleton X, Cnt(X) is a set, and X is equal to Sng(Cnt(X)).

705(T) Sng corollary

```
A - + - + - +
^ S   @
x   ^   ^ S
    x   x n ^
          g x
```

```
A(x|S(x)) @(x,Sng(x))
```

*For any set **x**, **x** belongs to Sng(**x**).*

708(T) A class is equal to the class of all its elements

```
A - - + - - - +
^ T =
X     ^ K - + -
      X ^ @
        x   ^ ^
            x X
```

```
A(X) =(X,{x|@(x,X)})
```

*For any **X**, **X** is equal to {**x** | **x** ∈ **X**}.*

709(T) Class selection theorem

```
A - - - + - - - - - - - - - + - - - - - - - + - - - - - - - - - - -
^ T A - - + - - - - - - - - - - - - - - - - - - - - - + - - - -
X   ^ D     A - - - - - - - - - - - - - - - - + - - - - - - - +
    Y i ^ ^ ^ T A - - - - - - - - - - - - - - - - - - - - - + -
      s X Y A   ^ T A - + - - - - - - + - - - - - - + - - - - -
                B   ^ @     =
                    x   ^ ^   K - - - - - - + - - - - - - + - ^
                        x X   ^ V                             A
                              v   X             X
                                    @     @       @     @
                                      ^ ^   ^ ^     ^ ^   ^ ^
                                      x X   v A     x Y   v B
```

```
A(X) A(Y|Dis(X,Y)) A(A) A(B) A(x|@(x,X))
=({v|V(X(@(x,X),@(v,A)),X(@(x,Y),@(v,B)))},A)
```

Functions

In the chapter "Graph functions", we have defined, at the level of graphs, many of the fundamental properties of functions. The main reason for this approach is the usual definition of a function.

The usual definition of a function

*Let **A**, **B**, **F** be classes, where **F** is included in **A** $\times$ **B**, and **f** = (**A**, **B**, **F**).*
***f** is a function from **A** to **B** if and only if, for each **x** $\in$ **A**, there exists a unique **y** $\in$ **B** such that (**x**, **y**) $\in$ **F**.*
*In this case, we write **f** : **A** $\to$ **B** and **f**(**x**) = **y**.*

As we can see, a function is an ordered triple of classes. This is very nice, but not possible! Just like for ordered pairs, we can only define a triple of sets. Obviously, we can define a function only for sets, but in a class theory, we consider this to be too restrictive.

Instead of defining a function as a class, we define a function as a (primitive) relation between the classes A, B, F. In other words, the classes A, B, F define a function if they satisfy the necessary conditions. This also means that we no longer have classes that are functions. Our definition of a function as a (primitive) relation generates all the properties of the usual function, but for classes, not only for sets.

801(R) Fun definition (function)

```
A - - - - - - - + - - - - - - - - - + - - - -
^ T A - - - - - - + - - - - - - - - - - - - - +
X   ^ T A - - - - - + - - + - - - + - - - + -
    Y   ^ T D
        F     F       X
              u ^ ^ ^   F   X
              n X Y F   u ^   =       P
                        n F     D   ^ a R   ^
                        c       o ^ X r a ^ Y
                                m F   t n F
```

A(X) A(Y) A(F) D(Fun(X,Y,F),X(Func(F),X(=(Dom(F),X),Part(Ran(F),Y))))

*Let **X**, **Y**, **F** be classes. The classes **X**, **Y**, **F** define a function if and only if **F** is a graph function, Dom(**F**) = **X**, and Ran(**F**) is included in **Y**.*

Although Fun of X, Y, F is a relationship between X, Y, F, it is often used to call F a "function". It would be correct to call F the graph of the function. F is a graph function.

802(T) Fun corollary

```
A - - - - - + - - - - - + - - - - - - - - - - - - -
^ T A - - - - + - - - - - - - - - - + - - - - - - -
X   ^ T A - - - + - - - - - - - + - - - - - - + - +
    Y   ^ F       A - + - - - - - + - - - + - - + -
        F u ^ ^ ^ ^ @     X
          n X Y F x   ^ ^   @         @
                      x X     V     ^   P         ^
                              a ^ ^ Y   a ^ V     F
                              l F x     i x a ^ ^
                                        r   l F x
```

```
A(X) A(Y) A(F|Fun(X,Y,F)) A(x|@(x,X)) X(@(Val(F,x),Y),@(Pair(x,Val(F,x)),F))
```

*Let **X**, **Y**, **F** be classes such that Fun of **X**, **Y**, **F**. For any **x** belonging to **X**, Val(**F**, **x**) belongs to **Y**, and the pair of **x** and Val(**F**, **x**) belongs to **F**.*

803(T) Function accepts a bigger codomain

```
A - - - - - - - - - - - + - - - + - -
^ T A - - - + - - - - - - + - - - - -
X   ^ T A - - + - - - - - - - - - + -
    Y   ^ P     A - - - - - + - - - +
        Z a ^ ^ ^ T C
          r Y Z F     F       F
          t           u ^ ^ ^ u ^ ^ ^
                      n X Y F n X Z F
```

```
A(X) A(Y) A(Z|Part(Y,Z)) A(F) C(Fun(X,Y,F),Fun(X,Z,F))
```

805(T) Fun equality

```
A - - - - - + - - - - + - - - - - - - - - - - - - - - - - - - - - - -
^ T A - - - - + - - - - + - - - - - - - - - - - - - - - - - - - - - -
X   ^ T A - - - + - - - - - - - + - - - - - - - - - - - - + - - - - -
    Y   ^ F       A - - - + - - - + - - - - - - - - - - - - - - - - +
        F u ^ ^ ^ ^ F       D
          n X Y F G u ^ ^ ^   =     A - + - - - - - - + - - - - + - -
                    n X Y G     ^ ^ ^ S   A - + - - - - + - - - - + -
                                F G x   ^ ^ S   D
                                        x y   ^   @         @
                                              y     P     ^   P     ^
                                                    a ^ ^ F   a ^ ^ G
                                                    i x y     i x y
                                                    r         r
```

```
A(X) A(Y) A(F|Fun(X,Y,F)) A(G|Fun(X,Y,G)) D(=(F,G),A(x|S(x)) A(y|S(y))
D(@(Pair(x,y),F),@(Pair(x,y),G)))
```

We consider that the reader has familiarized himself with the text representation of sentences. For this reason, we will give up their representation in natural language, with the exception of cases where we want to draw attention to special problems.

810(T) Fun equality

```
A - - - - - + - - - - + - - - - - - - - - + - - - - - - -
^ T A - - - - + - - - - + - - - - - - - - - - - - - - - -
X   ^ T A - - - + - - - - - - - + - - - - - - - + - - - -
    Y   ^ F       A - - - + - - - + - - - - - - - - - + -
        F u ^ ^ ^ ^ F       D
          n X Y F G u ^ ^ ^   =     A - + - - - - + - - +
                    n X Y G     ^ ^ ^ @     =
                                F G x   ^ ^   V     V
                                        x X   a ^ ^ a ^ ^
                                              l F x l G x
```

`A(X) A(Y) A(F|Fun(X,Y,F)) A(G|Fun(X,Y,G)) D(=(F,G),A(x|@(x,X))`
`=(Val(F,x),Val(G,x)))`

*Let **X**, **Y**, **F**, **G** be classes such that we have Fun of **X**, **Y**, **F** and Fun of **X**, **Y**, **G**.*
***F** = **G** if and only if for any **x** belonging to **X**, Val(**F**, **x**) = Val(**G**, **x**).*

829(T) Fun equality

```
A - - - - - + - - - - + - - - - - - - + - - - - + - - - - - - -
^ T A - - - - + - - - - + - - - - - - - - - - - - - - - - - - -
X   ^ T A - - - + - - - - - - - + - - - - - - - - - - + - - - -
    Y   ^ F       A - - - + - - - + - - - - - - - - - - - - + -
        F u ^ ^ ^ ^ F       D
          n X Y F G u ^ ^ ^   =     =
                    n X Y G     ^ ^   ^ K - - + - - - - + - - +
                                F G   X ^ X
                                        x   @     =
                                              ^ ^   V     V
                                              x X   a ^ ^ a ^ ^
                                                    l F x l G x
```

`A(X) A(Y) A(F|Fun(X,Y,F)) A(G|Fun(X,Y,G))`
`D(=(F,G),=(X,{x|X(@(x,X),=(Val(F,x),Val(G,x)))}))`

*Let **X**, **Y**, **F**, **G** be classes such that we have Fun of **X**, **Y**, **F** and Fun of **X**, **Y**, **G**.*
***F** = **G** if and only if **X** is equal to the class of all **x** in **X** and Val(**F**, **x**) = Val(**G**, **x**).*

832(T) Ran explanation for Fun

```
A - - - - - + - - - - - - - - - - - - - - + - - - - -
^ T A - - - - + - - - - + - - - - - - - - - - - - - -
X   ^ T A - - - + - - - - - - - - + - - - - - - + - -
    Y   ^ F       A - + - - - + - - - - - - - - - - +
        F u ^ ^ ^ ^ @     D
          n X Y F y   ^ ^   @       E - + - - - - + -
                      y Y     ^ R   ^ @     =
                              y a ^ x   ^ ^   V     ^
                                n F     x X   a ^ ^ y
                                              l F x
```

`A(X) A(Y) A(F|Fun(X,Y,F)) A(y|@(y,Y)) D(@(y,Ran(F)),E(x|@(x,X)) =(Val(F,x),y))`

*Let **X**, **Y**, **F** be classes such that we have Fun of **X**, **Y**, **F**, and let **y** be an element of **Y**. **y** belongs to Ran of **F** if and only if there exists **x** belonging to **X** such that Val(**F**, **x**) = **y**.*

901(R) Inj definition (injective function)

```
A - - - - - - - + - - - - + - - - - - + - - - + - - - - - - - - - - -
^ T A - - - - - - + - - - - + - - - - - - - - - - - - - - - - - - - -
X   ^ T A - - - - - + - - - - + - - - - - - - - - - - + - - + - - - -
    Y   ^ T D
        F     I       X
              n ^ ^ ^   F       A - + - - - - - - - - - + - - - - + -
              j X Y F   u ^ ^ ^ ^ @     A - + - - - - - - - - + - - +
                        n X Y F x   ^ ^ ^ @     C
                                1   x X x   ^ ^   =             =
                                    1   2   x X     V     V       ^ ^
                                            2       a ^ ^ a ^ ^   x x
                                                    l F x l F x   1 2
                                                        1     2
```

```
A(X) A(Y) A(F) D(Inj(X,Y,F),X(Fun(X,Y,F),A(x1|@(x1,X)) A(x2|@(x2,X))
C(=(Val(F,x1),Val(F,x2)),=(x1,x2))))
```

*Let **X**, **Y**, **F** be classes. **X**, **Y**, **F** form an injective function if and only if **X**, **Y**, **F** form a function, and for any $\boldsymbol{x_1}$, $\boldsymbol{x_2}$ belonging to **X**, if Val(**F**, $\boldsymbol{x_1}$) = Val(**F**, $\boldsymbol{x_2}$), then $\boldsymbol{x_1} = \boldsymbol{x_2}$.*

902(R) Sur definition (surjective function)

```
A - - - - - - - + - - - - + - - - - - - - - - + - - - - -
^ T A - - - - - - + - - - - + - - - - + - - - - - - - - -
X   ^ T A - - - - - + - - - - + - - - - - - - - - - - + -
    Y   ^ T D
        F     S       X
              u ^ ^ ^   F       A - + - - - - - - + - - -
              r X Y F   u ^ ^ ^ ^ @     E - + - - - - - +
                        n X Y F y   ^ ^ ^ @     =
                                    y Y x   ^ ^   ^ V
                                            x X   y a ^ ^
                                                    l F x
```

```
A(X) A(Y) A(F) D(Sur(X,Y,F),X(Fun(X,Y,F),A(y|@(y,Y)) E(x|@(x,X)) =(y,Val(F,x)))))
```

*Let **X**, **Y**, **F** be classes. Sur of **X**, **Y**, **F** if and only if Fun of **X**, **Y**, **F**, and for any **y** in **Y** there exists **x** in **X** so that **y** = Val(**F**, **x**).*

903(R) Bij definition (bijective function)

```
A - - - - - - - + - - - - + - - - + - -
^ T A - - - - - - + - - - - + - - - + -
X   ^ T A - - - - - + - - - - + - - - +
    Y   ^ T D
        F     B       X
              i ^ ^ ^   I       S
              j X Y F   n ^ ^ ^ u ^ ^ ^
                        j X Y F r X Y F
```

```
A(X) A(Y) A(F) D(Bij(X,Y,F),X(Inj(X,Y,F),Sur(X,Y,F)))
```

*Let **X**, **Y**, **F** be classes.*
*Bij of **X**, **Y**, **F** if and only if Inj of **X**, **Y**, **F** and Sur of **X**, **Y**, **F**.*

911(T) Sur corollary

```
A - - - - - - - + - - - - + - - - - - -
^ T A - - - - - - + - - - - + - - + - -
X   ^ T A - - - - - + - - - - + - - - +
    Y   ^ T D
        F     S       X
              u ^ ^ ^   F       =
              r X Y F   u ^ ^ ^   ^ R
                        n X Y F   Y a ^
                                    n F
```

`A(X) A(Y) A(F) D(Sur(X,Y,F),X(Fun(X,Y,F),=(Y,Ran(F))))`

*Let **X**, **Y**, **F** be classes. Sur(**X**, **Y**, **F**) if and only if Fun(**X**, **Y**, **F**) and **Y** = Ran(**F**).*

912(T) Func of F is Fun of Dom, Ran, F

```
A - - - + - - + - + +
^ T D
F     F   F
      u ^ u D   R   ^
      n F n o ^ a ^ F
      c     m F n F
```

`A(F) D(Func(F),Fun(Dom(F),Ran(F),F))`

***F** is a graph function if and only if Fun(Dom(**F**), Ran(**F**), **F**).*

913(T) Func of F is Sur of Dom, Ran, F

```
A - - - + - - + - + +
^ T D
F     F   S
      u ^ u D   R   ^
      n F r o ^ a ^ F
      c     m F n F
```

`A(F) D(Func(F),Sur(Dom(F),Ran(F),F))`

***F** is a graph function if and only if Sur(Dom(**F**), Ran(**F**), **F**).*

914(T) SFunc of F is Inj of Dom, Ran, F

```
A - - - + - - + - + +
^ T D
F     S   I
      F ^ n D   R   ^
      u F j o ^ a ^ F
      n     m F n F
      c
```

`A(F) D(SFunc(F),Inj(Dom(F),Ran(F),F))`

***F** is a simple graph function if and only if Inj(Dom(**F**), Ran(**F**), **F**).*

915(T) SFunc of F is Bij of Dom, Ran, F

```
A - - - + - - + - + +
^ T D
F     S   B
      F ^ i D   R   ^
      u F j o ^ a ^ F
      n     m F n F
      c
```

`A(F) D(SFunc(F),Bij(Dom(F),Ran(F),F))`

__F__ is a simple graph function if and only if Bij(Dom(__F__), Ran(__F__), __F__).

917(T) Bij explanation with SFunc

```
A - - - - - - - + - - - - - - - - - - + - - - -
^ T A - - - - - - + - - - - - - - - - - - - - +
X   ^ T A - - - - - + - - + - - - + - - - + -
    Y   ^ T D
        F     B       X
              i ^ ^ ^   S   X
              j X Y F   F ^   =       =
                        u F     D   ^   R   ^
                        n       o ^ X   a ^ Y
                        c       m F     n F
```

`A(X) A(Y) A(F) D(Bij(X,Y,F),X(SFunc(F),X(=(Dom(F),X),=(Ran(F),Y))))`

Let __X__, __Y__, __F__ be classes. Bij(__X__, __Y__, __F__) if and only if __F__ is a simple graph function, Dom(__F__) = __X__, and Ran(__F__) = __Y__.

918(T) OneToOne explanation with Bij

```
A - - - - - + - - - - - + - -
^ T A - - - - + - - - - - + -
X   ^ T D
    Y     O     E - - - - +
          n ^ ^ ^ T B
          e X Y F   i ^ ^ ^
          T         j X Y F
          o
          O
          n
          e
```

`A(X) A(Y) D(OneToOne(X,Y),E(F) Bij(X,Y,F))`

Let __X__, __Y__ be classes. OneToOne of __X__, __Y__ if and only if there exists a class __F__ such that Bij(__X__,__Y__,__F__).

919(T) Fun of X, X, Ident of X

```
A - - + + - +
^ T F
X   u ^ ^ I
    n X X d ^
          e X
          n
          t
```

```
A(X) Fun(X,X,Ident(X))
```

922(T) Bij of X, X, Ident of X

```
A - - + + - +
^ T B
X   i ^ ^ I
    j X X d ^
          e X
          n
          t
```

```
A(X) Bij(X,X,Ident(X))
```

930(T) Const is Fun

```
A - - - - - - - - + - - + -
^ T A - - - - + - - + - - -
A   ^ T A - + - - - - - - +
    B   ^ @     F
        b   ^ ^ u ^ ^ C
            b B n A B o ^ ^
                      n A b
                      s
                      t
```

```
A(A) A(B) A(b|@(b,B)) Fun(A,B,Const(A,b))
```

933(T) Fun restriction is Fun

```
A - - - - - + - - - - - - - + - - - - -
^ T A - - - - + - - - - - - - + - - -
X   ^ T A - - - + - - - - - - - - + -
    Y   ^ F       A - - - - + - - - +
        F u ^ ^ ^ ^ T F
          n X Y F A   u I     ^ D
                      n   ^ ^ Y o ^ ^
                          X A   m F A
                                I
                                n
                                t
```

```
A(X) A(Y) A(F|Fun(X,Y,F)) A(A) Fun(I(X,A),Y,DomInt(F,A))
```

935(T) Disjoint domains and the union of graph functions

```
A - - - + - - - - - + - - - - - - - - - + - - - - -
^ T A - - + - - - - - - - - - + - - - - - - + - - -
X   ^ D     A - - - - + - - - - + - - - - - + - - -
1   X i ^ ^ ^ T A - - - + - - - - - - - - - - - + -
    2 s X X Y   ^ F       A - - - + - - - - - - - +
        1 2     F u ^ ^ ^ ^ F       F
                  n X Y F G u ^ ^ ^ u U     ^ U
                    1       n X Y G n   ^ ^ Y   ^ ^
                              2         X X     F G
                                        1 2
```

```
A(X1) A(X2|Dis(X1,X2)) A(Y) A(F|Fun(X1,Y,F)) A(G|Fun(X2,Y,G))
Fun(U(X1,X2),Y,U(F,G))
```

*Let **X1**, **X2** be disjoint classes and **Y**, **F**, **G** be classes with Fun of **X1**, **Y**, **F**, and Fun of **X2**, **Y**, **G**. Then Fun of U(**X1**, **X2**), **Y**, U(**F**, **G**).*

939(T) Composite of functions is a function

```
A - - - - - - - + - - - - - - - - + - - - -
^ T A - - - - - - + - - - + - - - - - - - -
X   ^ T A - - - - - - - - - + - - - + - - -
    Y   ^ T A - - - + - - - - - - - - - + -
        Z   ^ F       A - - - + - - - - - +
            F u ^ ^ ^ ^ F       F
              n X Y F G u ^ ^ ^ u ^ ^ o
                        n Y Z G n X Z   ^ ^
                                        F G
```

```
A(X) A(Y) A(Z) A(F|Fun(X,Y,F)) A(G|Fun(Y,Z,G)) Fun(X,Z,o(F,G))
```

*Let **X**, **Y**, **Z** be classes with Fun of **X**, **Y**, **F** and Fun of **Y**, **Z**, **G**. Then Fun of **X**, **Z**, o(**F**,**G**).*

940(T) Val of composite of functions

```
A - - - - - - - + - - - - - - - - - - + - - - - - - - - - - -
^ T A - - - - - - + - - - + - - - - - - - - - - - - - - - - -
X   ^ T A - - - - - - - - - + - - - - - - - - - - - - - - - -
    Y   ^ T A - - - + - - - - - - - - - - - - + - - - - - + -
        Z   ^ F       A - - - + - - - - - - - - + - - + - - -
            F u ^ ^ ^ ^ F       A - + - - - - - - + - - - - +
              n X Y F G u ^ ^ ^ ^ @     =
                        n Y Z G x   ^ ^   V         V
                                    x X   a o     ^ a ^ V
                                          l   ^ ^ x l G a ^ ^
                                              F G       l F x
```

```
A(X) A(Y) A(Z) A(F|Fun(X,Y,F)) A(G|Fun(Y,Z,G)) A(x|@(x,X))
=(Val(o(F,G),x),Val(G,Val(F,x)))
```

*Let **X**, **Y**, **Z** be classes with Fun of **X**, **Y**, **F** and Fun of **Y**, **Z**, **G**. For any **x** belonging to **X**, Val(o(**F**, **G**), **x**) = Val(**G**, Val(**F**, **x**)).*

943(T) Inv of a Bij is Func

```
 A - - - - - + - - - - -
 ^ T A - - - - + - - - -
 X   ^ T A - - - + - - +
     Y   ^ B       F
         F i ^ ^ ^ u I
           j X Y F n n ^
                   c v F
```

```
A(X) A(Y) A(F|Bij(X,Y,F)) Func(Inv(F))
```

944(T) Inv of a Bij is Fun

```
 A - - - - - + - - - - + - -
 ^ T A - - - - + - - + - - -
 X   ^ T A - - - + - - - - +
     Y   ^ B       F
         F i ^ ^ ^ u ^ ^ I
           j X Y F n Y X n ^
                         v F
```

```
A(X) A(Y) A(F|Bij(X,Y,F)) Fun(Y,X,Inv(F))
```

948(T) Inv of Bij is Bij

```
 A - - - - - + - - - - + - -
 ^ T A - - - - + - - + - - -
 X   ^ T A - - - + - - - - +
     Y   ^ B       B
         F i ^ ^ ^ i ^ ^ I
           j X Y F j Y X n ^
                         v F
```

```
A(X) A(Y) A(F|Bij(X,Y,F)) Bij(Y,X,Inv(F))
```

949(T) Val of Inv

```
 A - - - - - + - - - - - + - - - - - - - -
 ^ T A - - - - + - - - - - - - - - - - - -
 X   ^ T A - - - + - - - - - - - - + - + - -
     Y   ^ B       A - + - - - - - - - - + +
         F i ^ ^ ^ ^ @     =
           j X Y F x   ^ ^   V           ^
                       x X   a I   V     x
                             l n ^ a ^ ^
                               v F l F x
```

```
A(X) A(Y) A(F|Bij(X,Y,F)) A(x|@(x,X)) =(Val(Inv(F),Val(F,x)),x)
```

950(T) Val of Inv

```
A - - - - - + - - - - - - - - - - - - - -
^ T A - - - - + - - - - + - - - - - - - -
X   ^ T A - - - + - - - - - - - + - - + - -
    Y   ^ B       A - + - - - - - - - + +
        F i ^ ^ ^ ^ @     =
          j X Y F y   ^ ^   V           ^
                      y Y   a ^ V       y
                            l F a I   ^
                                l n ^ y
                                  v F
```

A(X) A(Y) A(F|Bij(X,Y,F)) A(y|@(y,Y)) =(Val(F,Val(Inv(F),y)),y)

951(T) Composite of a bijection with its inverse

```
A - - - - - + - - - - - - - - +
^ T A - - - - + - - - - - - - -
X   ^ T A - - - + - - + - + - -
    Y   ^ B       =
        F i ^ ^ ^   o       I
          j X Y F     ^ I   d ^
                      F n ^ e X
                        v F n
                            t
```

A(X) A(Y) A(F|Bij(X,Y,F)) =(o(F,Inv(F)),Ident(X))

952(T) Composite of the inverse of a bijection with the bijection

```
A - - - - - + - - - - - - - - -
^ T A - - - - + - - - - - - - +
X   ^ T A - - - + - - - + + - -
    Y   ^ B       =
        F i ^ ^ ^   o       I
          j X Y F     I   ^ d ^
                      n ^ F e Y
                      v F   n
                            t
```

A(X) A(Y) A(F|Bij(X,Y,F)) =(o(Inv(F),F),Ident(Y))

954(T) Invertible is Bij

```
A - - - - - + - - - - - + - - - + - -
^ T A - - - - + - - - + - - - - - - + -
X   ^ T A - - - + - - - - - - + - - - - +
    Y   ^ F       C
        F u ^ ^ ^   F         B
          n X Y F   u ^ ^ I   i ^ ^ ^
                    n Y X n ^ j X Y F
                            v F
```

A(X) A(Y) A(F|Fun(X,Y,F)) C(Fun(Y,X,Inv(F)),Bij(X,Y,F))

*Let **X**, **Y** , **F** be clases with Fun of **X**, **Y**, **F**. If Fun of **Y**, **X**, Inv(**F**), then Bij of **X**, **Y**, **F**.*

955(T) Inv of Invertible is Bij

```
A - - - - - + - - - - - + - - - - + - -
^ T A - - - - + - - - + - - - - + - - -
X   ^ T A - - - + - - - - - + - - - - +
    Y   ^ F       C
        F u ^ ^ ^   F         B
          n X Y F   u ^ ^ I   i ^ ^ I
                    n Y X n ^ j Y X n ^
                          v F       v F
```

`A(X) A(Y) A(F|Fun(X,Y,F)) C(Fun(Y,X,Inv(F)),Bij(Y,X,Inv(F)))`

957(T) Composite equal to Ident implies Inj

```
A - - - - - + - - - - - + - - - - - - - + - + - -
^ T A - - - - + - - - + - - - - - - - - - - - + -
X   ^ T A - - - + - - - - - - - - + - - - - - - +
    Y   ^ F       A - - - + - - - - + - - - - - -
        F u ^ ^ ^ ^ F       C
          n X Y F G u ^ ^ ^   =           I
                    n Y X G     o     I   n ^ ^ ^
                                  ^ ^ d ^ j X Y F
                                  F G e X
                                      n
                                      t
```

`A(X) A(Y) A(F|Fun(X,Y,F)) A(G|Fun(Y,X,G)) C(=(o(F,G),Ident(X)),Inj(X,Y,F))`

*Let **X**, **Y**, **F**, **G** be classes with Fun of **X**, **Y**, **F** and Fun of **Y**, **X**, **G**. If o(**F**,**G**) is equal to Ident(**X**), then Inj of **X**, **Y**, **F**.*

958(T) Composite equal to Ident implies Sur

```
A - - - - - + - - - - - + - - - - - - - - - + - -
^ T A - - - - + - - - + - - - - - - - - + - - + -
X   ^ T A - - - + - - - - - - - - - + - - - - - +
    Y   ^ F       A - - - + - - - + - - - - - - -
        F u ^ ^ ^ ^ F       C
          n X Y F G u ^ ^ ^   =           S
                    n Y X G     o     I   u ^ ^ ^
                                  ^ ^ d ^ r X Y F
                                  G F e Y
                                      n
                                      t
```

`A(X) A(Y) A(F|Fun(X,Y,F)) A(G|Fun(Y,X,G)) C(=(o(G,F),Ident(Y)),Sur(X,Y,F))`

*Let **X**, **Y**, **F**, **G** be classes with Fun of **X**, **Y**, **F** and Fun of **Y**, **X**, **G**. If o(**G**, **F**) is equal to Ident(**Y**), then Sur of **X**, **Y**, **F**.*

959(T) Composite equal to Ident implies Bij

```
A - - - - - + - - - - - + - - - - - - - - - + - - - - - - - + - -
^ T A - - - - + - - - + - - - - - - - - - - - - - - - - + - - + -
X   ^ T A - - - + - - - - - - - - - - + - - - - - - + - - - - - +
    Y   ^ F       A - - - + - - - - - - + - - - - + - - - - - - -
        F u ^ ^ ^ ^ F       C
          n X Y F G u ^ ^ ^   X                         B
                    n Y X G     =           =           i ^ ^ ^
                                  o     I     o     I   j X Y F
                                    ^ ^ d ^     ^ ^ d ^
                                    F G e X     G F e Y
                                        n           n
                                        t           t
```

```
A(X) A(Y) A(F|Fun(X,Y,F)) A(G|Fun(Y,X,G))
C(X(=(o(F,G),Ident(X)),=(o(G,F),Ident(Y))),Bij(X,Y,F))
```

*Let **X**, **Y**, **F**, **G** be classes with Fun of **X**, **Y**, **F** and Fun of **Y**, **X**, **G**. If o(**F**,**G**) is equal to Ident(**X**) and o(**G**, **F**) is equal to Ident(**Y**), then Bij of **X**, **Y**, **F**.*

964(T) Composite of surjections is a surjection

```
A - - - - - - - + - - - - - - - - + - - - -
^ T A - - - - - - + - - - + - - - - - - - -
X   ^ T A - - - - - - - - - + - - - + - - -
    Y   ^ T A - - - + - - - - - - - - - + -
        Z   ^ S       A - - - + - - - - - +
            F u ^ ^ ^ ^ S       S
              r X Y F G u ^ ^ ^ u ^ ^ o
                        r Y Z G r X Z   ^ ^
                                        F G
```

```
A(X) A(Y) A(Z) A(F|Sur(X,Y,F)) A(G|Sur(Y,Z,G)) Sur(X,Z,o(F,G))
```

*Let **X**, **Y**, **Z**, **F**, **G** be classes. If Sur of **X**, **Y**, **F** and Sur of **Y**, **Z**, **G**, then Sur of **X**, **Z**, o(**F**,**G**).*

965(T) Composite of bijections is a bijection

```
A - - - - - - - + - - - - - - - - + - - - -
^ T A - - - - - - + - - - + - - - - - - - -
X   ^ T A - - - - - - - - - + - - - + - - -
    Y   ^ T A - - - + - - - - - - - - - + -
        Z   ^ B       A - - - + - - - - - +
            F i ^ ^ ^ ^ B       B
              j X Y F G i ^ ^ ^ i ^ ^ o
                        j Y Z G j X Z   ^ ^
                                        F G
```

```
A(X) A(Y) A(Z) A(F|Bij(X,Y,F)) A(G|Bij(Y,Z,G)) Bij(X,Z,o(F,G))
```

*Let **X**, **Y**, **Z**, **F**, **G** be classes. If Bij of **X**, **Y**, **F** and Bij of **Y**, **Z**, **G**, then Bij of **X**, **Z**, o(**F**,**G**).*

966(T) Inv of Composite of Bij is composite of Inv

```
A - - - - - - - + - - - - - - - - - - + - - - - - - - - - - - - - -
^ T A - - - - - - + - - - + - - - - - - - - - - - - - - - - - - - -
X   ^ T A - - - - - - - - - + - - - + - - - - - - - - - - - - - - -
    Y   ^ T A - - - + - - - - - - - - - - - + - - - - + - - - - - +
        Z   ^ B       A - - - + - - - - - - - + - - - - + - - + - -
            F i ^ ^ ^ ^ B       X
              j X Y F G i ^ ^ ^   B             =
                        j Y Z G   i ^ ^ I         I     o
                                  j Z X n o       n o     I   I
                                        v   ^ ^   v   ^ ^   n ^ n ^
                                            F G       F G   v G v F
```

A(X) A(Y) A(Z) A(F|Bij(X,Y,F)) A(G|Bij(Y,Z,G))
X(Bij(Z,X,Inv(o(F,G))),=(Inv(o(F,G)),o(Inv(G),Inv(F))))

968(T) Fun and Ran

```
A - - - - - + - - - + - - -
^ T A - - - - + - - - - - -
X   ^ T A - - - + - - - + +
    Y   ^ F       F
        F u ^ ^ ^ u ^ R   ^
          n X Y F n X a ^ F
                      n F
```

A(X) A(Y) A(F|Fun(X,Y,F)) Fun(X,Ran(F),F)

969(T) Fun with Ran as codomain is Sur

```
A - - - - - + - - - + - - -
^ T A - - - - + - - - - - -
X   ^ T A - - - + - - - + +
    Y   ^ F       S
        F u ^ ^ ^ u ^ R   ^
          n X Y F r X a ^ F
                      n F
```

A(X) A(Y) A(F|Fun(X,Y,F)) Sur(X,Ran(F),F)

971(T) Inj with Ran as codomain is Bij

```
A - - - - - + - - - + - - -
^ T A - - - - + - - - - - -
X   ^ T A - - - + - - - + +
    Y   ^ I       B
        F n ^ ^ ^ i ^ R   ^
          j X Y F j X a ^ F
                      n F
```

A(X) A(Y) A(F|Inj(X,Y,F)) Bij(X,Ran(F),F)

975(T) Empty domain implies empty graph

```
A - - - - - + - - - - + - -
^ T A - - - - + - - - - - -
X   ^ T A - - - + - - - - +
    Y   ^ F       C
        F u ^ ^ ^   E   E
          n X Y F   m ^ m ^
                    p X p F
                    t   t
                    y   y
```

`A(X) A(Y) A(F|Fun(X,Y,F)) C(Empty(X),Empty(F))`

976(T) Fun of 0, Y, 0

```
A - - - + -
^ T F
Y   u 0 ^ 0
    n   Y
```

`A(Y) Fun(0,Y,0)`

978(T) Inj of 0, Y, 0

```
A - - - + -
^ T I
Y   n 0 ^ 0
    j   Y
```

`A(Y) Inj(0,Y,0)`

986(T) Partial reverse for surjections

```
A - + - - - - + - - - - - + - - - - - - -
^ S   A - - - - + - - - + - - - - - - - +
X   ^ ^ T A - - - + - - - - - - - - + - -
    X Y   ^ S       E - - - + - - + - - -
          F u ^ ^ ^ ^ F       =
            r X Y F G u ^ ^ ^   o     I
                      n Y X G     ^ ^ d ^
                                  G F e Y
                                      n
                                      t
```

`A(X|S(X)) A(Y) A(F|Sur(X,Y,F)) E(G|Fun(Y,X,G)) =(o(G,F),Ident(Y))`

*Let **X** be a set and **Y** a class. For any **F** with Sur(**X**, **Y**, **F**), there exists a **G** with Fun(**Y**, **X**, **G**) such that o(**G**, **F**) is equal to Ident(**Y**).*

990(T) Inj reverse

```
A - - + - - - - + - - - - + - - - - - + - - - - - - +
^ N     A - - - - + - - - - + - - - + - - - - - - - -
X   E   ^ T A - - - + - - - - + - - - - - - - + - - -
    m ^ Y   ^ F       D
    p X     F u ^ ^ ^   I       E - - - + - - - + - -
    t         n X Y F   n ^ ^ ^ ^ F       =
    y                   j X Y F G u ^ ^ ^   o     I
                                  n Y X G     ^ ^ d ^
                                              F G e X
                                                  n
                                                  t
```

A(X|N(Empty(X))) A(Y) A(F|Fun(X,Y,F)) D(Inj(X,Y,F),E(G|Fun(Y,X,G))
=(o(F,G),Ident(X)))

*Let **X** be a not empty class and **Y**, **F be** classes with Fun(**X**, **Y**, **F**). Inj(**X**, **Y**, **F**) if and only if there exists **G** with Fun(**Y**, **X**, **G**) such that o(**F**, **G**) is equal to Ident(**X**).*

991(T) Replacement corollary for Sur

```
A - + - - - - + - - - -
^ S   A - - - - + - - +
X   ^ ^ T A - - - + - -
    X Y   ^ S       S
          F u ^ ^ ^   ^
            r X Y F   Y
```

A(X|S(X)) A(Y) A(F|Sur(X,Y,F)) S(Y)

992(T) Corollary of the axiom of replacement

```
A - + - - - - + - - - -
^ S   A - - - - + - - +
X   ^ ^ T A - - - + - -
    X Y   ^ B       S
          F i ^ ^ ^   ^
            j X Y F   Y
```

A(X|S(X)) A(Y) A(F|Bij(X,Y,F)) S(Y)

1004(T) Sur of 0, 0, 0

```
S
u 0 0 0
r
```

Sur(0,0,0)

1006(T) Sur of 0, Y, 0 iff Y = 0

```
A - - - - + - - + -
^ T D
Y     S       =
      u 0 ^ 0   ^ 0
      r   Y     Y
```

```
A(Y) D(Sur(0,Y,0),=(Y,0))
```

Relations

Usually, we say that R is a binary relation on/in X if R is included in the Cartesian product of X and X. In this case, if x and y are two elements of X, we say that x, y are in relation R if the ordered pair (x, y) belongs to R. Instead of $(x, y) \in R$, we can write x R y.

Because R is included in the Cartesian product $X \times X$, R is a graph. This observation allows us to use some properties of graphs (transitivity, symmetry) to define properties of binary relations.

1051(R) Rel definition (R is a relation in X)

```
A - - - - - + - - - - + +
^ T A - - - - + - + - - -
X   ^ T D
    R     R     P
          e ^ ^ a ^ C
          l X R r R a ^ ^
                t   r X X
                    t
```

```
A(X) A(R) D(Rel(X,R),Part(R,Cart(X,X)))
```

*Let **X**, **R** be classes. **R** is a relation in **X** if and only if **R** is included in Cart(**X**, **X**).*

1052(T) Rel is Graph

```
A - - - + - - -
^ T A - - + - +
X   ^ R     G
    R e ^ ^ r ^
      l X R a R
            p
            h
```

```
A(X) A(R|Rel(X,R)) Graph(R)
```

1053(T) Part of Rel is Rel

```
A - - - + - - - - - - - + -
^ T A - - + - - - + - - -
X   ^ R     A - + - - - +
    R e ^ ^ ^ P     R
      l X R Q a ^ ^ e ^ ^
              r Q R l X Q
              t
```

```
A(X) A(R|Rel(X,R)) A(Q|Part(Q,R)) Rel(X,Q)
```

1054(R) Ref definition (reflexive relation)

```
A - - - - - + - - - + - - - - + - - - - -
^ T A - - - - + - - - + - - - - - - - - +
X   ^ T D
    R     R     X
          e ^ ^   R     A - + - - - + + -
          f X R   e ^ ^ ^ @     @
                  l X R x   ^ ^   P     ^
                            x X   a ^ ^ R
                                  i x x
                                  r
```

`A(X) A(R) D(Ref(X,R),X(Rel(X,R),A(x|@(x,X)) @(Pair(x,x),R)))`

*Let **X**, **R** be classes. **R** is a reflexive relation in **X** if and only if R is a relation in X and for any **x** belonging to **X**, Pair(**x**, **x**) belongs to **R**.*

1055(R) Equ definition (equivalence relation)

```
A - - - - - + - - - + - - - - - -
^ T A - - - - + - - - + - - + - +
X   ^ T D
    R     E     X
          q ^ ^   R     X
          u X R   e ^ ^   T   S
                  f X R   r ^ y ^
                          a R m R
```

`A(X) A(R) D(Equ(X,R),X(Ref(X,R),X(Tra(R),Sym(R))))`

*Let **X**, **R** be classes. **R** is an equivalence relation in **X** if and only if **R** is a reflexive relation in **X**, **R** is a transitive graph, and **R** is a symmetric graph.*

1056(R) Ord definition (X is (partially) ordered by R)

```
A - - - - - + - - - + - - - - - -
^ T A - - - - + - - - + - - + - +
X   ^ T D
    R     O     X
          r ^ ^   R     X
          d X R   e ^ ^   T   A
                  f X R   r ^ n ^
                          a R t R
```

`A(X) A(R) D(Ord(X,R),X(Ref(X,R),X(Tra(R),Ant(R))))`

*Let **X**, **R** be classes. **X** is ordered by **R** if and only if **R** is a reflexive relation in **X**, **R** is a transitive graph, and **R** is an antisymmetric graph.*

Note that transitivity, symmetry, and antisymmetry are defined for graphs, while reflexivity is defined for a pair of classes that form a relation (Rel(X, R)). Since reflexivity is used to define equivalence relations and order relations, it is not correct to say that R is an equivalence relation. We must say that X and R form an equivalence relation (Equ(X, R)); similarly for the order relation (Ord(X, R)).

1068(F) EqCl definition (equivalence class)

```
A - - - + - - - - + - - + - - - - - - + - - - - -
^ T A - - + - - - - - - - - + - - - - - - - - - - +
X   ^ E     A - + - - - - - - + - - - - - - - + - -
    R q ^ ^ ^ @     =
      u X R x   ^ ^   E       K - - + - - - - + -
                x X   q ^ ^ ^ ^ X
                      C X R x y   @     @
                      l             ^ ^   P     ^
                                    y X   a ^ ^ R
                                          i x y
                                          r
```

`A(X) A(R|Equ(X,R)) A(x|@(x,X)) =(EqCl(X,R,x),{y|X(@(y,X),@(Pair(x,y),R))})`

*Let **X** be a class, **R** an equivalence relation in **X**, and **x** an element of **X**.*
*EqCl(**X**, **R**, **x**) is the class of any **y** in **X** with Pair(**x**, **y**) in **R**.*

From the definition above it can be seen that an equivalence class depends on three arguments: X, R, and x (EqCl(X, R, x)). Therefore, the expression "equivalence class of x" is incorrect.

1069(T) EqCl explanation

```
A - - - + - - - - + - - - - - - - + - - - - - + - - - - -
^ T A - - + - - - - - - - - - - - - - + - - - - - - - - - +
X   ^ E     A - + - - - - - - - - - - + - - - - - - + - -
    R q ^ ^ ^ @     A - - - + - - - - - - - + - - - - + -
      u X R x   ^ ^ ^ T D
                x X y     @           X
                            ^ E         @     @
                            y q ^ ^ ^     ^ ^   P     ^
                              C X R x     y X   a ^ ^ R
                              l                 i x y
                                                r
```

`A(X) A(R|Equ(X,R)) A(x|@(x,X)) A(y) D(@(y,EqCl(X,R,x)),X(@(y,X),@(Pair(x,y),R)))`

*Let **X** be a class, **R** an equivalence relation in **X**, **x** in **X**, and **y** a class. **y** belongs to EqCl(**X**,**R**,**x**) if and only if **y** belongs to **X** and the pair of **x**, **y** belongs to **R**.*

1071(T) Equ corollary Sym

```
A - - - + - - - - - - - - - - - - - - - - - - -
^ T A - - + - - - - - - - - - - - + - - - - - +
X   ^ E     A - + - - - - - - - + - - - - - - + -
    R q ^ ^ ^ S   A - + - - - - + - - - - + - -
      u X R x   ^ ^ S   D
                x y   ^   @         @
                      y     P     ^   P     ^
                            a ^ ^ R   a ^ ^ R
                            i x y     i y x
                            r         r
```

`A(X) A(R|Equ(X,R)) A(x|S(x)) A(y|S(y)) D(@(Pair(x,y),R),@(Pair(y,x),R))`

1073(T) Belonging to its own equivalence class

```
A - - - + - - - - + - - - + - -
^ T A - - + - - - - - - - - - + -
X   ^ E     A - + - - + - - - +
    R q ^ ^ ^ @     @
      u X R x   ^ ^   ^ E
                x X   x q ^ ^ ^
                        C X R x
                        l
```

```
A(X) A(R|Equ(X,R)) A(x|@(x,X)) @(x,EqCl(X,R,x))
```

In the following theorem, we will see that one of the main qualities of the equivalence classes can be formulated as follows: x1 R x2 if and only if the equivalence class of x1 is equal to the equivalence class of x2. Sorry! This is how it is usually written...

1075(T) Equivalence is equality of equivalence classes

```
A - - - + - - - - + - - - + - - - - - - - - + - - - + - -
^ T A - - + - - - - - - - - - - - - - + - - - + - - - + -
X   ^ E     A - + - - - - - - - - + - - - - - - + - - - -
    R q ^ ^ ^ @     A - + - - - - - - + - - - - - - - - - +
      u X R x   ^ ^ ^ @     D
            1   x X x   ^ ^   @         =
                1   2   x X     P     ^   E       E
                        2       a ^ ^ R   q ^ ^ ^ q ^ ^ ^
                                i x x     C X R x C X R x
                                r 1 2     l     1 l     2
```

```
A(X) A(R|Equ(X,R))
A(x1|@(x1,X)) A(x2|@(x2,X))
D(@(Pair(x1,x2),R),=(EqCl(X,R,x1),EqCl(X,R,x2)))
```

1078(T) Not-equivalence is disjoint equivalence classes

```
A - - - + - - - - + - - - + - - - - - - - - - + - - - + - -
^ T A - - + - - - - - - - - - - - - - - + - - - + - - - + -
X   ^ E     A - + - - - - - - - - - + - - - - - - + - - - -
    R q ^ ^ ^ @     A - + - - - - - - - + - - - - - - - - - +
      u X R x   ^ ^ ^ @     D
            1   x X x   ^ ^   N           D
                1   2   x X     @         i E       E
                        2         P     ^ s q ^ ^ ^ q ^ ^ ^
                                  a ^ ^ R   C X R x C X R x
                                  i x x     l     1 l     2
                                  r 1 2
```

```
A(X) A(R|Equ(X,R)) A(x1|@(x1,X)) A(x2|@(x2,X))
D(N(@(Pair(x1,x2),R)),Dis(EqCl(X,R,x1),EqCl(X,R,x2)))
```

1080(T) Equivalence classes are equal or disjoint

```
A - - - + - - - - + - - - + - - - + - - - + - - - - + - - - + - -
^ T A - - + - - - - - - - - - - - - - + - - - + - - - - + - - - + -
X   ^ E     A - + - - - - - - - - - - - + - - - - - - - - - + - - - -
    R q ^ ^ ^ @     A - + - - - - - - - - - - - - + - - - - - - - - - +
      u X R x   ^ ^ ^ @     V
            1   x X x   ^ ^   =               D
                1   2   x X     E       E       i E       E
                        2       q ^ ^ ^ q ^ ^ ^ s q ^ ^ ^ q ^ ^ ^
                                C X R x C X R x   C X R x C X R x
                                l     1 l     2   l     1 l     2
```

```
A(X) A(R|Equ(X,R)) A(x1|@(x1,X)) A(x2|@(x2,X))
V(=(EqCl(X,R,x1),EqCl(X,R,x2)),Dis(EqCl(X,R,x1),EqCl(X,R,x2)))
```

1082(T) Equivalence class of x is included in X

```
A - - - + - - - - + - - + - - +
^ T A - - + - - - - - - - - + - -
X   ^ E     A - + - - - - - + -
    R q ^ ^ ^ @     P
      u X R x   ^ ^ a E       ^
                x X r q ^ ^ ^ X
                    t C X R x
                      l
```

```
A(X) A(R|Equ(X,R)) A(x|@(x,X)) Part(EqCl(X,R,x),X)
```

1083(R) CmpAb definition (comparable elements)

```
A - - - + - - - - + - - - + - - + - - - - - - - - - - - - - - -
^ T A - - + - - - - - - - - - - - - + - - - - - - - - + - - - - +
X   ^ O     A - + - - - - - - - - - - + - - - - + - - - - - - + -
    R r ^ ^ ^ @     A - + - - - - - - - + - - - - + - - - - + - -
      d X R x   ^ ^ ^ @     D
                x X y   ^ ^   C         V
                        y X   m ^ ^ ^ ^   @         @
                              p X R x y     P     ^   P     ^
                              A             a ^ ^ R   a ^ ^ R
                              b             i x y     i y x
                                            r         r
```

```
A(X) A(R|Ord(X,R)) A(x|@(x,X)) A(y|@(y,X))
D(CmpAb(X,R,x,y),V(@(Pair(x,y),R),@(Pair(y,x),R)))
```

*Let **X**, **R** be classes, with **X** being ordered by **R**, and let **x**, **y** be elements of **X**. **x** and **y** are comparable if and only if Pair(**x**, **y**) belongs to R, or Pair(**y**, **x**) belongs to R.*

1084(R) FuOrd definition (X is fully ordered by R)

```
A - - - - - + - - - + - - - - - + - - - + - + - - -
^ T A - - - - + - - - + - - - - - - - - - - - + - -
X   ^ T D
    R     F     X
          u ^ ^   O     A - + - - - - - - - - - + -
          O X R   r ^ ^ ^ @     A - + - - - - - - +
          r       d X R x   ^ ^ ^ @     C
          d                 x X y   ^ ^ m ^ ^ ^ ^
                                    y X p X R x y
                                        A
                                        b
```

```
A(X) A(R) D(FuOrd(X,R),X(Ord(X,R),A(x|@(x,X)) A(y|@(y,X)) CmpAb(X,R,x,y)))
```

*Let **X**, **R** be classes. **X** is fully ordered by **R** if and only if **X** is ordered by **R** and all the elements of **X** are comparable.*

1085(R) WeOrd definition (X is well ordered by R)

```
A - - - - - + - - - + - - - - - + - - - - - - - - - - - - - - - - -
^ T A - - - - + - - - + - - - - - - - - - - - - - - - - - - - - - +
X   ^ T D
    R     W     X
          e ^ ^   F     A - - + - - - + - - - - + - - - + - - - - -
          O X R   u ^ ^ ^ X               E - + - - - - - - - + - -
          r       O X R Y   P     N       ^ @     A - + - - - - + -
          d       r         a ^ ^   =     y   ^ ^ ^ @     @
                  d         r Y X     ^ 0     y Y z   ^ ^   P     ^
                            t         Y               z Y   a ^ ^ R
                                                            i y z
                                                            r
```

```
A(X) A(R) D(WeOrd(X,R),X(FuOrd(X,R),A(Y|X(Part(Y,X),N(=(Y,0)))) E(y|@(y,Y))
A(z|@(z,Y)) @(Pair(y,z),R)))
```

*Let **X**, **R** be classes. **X** is well ordered by **R** if and only if **X** is fully ordered by **R**, and for any not empty **Y** included in **X**, there exists **y** belonging to **Y** such that for any **z** belonging to **Y**, Pair(**y**, **z**) belongs to **R**.*

1086(T) Existence for Min

```
A - - - + - - - - - + - - - - - - - - - - - - - - - - - - -
^ T A - - + - - - - - - - - - - - - - - - - - - - - - - - +
X   ^ W     A - - + - - - + - - - - - - + - - - + - - - - -
    R e ^ ^ ^ X               E - - - + - - - - - - - + - -
      O X R Y   P     N       ^ T X
      r         a ^ ^   =     y     @     A - + - - - - + -
      d         r Y X     ^ 0         ^ ^ ^ @     @
                t         Y           y Y z   ^ ^   P     ^
                                              z Y   a ^ ^ R
                                                    i y z
                                                    r
```

```
A(X) A(R|WeOrd(X,R)) A(Y|X(Part(Y,X),N(=(Y,0)))) E(y) X(@(y,Y),A(z|@(z,Y))
@(Pair(y,z),R))
```

1087(T) Uniqueness for Min

```
A - - - + - - - - - + - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
^ T A - - + - - - - - - - - - - - - - - - - - - - - - - + - - - - - - - - - - - - - + - - -
X   ^ W     A - - + - - - + - - - - - + - - - + - - - - - - - - - + - - - + - - - - - - - -
    R e ^ ^ ^ X               A - - + - - - - - - - + - - - - - - - - - - - - - - - - - + -
      O X R Y   P     N       ^ X                       A - - + - - - - - - - + - - - - +
      r         a ^ ^   =     y   @     A - + - - - - + - ^ X                         =
      d         r Y X     ^ 0 1     ^ ^ ^ @     @         y   @     A - + - - - - + -   ^ ^
                t         Y         y Y z   ^ ^   P     ^ 2     ^ ^ ^ @     @           y y
                                    1   1   z Y   a ^ ^ R       y Y z   ^ ^   P     ^   1 2
                                            1     i y z         2   2   z Y   a ^ ^ R
                                                  r 1 1                 2     i y z
                                                                              r 2 2
```

A(X) A(R|WeOrd(X,R)) A(Y|X(Part(Y,X),N(=(Y,0))))
A(y1|X(@(y1,Y),A(z1|@(z1,Y)) @(Pair(y1,z1),R)))
A(y2|X(@(y2,Y),A(z2|@(z2,Y)) @(Pair(y2,z2),R))) =(y1,y2)

1088(F) Min definition

```
A - - - + - - - - - + - - - - - - - + - - - - - - - - - - + - - - -
^ T A - - + - - - - - - - - - - - - - + - - - - - - - - - - + - - +
X   ^ W     A - - + - - - + - - - - - - + + - - - + - - - - - + - -
    R e ^ ^ ^ X               X
      O X R Y   P     N         @           A - + - - - - - - - + -
      r         a ^ ^   =         M       ^ ^ @     @
      d         r Y X     ^ 0     i ^ ^ ^ Y y   ^ ^   P           ^
                t         Y       n X R Y       y Y   a M       ^ R
                                                      i i ^ ^ ^ y
                                                      r n X R Y
```

A(X) A(R|WeOrd(X,R)) A(Y|X(Part(Y,X),N(=(Y,0)))) X(@(Min(X,R,Y),Y),A(y|@(y,Y))
@(Pair(Min(X,R,Y),y),R))

Natural numbers

We all have an intuitive understanding of natural numbers; we know they are 0, 1, 2, 3, and so on, and we can do calculations with them. However, from a mathematical point of view, this is not enough. Through his axioms, Peano succeeded in creating a standard formalization of arithmetic.

In the following, we will define the set of natural numbers as a class, as well as each natural number separately. Of course, we will prove that these classes exist and that they are unique. For this, we will need another axiom, the axiom of infinity. In this way, Peano's axioms become simple theorems. It is interesting to see the enigmatic principle of mathematical induction as a simple theorem!

We will first define the notion of the successor of a set (Suc) and then the notion of the successor set (SucSet). The axiom of infinity guarantees that there exists a successor set. By intersecting the successor sets, we will obtain the natural numbers.

After we define the notion of transitive set (TraSet), we will show that natural numbers are transitive sets. Finally, we will show that our set of natural numbers satisfies the expected properties, including Peano's axioms.

The empty set is the natural number 0. Using the notion of the successor of a set, it is "natural" to consider that 1 is the successor of 0, 2 is the successor of 1, and so on.

A set is included in its successor, but it also belongs to its successor. This last property allows us to define the relation "≤" for the natural numbers.

To prove the recursion theorem, we will first define the notions of recursive graph (RecGra) and recursive function (RecFun). By intersecting the recursive graphs, we will obtain a graph (RecInt). Finally, we will prove a theorem that shows that RecInt is a recursive function.

The recursion theorem is a direct consequence of this theorem. This shows the importance of the RecInt function.

1201(F) Suc definition (the successor of a set)

```
A - + - - + - + - +
^ S   =
x   ^   S   U
    x   u ^   ^ S
        c x   x n ^
                g x
```

```
A(x|S(x)) =(Suc(x),U(x,Sng(x)))
```

*The successor of **x** is the union of **x** and Sng(**x**).*

1202(R) SucSet definition (successor set)

```
A - - - + - - + - - - + - - - + - - - +
^ T D
X     S   X
      u ^   S   X
      c X     ^   @     A - + - - - + -
      S       X     0 ^ ^ @     @
      e               X x   ^ ^   S   ^
      t                     x X   u ^ X
                                  c x
```

```
A(X) D(SucSet(X),X(S(X),X(@(0,X),A(x|@(x,X)) @(Suc(x),X))))
```

*Let **X** be a class. **X** is a successor set if and only if **X** is a set, 0 is an element of **X**, and for any **x** belonging to **X**, the successor of **x** belongs to **X**.*

1203(A) Axiom of infinity

```
 E - - +
 ^ T S
 X   u ^
     c X
     S
     e
     t
```

```
E(X) SucSet(X)
```

There exists an ***X*** *such that* ***X*** *is a successor set.*

1204(F) Nat definition (the set of natural numbers)

```
 =
   N G
   a I K - +
   t   ^ S
       X u ^
         c X
         S
         e
         t
```

```
=(Nat,GI({X|SucSet(X)}))
```

The set of natural numbers is the general intersection of the class of all successor sets.

1205(T) Nat explanation

```
 A - + - - + - - - - - + -
 ^ S   D
 n   ^   @     A - + - - +
     n     ^ N ^ S   @
           n a X u ^   ^ ^
             t   c X   n X
                 S
                 e
                 t
```

```
A(n|S(n)) D(@(n,Nat),A(X|SucSet(X)) @(n,X))
```

Let ***n*** *be a set.* ***n*** *is a natural number if and only if, for any successor set* ***X***, ***n*** *belongs to* ***X***.

1208(T) Suc explanation

```
 A - + - - - - - - - + - - - + - - +
 ^ S   A - + - - + - - - - + - - + -
 x   ^ ^ S   D
     x u   ^   @       V
           u     ^ S     @     =
                 u u ^     ^ ^   ^ ^
                   c x     u x   u x
```

```
A(x|S(x)) A(u|S(u)) D(@(u,Suc(x)),V(@(u,x),=(u,x)))
```

Let ***x***, ***u*** *be sets.* ***u*** *belongs to Suc(****x****) if and only if* ***u*** *belongs to* ***x*** *or* ***u*** *is equal to* ***x***.

1209(T) A set belongs to its successor

```
 A - + - + - +
 ^ S   @
 x   ^   ^ S
     x   x u ^
           c x
```

```
A(x|S(x)) @(x,Suc(x))
```

1210(T) Suc is not empty

```
 A - + - - - + -
 ^ S   N
 x   ^   =
     x     S   0
           u ^
           c x
```

```
A(x|S(x)) N(=(Suc(x),0))
```

*For any set **x**, Suc(**x**) is not equal to 0.*

1212(T) Nat corollary

```
 A - + - - +
 ^ S   P
 X u ^ a N ^
   c X r a X
   S   t t
   e
   t
```

```
A(X|SucSet(X)) Part(Nat,X)
```

*For any successor set **X**, Nat is included in **X**.*

1213(T) Nat is a set

```
 S
   N
   a
   t
```

```
S(Nat)
```

1214(T) Mathematical induction (Peano 4)

```
 A - - - - + - - - - + - - - + - - - + - + -
 ^ T C
 X     X                               =
         P     X                         ^ N
         a ^ N   @     A - + - - - + -   X a
         r X a     0 ^ ^ @     @           t
         t   t       X n   ^ ^   S   ^
                           n X   u ^ X
                                 c n
```

```
A(X) C(X(Part(X,Nat),X(@(0,X),A(n|@(n,X)) @(Suc(n),X))),=(X,Nat))
```

*Let **X** be a class. If **X** is included in Nat, 0 belongs to **X**, and for any **n** belonging to **X**, Suc(**n**) belongs to **X**, then **X** is equal to Nat.*

1215(T) 0 is a natural number (Peano 1)

```
 @
   0 N
     a
     t
```

```
@(0,Nat)
```

1216(R) TraSet definition (transitive set)

```
 A - - - + - - + - - - + - - +
 ^ T D
 X     T   X
       r ^   S   A - + - - + -
       a X     ^ ^ @     P
       S       X x   ^ ^ a ^ ^
       e             x X r x X
       t                 t
```

```
A(X) D(TraSet(X),X(S(X),A(x|@(x,X)) Part(x,X)))
```

*Let **X** be a class. **X** is a transitive set if and only if **X** is a set, and for any **x** belonging to **X**, **x** is included in **X**.*

1217(T) 0 is a transitive set

```
 T
 r 0
 a
 S
 e
 t
```

```
TraSet(0)
```

1218(T) Suc is a set

```
 A - + - - +
 ^ S   S
 x   ^   S
     x   u ^
         c x
```

```
A(x|S(x)) S(Suc(x))
```

*For any set **x**, Suc(**x**) is a set.*

1219(T) The successor of a natural number is a natural number (Peano 2)

```
A - + - - + - - - + -
^ S   C
n   ^   @     @
    n     ^ N   S   N
          n a   u ^ a
            t   c n t
```

`A(n|S(n)) C(@(n,Nat),@(Suc(n),Nat))`

1220(T) The successor of a natural number is a natural number

```
A - + - - - + -
^ @     @
n   ^ N   S   N
    n a   u ^ a
      t   c n t
```

`A(n|@(n,Nat)) @(Suc(n),Nat)`

1221(T) Suc corollary

```
A - + - - - + - - - +
^ S   A - + - - + - -
x   ^ ^ @     @
    x u   ^ ^   ^ S
          u x   u u ^
                  c x
```

`A(x|S(x)) A(u|@(u,x)) @(u,Suc(x))`

1222(T) A set is included in its successor

```
A - + - + - +
^ S   P
x   ^ a ^ S
    x r x u ^
      t   c x
```

`A(x|S(x)) Part(x,Suc(x))`

1223(T) TraSet corollary

```
A - + - - - - + - - - +
^ T   A - + - - - + - -
x r ^ ^ @       P
  a x y   ^ S   a ^ S
  S       y u ^ r y u ^
  e         c x t   c x
  t
```

`A(x|TraSet(x)) A(y|@(y,Suc(x))) Part(y,Suc(x))`

1224(T) The successor of a transitive set is a transitive set

```
 A - + - - +
 ^ T   T
 x r ^ r S
   a x a u ^
   S   S c x
   e   e
   t   t

A(x|TraSet(x)) TraSet(Suc(x))
```

1228(T) TraSet corollary

```
 =
   K - - + - - + N
   ^ X           a
   u   @     T   t
         ^ N r ^
         u a a u
           t S
             e
             t

=({u|X(@(u,Nat),TraSet(u))},Nat)
```

1229(T) Natural numbers are transitive sets

```
 A - + - - +
 ^ @     T
 n   ^ N r ^
     n a a n
       t S
         e
         t

A(n|@(n,Nat)) TraSet(n)
```

1231(T) Equality of successors of transitive sets implies their equality

```
 A - + - - - - - - + - - - + -
 ^ T   A - + - - - - - + - - +
 X r ^ ^ T   C
   a X Y r ^   =         =
   S     a Y     S   S     ^ ^
   e     S       u ^ u ^   X Y
   t     e       c X c Y
         t

A(X|TraSet(X)) A(Y|TraSet(Y)) C(=(Suc(X),Suc(Y)),=(X,Y))
```

1232(T) Equality of successors of natural numbers implies their equality (Peano 5)

```
 A - + - - - - - - - - - + - - - + -
 ^ @     A - + - - - - - - - + - - - +
 m   ^ N ^ @     C
     m a n   ^ N   =         =
       t     n a     S   S     ^ ^
               t     u ^ u ^   m n
                     c m c n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) C(=(Suc(m),Suc(n)),=(m,n))
```

1233(T) The successor of a natural number is not null (Peano 3)

```
 A - + - - - - - + -
 ^ @     N
 n   ^ N   =
     n a     S   0
       t     u ^
             c n
```

```
A(n|@(n,Nat)) N(=(Suc(n),0))
```

Finite recursion

With the help of mathematical induction (Peano 4), we can demonstrate various formulas for natural numbers. Mathematical induction is also closely related to the recursive definition of a function. We are talking about the possibility of defining a function having as its domain the set of natural numbers, defining the value of the function for 0, and defining the value of the function for n + 1 using the value of the function for n, which we assume to be known. But let's try to describe more precisely the recursive definition of a function.

*Let **A** be a set, **a** an element of **A**, **F** a function from **A** to **A**, and **P** a class.*

*We say that **P** is a recursive function (we write RecFun(**A**, **a**, **F**, **P**) if and only if*

- ***P** is a function from Nat to **A**.*
- *Val(**P**, 0) = **a**.*
- *For any natural number **n**, Val(**P**, Suc(**n**)) = Val(**F**, Val(**P**, **n**)).*

We want to prove that under the given conditions for A, a, F, there exists a unique function P. For this, we will first define a recursive graph G (we write RecGra(A, a, F, G)), similar to a recursive function, but replacing the function condition with the graph condition. In the next step, we will intersect all the recursive graphs, obtaining a recursive intersection:

RecInt(**A**, **a**, **F**) = GI {**G** | RecGra(**A**, **a**, **F**, **G**)}, where GI is the general intersection. For this recursive intersection, we will first show that it is a recursive graph. Then we will show that it is indeed a recursive function, and finally we will show that it is unique. In other words, the recursive intersection is exactly the function we are looking for.

1239(R) RecGra definition (recursive graph)

```
A - + - - - + - - + + - - - - - + - - - - - - - - + - - - - - - - - - - - - - - - - - - - - - - - - - - -
^ S   A - + - - - - - - - - - - - + - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - - - - - -
A   ^ ^ @     A - - - + - - - - - - + - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - + - -
    A a   ^ ^ ^ F       A - - - - - - + - - + - - - - - - - - + - - - - - - - - - - - - + - - - - - - - +
          a A F u ^ ^ ^ ^ T D
                n A A F G     R         X
                              e ^ ^ ^ ^   P         X
                              c A a F G   a ^ C       @         A - + - - - - - - - + - - - - - + - - - -
                              G           r G a N ^     P     ^ ^ @     A - - + - - - + - - - - - - - + -
                              r           t   r a A     a 0 ^ G n   ^ N ^ X               @
                              a               t t       i   a       n a x   S   @           P           ^
                                                        r             t       ^   P     ^   a S   V     G
                                                                              x   a ^ ^ G   i u ^ a ^ ^
                                                                                  i n x     r c n l F x
                                                                                  r
```

A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) A(G)
D(RecGra(A,a,F,G),X(Part(G,Cart(Nat,A)),X(@(Pair(0,a),G),A(n|@(n,Nat))
A(x|X(S(x),@(Pair(n,x),G))) @(Pair(Suc(n),Val(F,x)),G))))

*Let **A** be a set, **a** an element of **A**, **F** a class with Fun(**A**, **A**, **F**), and **G** a class. **G** is a recursive graph (RecGra(**A**, **a**, **F**, **G**)) if and only if **G** is included in the Cartesian product of Nat and **A**, Pair(0, **a**) belongs to **G**, and for any natural number **n** and for any set **x** with Pair(**n**, **x**) belonging to **G**, Pair(Suc(**n**), Val(**F**, **x**)) belongs to **G**.*

1240(T) A recursive graph is a graph

```
A - + - - - + - - + + - - - + - - - - -
^ S   A - + - - - - - - - - - + - - - -
A   ^ ^ @     A - - - + - - - - + - - -
    A a   ^ ^ ^ F       A - - - - + - +
          a A F u ^ ^ ^ ^ R         G
                n A A F G e ^ ^ ^ ^ r ^
                          c A a F G a G
                          G         p
                          r         h
                          a
```

A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) A(G|RecGra(A,a,F,G)) Graph(G)

1241(T) A recursive graph is a set

```
A - + - - - + - - + + - - - + - - - - -
^ S   A - + - - - - - - - - - + - - - -
A   ^ ^ @     A - - - + - - - - + - - -
    A a   ^ ^ ^ F       A - - - - + - +
          a A F u ^ ^ ^ ^ R         S
                n A A F G e ^ ^ ^ ^   ^
                          c A a F G   G
                          G
                          r
                          a
```

A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) A(G|RecGra(A,a,F,G)) S(G)

1244(T) Cart of Nat and a set is RecGra

```
A - + - - - + - - + + - - + - - - - +
^ S   A - + - - - - - - - - + - - - -
A   ^ ^ @     A - - - + - - - + - - -
    A a   ^ ^ ^ F       R
          a A F u ^ ^ ^ e ^ ^ ^ C
                n A A F c A a F a N ^
                        G       r a A
                        r       t t
                        a
```

```
A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) RecGra(A,a,F,Cart(Nat,A))
```

1245(F) RecInt definition

```
A - + - - - + - - + + - - - + - - - - - + - - -
^ S   A - + - - - - - - - - - + - - - - - + - -
A   ^ ^ @     A - - - + - - - - + - - - - - + -
    A a   ^ ^ ^ F       =
          a A F u ^ ^ ^   R       G
                n A A F   e ^ ^ ^ I K - - - - +
                          c A a F   ^ R
                          I         G e ^ ^ ^ ^
                          n           c A a F G
                          t           G
                                      r
                                      a
```

```
A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) =(RecInt(A,a,F),GI({G|RecGra(A,a,F,G)}))
```

*Let **A** be a set, **a** in **A** and **F** with Fun(**A**, **A**, **F**). Then RecInt(**A**, **a**, **F**) is the general intersection of the class of all **G** with RecGra(**A**, **a**, **F**, **G**).*

1250(T) RecInt is a set

```
A - + - - - + - - + + - - - + - -
^ S   A - + - - - - - - - - - + -
A   ^ ^ @     A - - - + - - - - +
    A a   ^ ^ ^ F       S
          a A F u ^ ^ ^   R
                n A A F   e ^ ^ ^
                          c A a F
                          I
                          n
                          t
```

```
A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) S(RecInt(A,a,F))
```

1251(T) RecInt is Graph

```
A - + - - - + - - + + - - - + - -
^ S   A - + - - - - - - - - - + -
A   ^ ^ @     A - - - + - - - - +
    A a   ^ ^ ^ F       G
          a A F u ^ ^ ^ r R
                n A A F a e ^ ^ ^
                        p c A a F
                        h I
                          n
                          t
```

```
A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) Graph(RecInt(A,a,F))
```

1252(T) Pair of 0 and a belongs to RecInt

```
A - + - - - + - - + + - - - - - - + - -
^ S   A - + - - - - - - - - - + - - + -
A   ^ ^ @     A - - - + - - - - - - - +
    A a   ^ ^ ^ F       @
          a A F u ^ ^ ^   P     R
                n A A F   a 0 ^ e ^ ^ ^
                          i   a c A a F
                          r     I
                                n
                                t
```

```
A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) @(Pair(0,a),RecInt(A,a,F))
```

*If **A** is a set, **a** an element of **A**, and **F** a class with Fun of **A**, **A**, **F**, then Pair(0, **a**) belongs to RecInt of **A**, **a**, **F**.*

1257(T) RecInt is RecGra

```
A - + - - - + - - + + - - + - - - + - -
^ S   A - + - - - - - - - - + - - - + -
A   ^ ^ @     A - - - + - - - + - - - +
    A a   ^ ^ ^ F       R
          a A F u ^ ^ ^ e ^ ^ ^ R
                n A A F c A a F e ^ ^ ^
                        G       c A a F
                        r       I
                        a       n
                                t
```

```
A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) RecGra(A,a,F,RecInt(A,a,F))
```

1259(T) RecGra existence

```
A - + - - - + - - + + - - - - + - - -
^ S   A - + - - - - - - - - - - - + - -
A   ^ ^ @     A - - - + - - - - - - + -
    A a   ^ ^ ^ F       E - - - - - - +
          a A F u ^ ^ ^ ^ T R
                n A A F G   e ^ ^ ^ ^
                            c A a F G
                            G
                            r
                            a
```

`A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) E(G) RecGra(A,a,F,G)`

*For any set **A**, and for any **a** in A, and for any **F** with Fun of **A**, **A**, **F**, there exists **G** such as RecGra of **A**, **a**, **F**, **G**.*

1264(T) Dom of RecInt is Nat

```
A - + - - - + - - + + - - - - + - - -
^ S   A - + - - - - - - - - - - - + - -
A   ^ ^ @     A - - - + - - - - - - + -
    A a   ^ ^ ^ F       =
          a A F u ^ ^ ^   D         N
                n A A F   o R       a
                          m e ^ ^ ^ t
                            c A a F
                            I
                            n
                            t
```

`A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) =(Dom(RecInt(A,a,F)),Nat)`

1280(T) RecInt is included in all RecGra

```
A - + - - - + - - + + - - - + - - - - - - + - - -
^ S   A - + - - - - - - - - - - - + - - - - - - + - -
A   ^ ^ @     A - - - + - - - - - + - - - - - - + -
    A a   ^ ^ ^ F       A - - - - + - - - - - - +
          a A F u ^ ^ ^ ^ R         P
                n A A F G e ^ ^ ^ ^ a R         ^
                          c A a F G r e ^ ^ ^ G
                          G         t c A a F
                          r           I
                          a           n
                                      t
```

`A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) A(G|RecGra(A,a,F,G)) Part(RecInt(A,a,F),G)`

1299(T) RecInt is Fun

```
A - + - - - + - - + + - - - + - + - -
^ S   A - + - - - - - - - - - - - + -
A   ^ ^ @     A - - - + - - - - - - - +
    A a   ^ ^ ^ F       F
          a A F u ^ ^ ^ u N ^ R
                n A A F n a A e ^ ^ ^
                          t   c A a F
                              I
                              n
                              t
```

```
A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) Fun(Nat,A,RecInt(A,a,F))
```

1301(R) RecFun definition (recursive function)

```
A - + - - - + - - + + - - - - - + - - - - - - + - - + - - -
^ S   A - + - - - - - - - - - - - + - - - - - - - - - - + - -
A   ^ ^ @     A - - - + - - - - - - - + - - - - - - - - - - + -
    A a   ^ ^ ^ F       A - - - - - - - + - - - - + - - - - - +
          a A F u ^ ^ ^ ^ T D
                n A A F G     R         X
                              e ^ ^ ^ ^   F       R
                              c A a F G   u N ^ ^ e ^ ^ ^ ^
                              F           n a A G c A a F G
                              u             t     G
                              n                   r
                                                  a
```

```
A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) A(G)
D(RecFun(A,a,F,G),X(Fun(Nat,A,G),RecGra(A,a,F,G)))
```

*Let **A** be a set, **a** an element of **A**, and **F** with Fun(**A**, **A**, **F**), and **G** a class. RecFun of **A**, **a**, **F**, **G** if and only if Fun of Nat, **A**, **G** and RecGra of **A**, **a**, **F**, **G**.*

1302(T) RecInt is RecFun

```
A - + - - - + - - + + - - + - - - + - -
^ S   A - + - - - - - - - - - + - - - + -
A   ^ ^ @     A - - - + - - - + - - - - +
    A a   ^ ^ ^ F       R
          a A F u ^ ^ ^ e ^ ^ ^ R
                n A A F c A a F e ^ ^ ^
                        F       c A a F
                        u       I
                        n       n
                                t
```

```
A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) RecFun(A,a,F,RecInt(A,a,F))
```

1303(T) Recursion theorem (RecFun existence)

```
A - + - - - + - - + + - - - - + - - -
^ S   A - + - - - - - - - - - - + - -
A   ^ ^ @     A - - - + - - - - - + -
    A a   ^ ^ ^ F       E - - - - - +
          a A F u ^ ^ ^ ^ T R
                n A A F P   e ^ ^ ^ ^
                            c A a F P
                            F
                            u
                            n
```

A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) E(P) RecFun(A,a,F,P)

*For any set **A**, and for any **a** in **A**, and for any **F** with Fun(**A**, **A**, **F**), there exists **P** such as RecFun(**A**, **a**, **F**, **P**).*

1313(T) Two RecFun are equal (RecInt is the only one RecFun)

```
A - + - - - + - - + + - - - + - - - - - + - - - - - -
^ S   A - + - - - - - - - - - + - - - - - + - - - - -
A   ^ ^ @     A - - - + - - - - + - - - - - + - - - -
    A a   ^ ^ ^ F       A - - - - + - - - - - - - + -
          a A F u ^ ^ ^ ^ R         A - - - - + - - +
                n A A F P e ^ ^ ^ ^ ^ R         =
                          c A a F P Q e ^ ^ ^ ^   ^ ^
                          F           c A a F Q   P Q
                          u           F
                          n           u
                                      n
```

A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F))
A(P|RecFun(A,a,F,P)) A(Q|RecFun(A,a,F,Q)) =(P,Q)

1315(T) RecFun corollary

```
A - + - - - + - - + + - - - - - + - - - - - - + - - - - - - - - - - - - - - - - - - - - -
^ S   A - + - - - - - - - - - - - + - - - - - - - - - - - - + - - - - - - - - - - - - - -
A   ^ ^ @     A - - - + - - - - - - + - - - - - - - - - - - - - - - - - - - - - - + - - -
    A a   ^ ^ ^ F       A - - - - - - + - - - - + - - - + - - - - - - - - + - - - - - + -
          a A F u ^ ^ ^ ^ T D
                n A A F P     R         X
                              e ^ ^ ^ ^   F       X
                              c A a F P   u N ^ ^   =         A - + - - - - - + - - - - +
                              F           n a A P     V     ^ ^ @     =
                              u             t         a ^ 0 a n   ^ N   V       V
                              n                       l P         n a   a ^ S   a ^ V
                                                                    t   l P u ^ l F a ^ ^
                                                                            c n     l P n
```

A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) A(P)
D(RecFun(A,a,F,P),X(Fun(Nat,A,P),X(=(Val(P,0),a),A(n|@(n,Nat))
=(Val(P,Suc(n)),Val(F,Val(P,n))))))

*Let **A** be a set, **a** an element of **A**, **F** a function from **A** to **A**, and **P** a class. RecFun(**A**, **a**, **F**, **P**) if and only if **P** is a function from Nat to **A**, Val(**P**, 0) = **a**, and for any natural number **n**, Val(P, Suc(**n**)) = Val(**F**, Val(**P**, **n**)).*

1326(T) A natural number is 0, or it is the successor of a natural number

```
A - + - - - + - - - - - - + - -
^ @     V
n   ^ N   =     E - + - - - - +
    n a     ^ 0 ^ @     =
      t     n   k   ^ N   ^ S
                    k a   n u ^
                      t     c k
```

A(n|@(n,Nat)) V(=(n,0),E(k|@(k,Nat)) =(n,Suc(k)))

1327(T) RecInt corollary

```
A - + - - - + - - + + - - - - - + - - - - - - - - - - - + - - - - - - - - + - - -
^ S   A - + - - - - - - - - - - - + - - + - - - - - - - - + - - - - - - - - + - -
A   ^ ^ @     A - - - + - - - - - - + - - - - - - - - - - - + - - - + - - - - + -
    A a   ^ ^ ^ F       X
          a A F u ^ ^ ^   =               A - + - - - - - - - - + - - - - - - - +
                n A A F     V           ^ ^ @     =
                            a R     0 a n   ^ N   V             V
                            l e ^ ^ ^         n a   a R       S   a ^ V
                              c A a F           t   l e ^ ^ ^ u ^ l F a R       ^
                              I                       c A a F c n     l e ^ ^ ^ n
                              n                       I                 c A a F
                              t                       n                 I
                                                      t                 n
                                                                        t
```

A(A|S(A)) A(a|@(a,A)) A(F|Fun(A,A,F)) X(=(Val(RecInt(A,a,F),0),a),A(n|@(n,Nat))
=(Val(RecInt(A,a,F),Suc(n)),Val(F,Val(RecInt(A,a,F),n))))

*Let **A** be a set, **a** an element of **A**, and **F** a function from **A** to **A**.*
*Val(RecInt(**A**, **a**, **F**), 0) = **a**, and for any natural number **n**,*
*Val(RecInt(**A**, **a**, **F**), Suc(**n**)) = Val(**F**, Val(RecInt(**A**, **a**, **F**), **n**)).*

1337(T) F Inj implies RecInt is Inj

```
A - + - - - + - - - + + - - - - - - - - + - + - -
^ S   A - + - - - - - - - - - + - - - - - - - + -
A   ^ ^ @     A - - - - + - - - - + - - - - - - +
    A a   ^ ^ ^ X                   I
          a A F   I       N         n N ^ R
                  n ^ ^ ^   @       j a A e ^ ^ ^
                  j A A F     ^ R     t   c A a F
                              a a ^       I
                                n F       n
                                          t
```

A(A|S(A)) A(a|@(a,A)) A(F|X(Inj(A,A,F),N(@(a,Ran(F))))) Inj(Nat,A,RecInt(A,a,F))

Arithmetic of natural numbers

Using the notion of the successor of a set, we will define a graph (SucFun) so that Nat, Nat, SucFun will be a function. This function will allow us to define the addition of the natural numbers. Similarly, we will use another graph (MplFun) to define the multiplication of the natural numbers. Many of the following proofs are based on mathematical induction.

Given that any set belongs to its successor, we will use the belonging and the equality to define the relation "less than or equal to" for natural numbers. Later, we will use this (primitive) relation to define a graph (LER), which we will prove to be a relation according to definition 518. It will also be an order relation.

In the following, we will define the arithmetic operations for natural numbers. In our system, each definition receives a unique name, which is used to generate sentences. Since the addition of natural numbers is not the same as the addition of integer numbers, we will have to use different names for the two definitions; we cannot use "+" in both cases. For this reason, we will call the addition of natural numbers "N+" and the addition of integer numbers "Z+". We will do the same for other operations and classes.

1338(F) SucFun definition

```
 =
   S K - - - - - + - - - -
   u ^ E - + - - - - + - +
   c u ^ @     =
   F   n   ^ N   ^ P
   u       n a   u a ^ S
   n         t     i n u ^
                   r   c n
```

=(SucFun,{u|E(n|@(n,Nat)) =(u,Pair(n,Suc(n)))})

*SucFun = {**u** | there exists a natural number **n**, such that **u** = Pair(**n**, Suc(**n**))}*

1344(T) SucFun is Graph

```
 G
 r S
 a u
 p c
 h F
   u
   n
```

Graph(SucFun)

1347(T) Dom of SucFun is Nat

```
 =
   D   N
   o S a
   m u t
     c
     F
     u
     n

=(Dom(SucFun),Nat)
```

1349(T) SucFun is Func

```
 F
 u S
 n u
 c c
   F
   u
   n

Func(SucFun)
```

1352(T) Nat, Nat, SucFun is Fun

```
 F
 u N N S
 n a a u
   t t c
       F
       u
       n

Fun(Nat,Nat,SucFun)
```

1353(T) Pair of n and Suc of n belong to SucFun

```
 A - + - - - + - + -
 ^ @     @
 n   ^ N   P       S
     n a   a ^ S   u
       t   i n u ^ c
           r   c n F
                   u
                   n

A(n|@(n,Nat)) @(Pair(n,Suc(n)),SucFun)
```

1354(T) Val of SucFun is Suc

```
A - + - - - - + - +
^ @   =
n   ^ N   V     S
    n a   a S ^ u ^
      t   l u n c n
            c
            F
            u
            n
```

```
A(n|@(n,Nat)) =(Val(SucFun,n),Suc(n))
```

1355(F) RecNat definition

```
A - + - - - + - - + -
^ @   =
n   ^ N   R   R
    n a   e ^ e N ^ S
      t   c n c a n u
          N   I t   c
          a   n     F
          t   t     u
                    n
```

```
A(n|@(n,Nat)) =(RecNat(n),RecInt(Nat,n,SucFun))
```

1356(T) Fun of Nat, Nat, RecNat

```
A - + - - - - - +
^ @     F
m   ^ N u N N R
    m a n a a e ^
      t   t t c m
              N
              a
              t
```

```
A(m|@(m,Nat)) Fun(Nat,Nat,RecNat(m))
```

1357(T) RecNat corollary

```
A - + - - - - - + - + - - - - - - - + - - - - - + -
^ @     X
m   ^ N   =           A - + - - - - - - + - - - - +
    m a     V       ^ ^ @     =
      t     a R   0 m n   ^ N   V         S
            l e ^         n a   a R   S   u V
              c m           t   l e ^ u ^ c a R   ^
              N                   c m c n   l e ^ n
              a                   N             c m
              t                   a             N
                                  t             a
                                                t
```

```
A(m|@(m,Nat)) X(=(Val(RecNat(m),0),m),A(n|@(n,Nat))
=(Val(RecNat(m),Suc(n)),Suc(Val(RecNat(m),n))))
```

1358(F) N+ definition (addition of natural numbers)

```
A - + - - - - - - - + - - - + -
^ @     A - + - - - - + - - - +
m   ^ N ^ @     =
    m a n   ^ N   N     V
      t     n a   + ^ ^ a R   ^
              t     m n l e ^ n
                          c m
                          N
                          a
                          t
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) =(N+(m,n),Val(RecNat(m),n))
```

*For any natural numbers **m**, **n**, the addition of **m**, **n** is Val(RecNat(**m**), **n**).*

1359(T) 0 is neutral element for N+

```
A - + - - - + - +
^ @     =
m   ^ N   N     ^
    m a   + ^ 0 m
      t     m
```

```
A(m|@(m,Nat)) =(N+(m,0),m)
```

*Foe any natural number **m**, the adition of **m**, 0 is **m**.*

1360(T) N+ of m and Suc of n is Suc of N+ of m, n

```
A - + - - - - - - - + - - - - + -
^ @     A - + - - - - - + - - - +
m   ^ N ^ @     =
    m a n   ^ N   N       S
      t     n a   + ^ S   u N
              t     m u ^ c + ^ ^
                      c n     m n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) =(N+(m,Suc(n)),Suc(N+(m,n)))
```

*Let **m**, **n** be natural numbers. N+ of **m**, Suc(**n**) is equal to Suc(N+(**m**, **n**)).*
***m** + Suc(**n**) = Suc(**m** + **n**)*

1361(F) 1 definition (natural number 1)

```
=
  1 S
    u 0
    c
```

```
=(1,Suc(0))
```

1362(T) 1 is a natural number

```
 @
   1 N
     a
     t
```

```
@(1,Nat)
```

1369(T) N+ of 1, n is Suc of n

```
 A - + - - - - + - +
 ^ @     =
 n   ^ N   N     S
     n a   + 1 ^ u ^
       t       n c n
```

```
A(n|@(n,Nat)) =(N+(1,n),Suc(n))
```

*For any natural number **n**, N+ of 1, **n** is equal to Suc of **n**.*
*1 + **n** = Suc(**n**)*

1375(T) N+ of 0 and n is n

```
 A - + - - - - + +
 ^ @     =
 n   ^ N   N     ^
     n a   + 0 ^ n
       t       n
```

```
A(n|@(n,Nat)) =(N+(0,n),n)
```

*For any natural number **n**, N+ of 0, **n** is equal to **n**.*
*0 + **n** = **n***

1376(T) N+ belongs to Nat

```
 A - + - - - - - - - + - -
 ^ @     A - + - - - - + -
 m   ^ N ^ @     @
     m a n   ^ N   N     N
       t     n a   + ^ ^ a
               t     m n t
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) @(N+(m,n),Nat)
```

The sum of natural numbers is a natural number.

1382(T) N+ is associative

```
A - + - - - - - - - - - - - - - + - - - + - - -
^ @     A - + - - - - - - - - - + - - - - + -
m   ^ N ^ @     A - + - - - - - - + - - - - +
    m a n   ^ N ^ @     =
      t     n a p   ^ N   N         N
              t     p a   + N     ^ + ^ N
                      t     + ^ ^ p   m + ^ ^
                              m n         n p
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) A(p|@(p,Nat)) =(N+(N+(m,n),p),N+(m,N+(n,p)))
```

*For any natural numbers **m**, **n**, **p**, (**m** + **n**) + **p** = **m** + (**n** + **p**).*

1388(T) N+ is commutative

```
A - + - - - - - - - + - - - +
^ @     A - + - - - - + - + -
m   ^ N ^ @     =
    m a n   ^ N   N     N
      t     n a   + ^ ^ + ^ ^
              t     m n   n m
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) =(N+(m,n),N+(n,m))
```

*For any natural numbers **m**, **n**, **m** + **n** = **n** + **m**.*

1389(F) MplFun definition

```
A - + - - - + - - - - - - - - - - - +
^ @     =
m   ^ N   M   K - - - - - + - - - - -
    m a   p ^ ^ E - + - - - - + - + -
      t   l m u ^ @     =
          F     p   ^ N   ^ P
          u         p a   u a ^ N
          n           t     i p + ^ ^
                            r     p m
```

```
A(m|@(m,Nat)) =(MplFun(m),{u|E(p|@(p,Nat)) =(u,Pair(p,N+(p,m)))})
```

*Let **m** be a natural number. MplFun of **m** is equal to the class of all **u** for which there exists a natural number **p**, such that **u** is equal to the pair of **p** and the sum of **p**, **m**.*

1395(T) MplFun is Graph

```
A - + - - - +
^ @     G
m   ^ N r M
    m a a p ^
      t p l m
        h F
          u
          n
```

```
A(m|@(m,Nat)) Graph(MplFun(m))
```

1401(T) Dom of MplFun is Nat

```
A - + - - - - + -
^ @     =
m   ^ N   D     N
    m a   o M   a
      t   m p ^ t
            l m
            F
            u
            n
```

```
A(m|@(m,Nat)) =(Dom(MplFun(m)),Nat)
```

1403(T) MplFun is a graph function

```
A - + - - - +
^ @     F
m   ^ N u M
    m a n p ^
      t c l m
          F
          u
          n
```

```
A(m|@(m,Nat)) Func(MplFun(m))
```

1405(T) Ran of MplFun is included in Nat

```
A - + - - - - + -
^ @     P
m   ^ N a R     N
    m a r a M   a
      t t n p ^ t
            l m
            F
            u
            n
```

```
A(m|@(m,Nat)) Part(Ran(MplFun(m)),Nat)
```

1406(T) Fun of Nat, Nat, MplFun

```
A - + - - - - - +
^ @     F
m   ^ N u N N M
    m a n a a p ^
      t   t t l m
              F
              u
              n
```

```
A(m|@(m,Nat)) Fun(Nat,Nat,MplFun(m))
```

*For any natural number **m**, Fun(Nat, Nat, MplFun(**m**)).*

1407(F) N* definition (multiplication of natural numbers)

```
A - + - - - - - - - + - - - - - - + -
^ @     A - + - - - - + - - - - - - +
m   ^ N ^ @     =
    m a n   ^ N   N     V
      t     n a   * ^ ^ a R         ^
              t     m n l e N 0 M   n
                          c a   p ^
                          I t   l m
                          n     F
                          t     u
                                n
```

A(m|@(m,Nat)) A(n|@(n,Nat)) =(N*(m,n),Val(RecInt(Nat,0,MplFun(m)),n))

*Let **m**, **n** be natural numbers. The product of **m**, **n** is Val(RecInt(Nat,0,MplFun(**m**)),**n**).*

1408(T) Multiplication by 0

```
A - + - - - + - -
^ @     =
m   ^ N   N     0
    m a   * ^ 0
      t     m
```

A(m|@(m,Nat)) =(N*(m,0),0)

1409(T) Val of MplFun

```
A - + - - - - - - - - + - - - +
^ @     A - + - - - - - + - + -
m   ^ N ^ @     =
    m a n   ^ N   V       N
      t     n a   a M   ^ + ^ ^
              t   l p ^ n   n m
                    l m
                    F
                    u
                    n
```

A(m|@(m,Nat)) A(n|@(n,Nat)) =(Val(MplFun(m),n),N+(n,m))

*For any natural numbers **m**, **n**, Val(MplFun(**m**),**n**) is equal to **n** + **m**.*

1410(T) Val of RecInt belongs to Nat

```
A - + - - - - - - - - - - - - - - + - - -
^ @     A - - - + - - - - - - - - - + - -
a   ^ N ^ F       A - + - - - - - - - + -
    a a F u N N ^ ^ @     @
      t   n a a F n   ^ N   V           N
            t t       n a   a R       ^ a
                        t   l e N ^ ^ n t
                              c a a F
                              I t
                              n
                              t
```

```
A(a|@(a,Nat)) A(F|Fun(Nat,Nat,F)) A(n|@(n,Nat)) @(Val(RecInt(Nat,a,F),n),Nat)
```

1411(T) N* of m, n belongs to Nat

```
A - + - - - - - - - - + - -
^ @     A - + - - - - + -
m   ^ N ^ @     @
    m a n   ^ N   N     N
      t     n a   * ^ ^ a
              t     m n t
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) @(N*(m,n),Nat)
```

The product of natural numbers is a natural number.

1412(T) Multiplication by a successor

```
A - + - - - - - - - - + - - - - + - +
^ @     A - + - - - - - + - - - + -
m   ^ N ^ @     =
    m a n   ^ N   N       N
      t     n a   * ^ S   + N     ^
              t     m u ^   * ^ ^ m
                      c n     m n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) =(N*(m,Suc(n)),N+(N*(m,n),m))
```

1418(T) Multiplication of 0 by n is 0

```
A - + - - - - + -
^ @     =
n   ^ N   N     0
    n a   * 0 ^
      t       n
```

```
A(n|@(n,Nat)) =(N*(0,n),0)
```

1425(T) Multiplication of 1 by n is n

```
A - + - - - - + +
^ @     =
n   ^ N   N     ^
    n a   * 1 ^ n
      t       n
```

```
A(n|@(n,Nat)) =(N*(1,n),n)
```

*Let **n** be a natural number. N* of 1, **n** is equal to **n**.*
1·n = n

1433(T) Left distributivitiy of N* for N+

```
A - + - - - - - - - - - - - + - - - - - + - - + -
^ @     A - + - - - - - - - - - + - - - - + - - -
m   ^ N ^ @     A - + - - - - - - + - - - - - - +
    m a n   ^ N ^ @     =
      t     n a k   ^ N   N         N
              t     k a   * ^ N     + N     N
                      t     m + ^ ^   * ^ ^ * ^ ^
                                n k   m n     m k
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) A(k|@(k,Nat)) =(N*(m,N+(n,k)),N+(N*(m,n),N*(m,k)))
```

1441(T) Associativity of N*

```
A - + - - - - - - - - - - - - + - - - + - - -
^ @     A - + - - - - - - - - - + - - - - + -
m   ^ N ^ @     A - + - - - - - - + - - - - +
    m a n   ^ N ^ @     =
      t     n a k   ^ N   N         N
              t     k a   * N     ^ * ^ N
                      t     * ^ ^ k   m * ^ ^
                              m n         n k
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) A(k|@(k,Nat)) =(N*(N*(m,n),k),N*(m,N*(n,k)))
```

*Let **m**, **n**, **k** be natural numbers. N*(N*(**m**,**n**),**k**) is equal to N*(**m**,N*(**n**,**k**)).*
*(**m**·**n**) ·**k** = **m**·(**n**·**k**)*

1449(T) Multiplication of a successor by n

```
A - + - - - - - - - - + - - - + - -
^ @     A - + - - - - - + - - - + +
m   ^ N ^ @     =
    m a n   ^ N   N       N
      t     n a   * S   ^ + N     ^
              t     u ^ n   * ^ ^ n
                    c m       m n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) =(N*(Suc(m),n),N+(N*(m,n),n))
```

1450(T) 1 is a neutral element for multiplication

```
A - + - - - + - +
^ @     =
m   ^ N   N     ^
    m a   * ^ 1 m
      t     m
```

```
A(m|@(m,Nat)) =(N*(m,1),m)
```

*If **m** is a natural number, then **m**·1 = m.*

1458(T) Right distributivity of N* for N+

```
A - + - - - - - - - - - - - - - - - + - - - + - - +
^ @     A - + - - - - - - - - + - - - - + - - - -
m   ^ N ^ @     A - + - - - - - + - - - - - - + -
    m a n   ^ N ^ @     =
      t     n a k   ^ N   N         N
              t     k a   * N     ^ + N     N
                      t     + ^ ^ m   * ^ ^ * ^ ^
                              n k       n m   k m
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) A(k|@(k,Nat)) =(N*(N+(n,k),m),N+(N*(n,m),N*(k,m)))
```

*For any natural numbers **m**, **n**, **k**, N*(N+(**n**,**k**),**m**) is equal to N+(N*(**n**,**m**),N*(**k**,**m**)).*
*(**n** + **k**) ·**m** = **n**·**m** + **k**·**m***

1466(T) N* is commutative

```
A - + - - - - - - - + - - - +
^ @     A - + - - - - + - + -
m   ^ N ^ @     =
    m a n   ^ N   N     N
      t     n a   * ^ ^ * ^ ^
              t     m n   n m
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) =(N*(m,n),N*(n,m))
```

*For any natural numbers **m**, **n**, N*(**m**,**n**) is equal to N*(**n**,**m**).*
***m**·**n** = **n**·**m**.*

1467(R) N<= definition (less than or equal to for natural numbers)

```
A - + - - - - - - - + - - - + - - + -
^ @     A - + - - - - + - - - + - - +
m   ^ N ^ @     D
    m a n   ^ N   N     V
      t     n a   < ^ ^   @     =
              t   = m n     ^ ^   ^ ^
                            m n   m n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) D(N<=(m,n),V(@(m,n),=(m,n)))
```

1468(T) N<= is reflexive

```
A - + - - + +
^ @     N
m   ^ N < ^ ^
    m a = m m
      t
```

```
A(m|@(m,Nat)) N<=(m,m)
```

*For any natural number **m**, **m** <= **m**.*

1469(T) Belonging to a natural number implies inclusion

```
A - + - - - - + - - +
^ @     A - + - - + -
m   ^ N ^ @     P
    m a n   ^ ^ a ^ ^
      t     n m r n m
                t
```

```
A(m|@(m,Nat)) A(n|@(n,m)) Part(n,m)
```

*For any natural number **m**, and for any **n** belonging to **m**, **n** is included in **m**.*

1470(T) N<= is transitive

```
A - + - - - - - - - - - - - - + - - - - - + -
^ @     A - + - - - - - - - - - + - + - - - -
m   ^ N ^ @     A - + - - - - - - - - + - - +
    m a n   ^ N ^ @     C
      t     n a p   ^ N   X             N
              t     p a     N     N     < ^ ^
                      t     < ^ ^ < ^ ^ = m p
                            = m n = n p
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) A(p|@(p,Nat)) C(X(N<=(m,n),N<=(n,p)),N<=(m,p))
```

*Let **m**, **n**, **p** be natural numbers. If **m** <= **n** and **n** <= **p**, then **m** <= **p.***

1471(T) N<= is antisymmetric

```
A - + - - - - - - - - + - - - + - + -
^ @     A - + - - - - - + - + - - - +
m   ^ N ^ @     C
    m a n   ^ N   X             =
      t     n a     N     N       ^ ^
              t     < ^ ^ < ^ ^   m n
                    = m n = n m
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) C(X(N<=(m,n),N<=(n,m)),=(m,n))
```

*Let **m**, **n** be natural numbers. If **m** <= **n** and **n** <= **m**, then **m** = **n**.*

1478(T) 0 is less than or equal to every natural number

```
A - + - - - +
^ @     N
m   ^ N < 0 ^
    m a =   m
      t
```

```
A(m|@(m,Nat)) N<=(0,m)
```

1480(T) N<= is belonging to Suc

```
 A - + - - - - - - - + - - + - -
 ^ @     A - + - - - - + - - - +
 m   ^ N ^ @     D
     m a n   ^ N   N     @
       t     n a   < ^ ^   ^ S
               t   = m n   m u ^
                             c n
```

`A(m|@(m,Nat)) A(n|@(n,Nat)) D(N<=(m,n),@(m,Suc(n)))`

*Let **m**, **n** be natural numbers. **m** <= **n** iff **m** $\in$ Suc(**n**).*

1485(T) Equality of natural numbers implies equality of their successors

```
 A - + - - - - - - - + - - - + - -
 ^ @     A - + - - - - + - - - - - +
 m   ^ N ^ @     C
     m a n   ^ N   =     =
       t     n a     ^ ^   S   S
               t     m n   u ^ u ^
                           c m c n
```

`A(m|@(m,Nat)) A(n|@(n,Nat)) C(=(m,n),=(Suc(m),Suc(n)))`

1489(T) m belongs to n implies N<= of Suc(m), n

```
 A - + - - - - - - - + - - - + -
 ^ @     A - + - - - - + - - - +
 m   ^ N ^ @     C
     m a n   ^ N   @     N
       t     n a     ^ ^ < S   ^
               t     m n = u ^ n
                           c m
```

`A(m|@(m,Nat)) A(n|@(n,Nat)) C(@(m,n),N<=(Suc(m),n))`

*Let **m**, **n** be natural numbers. If **m** $\in$ **n**, then Suc(**m**) <= **n**.*

1490(T) m belongs to n implies Suc(m) belongs to Suc(n)

```
 A - + - - - - - - - + - - - + - -
 ^ @     A - + - - - - + - - - - - +
 m   ^ N ^ @     C
     m a n   ^ N   @     @
       t     n a     ^ ^   S   S
               t     m n   u ^ u ^
                           c m c n
```

`A(m|@(m,Nat)) A(n|@(n,Nat)) C(@(m,n),@(Suc(m),Suc(n)))`

*Let **m**, **n** be natural numbers. If **m** $\in$ **n**, then Suc(**m**) $\in$ Suc(**n**).*

1491(F) NLE definition (graph of N<=)

```
 =
   N K - - - - - - - - - - - - - + - - -
   L ^ E - + - - - - - - - + - - - - + -
   E u ^ @     E - + - - - - + - - - - +
       m   ^ N ^ @     X
           m a n   ^ N   N     =
             t     n a   < ^ ^   ^ P
                     t   = m n   u a ^ ^
                                   i m n
                                   r
```

```
=(NLE,{u|E(m|@(m,Nat)) E(n|@(n,Nat)) X(N<=(m,n),=(u,Pair(m,n)))})
```

1492(T) Pair of natural numbers is a set

```
 A - + - - - - - - - + -
 ^ @     A - + - - - - +
 m   ^ N ^ @     S
     m a n   ^ N   P
       t     n a   a ^ ^
               t   i m n
                   r
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) S(Pair(m,n))
```

1499(T) NLE explanation

```
 A - + - - - - - - - - + - - - + -
 ^ @     A - + - - - - - + - - - +
 m   ^ N ^ @     D
     m a n   ^ N   @         N
       t     n a     P     N < ^ ^
               t     a ^ ^ L = m n
                     i m n E
                     r
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) D(@(Pair(m,n),NLE),N<=(m,n))
```

*Let **m**, **n** be natural numbers. The pair of **m**, **n** belongs to NLE iff **m** <= **n**.*

1500(T) NLE is a graph

```
 G
 r N
 a L
 p E
 h
```

```
Graph(NLE)
```

1501(T) Rel of Nat, NLE (NLE is a relation in Nat)

```
 R
 e N N
 l a L
   t E

Rel(Nat,NLE)
```

1503(T) Nat, NLE is Ref (NLE is reflexive in Nat)

```
 R
 e N N
 f a L
   t E

Ref(Nat,NLE)
```

1506(T) NLE is Tra (NLE is a transitive graph)

```
 T
 r N
 a L
   E

Tra(NLE)
```

1508(T) NLE is Ant (NLE is an antisymmetric graph)

```
 A
 n N
 t L
   E

Ant(NLE)
```

1509(T) Nat, NLE is Ord (Nat is (partially) ordered by NLE)

```
 O
 r N N
 d a L
   t E

Ord(Nat,NLE)
```

1517(T) Nat with N<= is fully ordered

```
 A - + - - - - - - - + - - - +
 ^ @     A - + - - - - + - + -
 m   ^ N ^ @     V
     m a n   ^ N   N     N
       t     n a   < ^ ^ < ^ ^
               t   = m n = n m

A(m|@(m,Nat)) A(n|@(n,Nat)) V(N<=(m,n),N<=(n,m))
```

*Let **m**, **n** be natural numbers. We have **m** <= **n** or **n** <= **m**.*

1518(T) CmpAb(Nat, NLE, m, n)

```
A - + - - - - - - - - - + -
^ @     A - + - - - - - - +
m   ^ N ^ @     C
    m a n   ^ N m N N ^ ^
      t     n a p a L m n
              t A t E
                b
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) CmpAb(Nat,NLE,m,n)
```

1519(T) Nat is fully ordered

```
F
u N N
O a L
r t E
d
```

```
FuOrd(Nat,NLE)
```

1524(T) N+ of m, Suc(n) is N+ of Suc(m), n

```
A - + - - - - - - - + - - - - + -
^ @     A - + - - - - - + - - - - +
m   ^ N ^ @     =
    m a n   ^ N   N       N
      t     n a   + ^ S   + S   ^
              t     m u ^   u ^ n
                      c n   c m
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) =(N+(m,Suc(n)),N+(Suc(m),n))
```

*Let **m**, **n** be natural numbers. Then **m** + Suc(**n**) = Suc(**m**) + **n**.*

1531(T) Reduction for equality of natural numbers

```
A - + - - - - - - - - - - - - - + - - - - - - + -
^ @     A - + - - - - - - - - - - - - + - - - - +
m   ^ N ^ @     A - + - - - - - + - - + - - -
    m a n   ^ N ^ @     D
      t     n a k   ^ N   =             =
              t     k a     N     N       ^ ^
                      t     + ^ ^ + ^ ^   m n
                              m k   n k
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) A(k|@(k,Nat)) D(=(N+(m,k),N+(n,k)),=(m,n))
```

*Let **m**, **n**, **k** be natural numbers. **m** + **k** = **n** + **k** if and only if **m** = **n**.*

1533(T) A natural number less than or equal to 0 is 0

```
A - + - - - + - - + -
^ @     D
n   ^ N   N     =
    n a   < ^ 0   ^ 0
      t   = n     n
```

```
A(n|@(n,Nat)) D(N<=(n,0),=(n,0))
```

1534(T) m <= Suc(n) iff m <= n or m = Suc(n)

```
A - + - - - - - - - + - - - - + - - + - -
^ @     A - + - - - - - + - - - + - - - +
m   ^ N ^ @     D
    m a n   ^ N   N       V
      t     n a   < ^ S     N     =
              t   = m u ^   < ^ ^   ^ S
                      c n   = m n   m u ^
                                      c n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) D(N<=(m,Suc(n)),V(N<=(m,n),=(m,Suc(n))))
```

*Let **m**, **n** be natural numbers. **m** <= Suc(**n**) iff (**m** <= **n** ∨ **m** = Suc(**n**)).*

1544(T) Existence for N- (existence of difference of natural numbers)

```
A - + - - - - - - - - + - - - - - - - - - - +
^ @     A - - + - - + - - - - - - - - - + - -
m   ^ N ^ X             E - - - + - - - - + -
    m a n   @     N     ^ T X
      t       ^ N < ^ ^ d     @     =
              n a = n m         ^ N   N     ^
                t               d a   + ^ ^ m
                                  t     n d
```

```
A(m|@(m,Nat)) A(n|X(@(n,Nat),N<=(n,m))) E(d) X(@(d,Nat),=(N+(n,d),m))
```

*For any natural numbers **m**, **n** with **n** <= **m**, there exists **d** such that **d** is a natural number and **n** + **d** = **m**.*

1545(T) Uniqueness for N-

```
A - + - - - - - - - - + - - - - - - - - - + - - - - - - - - - + - - -
^ @     A - - + - - + - - - - - - - - + - - - - - - - - - + - - - - -
m   ^ N ^ X             A - - + - - - - + - - - - - - - - - - - - + -
    m a n   @     N     ^ X                 A - - + - - - - + - - - +
      t       ^ N < ^ ^ d   @     =         ^ X                 =
              n a = n m 1     ^ N   N     ^ d   @     =           ^ ^
                t             d a   + ^ ^ m 2     ^ N   N     ^   d d
                              1 t     n d         d a   + ^ ^ m   1 2
                                        1         2 t     n d
                                                            2
```

```
A(m|@(m,Nat)) A(n|X(@(n,Nat),N<=(n,m)))
A(d1|X(@(d1,Nat),=(N+(n,d1),m)))
A(d2|X(@(d2,Nat),=(N+(n,d2),m))) =(d1,d2)
```

1546(F) N- definition (partial operation N- (difference) of natural numbers)

```
A - + - - - - - - - - + - - - + - - - - - - - + - +
^ @     A - - + - - + - - - - - - + - - - - + - - + -
m   ^ N ^ X             X
    m a n   @     N       @         =
      t       ^ N < ^ ^     N     N   N         ^
              n a = n m     - ^ ^ a   + ^ N     m
                t             m n t     n - ^ ^
                                            m n
```

```
A(m|@(m,Nat)) A(n|X(@(n,Nat),N<=(n,m))) X(@(N-(m,n),Nat),=(N+(n,N-(m,n)),m))
```

*Let **m**, **n** be natural numbers, and **n** <= **m**. Then **m** – **n** is a natural number, and **n** + (**m** – **n**) = **m**.*

1547(T) A natural number is equal to 0 or 0 belongs to it

```
A - + - - - + - - - +
^ @     V
n   ^ N   =     @
    n a     ^ 0   0 ^
      t     n       n
```

```
A(n|@(n,Nat)) V(=(n,0),@(0,n))
```

*For any natural number **n**, **n** is equal to 0, or 0 belongs to **n**.*

1553(T) Difference of m and m is 0

```
A - + - - - + + -
^ @     =
m   ^ N   N     0
    m a   - ^ ^
      t     m m
```

```
A(m|@(m,Nat)) =(N-(m,m),0)
```

*For any natural number **m**, **m** – **m** = 0.*

1558(T) Difference null is equality

```
A - + - - - - - - - - + - - - + - - - + -
^ @     A - - + - - + - - - - - - + - - - - +
m   ^ N ^ X             D
    m a n   @     N       =         =
      t       ^ N < ^ ^     N     0   ^ ^
              n a = n m     - ^ ^     m n
                t             m n
```

```
A(m|@(m,Nat)) A(n|X(@(n,Nat),N<=(n,m))) D(=(N-(m,n),0),=(m,n))
```

*Let **m**, **n** be natural numbers with **n** <= **m**. Then **m** – **n** = 0 iff **m** = **n**.*

1563(T) A nat. number is less than or equal to its successor

```
 A - + - - + - +
 ^ @     N
 m   ^ N < ^ S
     m a = m u ^
       t     c m
```

`A(m|@(m,Nat)) N<=(m,Suc(m))`

1566(T) N<= is N<= for Suc

```
 A - + - - - - - - - + - - - + - -
 ^ @     A - + - - - - + - - - - +
 m   ^ N ^ @     D
     m a n   ^ N   N     N
       t     n a   < ^ ^ < S   S
               t   = m n = u ^ u ^
                           c m c n
```

`A(m|@(m,Nat)) A(n|@(n,Nat)) D(N<=(m,n),N<=(Suc(m),Suc(n)))`

*Let **m, n** be natural numbers. **m** <= **n** iff Suc(**m**) <= Suc(**n**).*

1573(T) Reduction for N<= of natural numbers

```
 A - + - - - - - - - - - - - + - - - + - - - -
 ^ @     A - + - - - - - - - - + - - - - - + -
 m   ^ N ^ @     A - + - - - - - - - - + - - +
     m a n   ^ N ^ @     D
       t     n a d   ^ N   N     N
               t     d a   < ^ ^ < N     N
                       t   = m n = + ^ ^ + ^ ^
                                     m d   n d
```

`A(m|@(m,Nat)) A(n|@(n,Nat)) A(d|@(d,Nat)) D(N<=(m,n),N<=(N+(m,d),N+(n,d)))`

*Let **m, n, d** be natural numbers. Then **m** <= **n** iff **m** + **d** <= **n** + **d**.*

1575(T) N- of m and 0 is m

```
 A - + - - - + - +
 ^ @     =
 m   ^ N   N     ^
     m a   - ^ 0 m
       t     m
```

`A(m|@(m,Nat)) =(N-(m,0),m)`

Various properties of the natural numbers 0, 1, 2

1577(T) 1 is the singleton of 0

```
 =
   1 S
     n 0
     g
```

```
=(1,Sng(0))

1 = {0}
```

1578(T) 0 belongs to 1

```
 @
   0 1

@(0,1)
```

1579(T) Belonging to 1 explanation

```
 A - - - + - - + -
 ^ T D
 x     @     =
         ^ 1   ^ 0
         x     x

A(x) D(@(x,1),=(x,0))
```

1580(F) 2 definition (natural number 2)

```
 =
   2 S
     u 1
     c

=(2,Suc(1))
```

1581(T) 2 is Dbl of 0, 1

```
 =
   2 D
     b 0 1
     l

=(2,Dbl(0,1))

2 = {0, 1}
```

1583(T) Belonging to 2 explanation

```
 A - - - + - - - + - - + -
 ^ T D
 x     @     V
         ^ 2   =     =
         x       ^ 0   ^ 1
                 x     x

A(x) D(@(x,2),V(=(x,0),=(x,1)))
```

*For any class **x**, **x** belongs to 2 iff **x** = 0 or **x** = 1.*

1584(T) 2 is a natural number

```
 @
   2 N
     a
     t

@(2,Nat)
```

1585(T) 0 belongs to 2

```
 @
   0 2

@(0,2)
```

1586(T) 1 belongs to 2

```
 @
   1 2

@(1,2)
```

1591(T) 0 and 1 are different

```
 N
   =
     0 1

N(=(0,1))
```

Natural numbers as domain (Sets, Finite, Path)

Let's remember the definition of a finite set:

*A set **S** is called finite if there exists a bijection **f** : {1, 2, … , n} → **S** for some natural number **n**.*

The domain of the function above is the set consisting of the elements 1, 2, and so on, up to **n**, where **n** is a natural number.

Among the problems we had in the process of adapting the mathematical statements to our system, we mentioned the case of the definition of the set $\{x_1, x_2, \ldots, x_n\}$. For the particular case of the set {1, 2, ... , n}, we will show here a simple and beautiful solution using the properties of natural numbers. Let's note that:

0 is the empty set

1 = Suc(0) = 0 U {0} = {0}

2 = Suc(1) = {0} U {{0}} = {0, {0}} = {0, 1}

3 = Suc(2) = {0, 1} U {{0, 1}} = {0, 1, {0, 1}} = {0, 1, 2}

Of course, we will show in a theorem that a natural number contains all its predecessors. This will also mean that a natural number **n** has exactly **n** elements.

The definition of a finite set could therefore be formulated much more simply as follows:

A set **S** is called finite if there exists a bijection **f** : **n** → **S** for some natural number **n**.

1801(T) OneToOne with a natural number implies set condition

```
A - + - - - + - - -
^ @     A - - + - +
n   ^ N ^ O     S
    n a X n ^ ^   ^
      t   e n X   X
          T
          o
          O
          n
          e
```

`A(n|@(n,Nat)) A(X|OneToOne(n,X)) S(X)`

*Let **n** be a natural number and **X** a class. If **n, X** are one-to-one, then **X** is a set.*

1802(F) Sets definition (class of the sets containing n elements)

```
A - + - - - + - - - + -
^ @     =
n   ^ N   S   K - - +
    n a   e ^ ^ O
      t   t n X n ^ ^
          s     e n X
                T
                o
                O
                n
                e
```

`A(n|@(n,Nat)) =(Sets(n),{X|OneToOne(n,X)})`

*Let **n** be a natural number. Sets(**n**) is the class of all sets with **n** elements.*

1803(T) Sets explanation

```
A - + - - - - - - - + - + -
^ @     A - - - + - - - - +
n   ^ N ^ T D
    n a X     @       O
      t         ^ S   n ^ ^
                X e ^ e n X
                  t n T
                  s   o
                      O
                      n
                      e
```

`A(n|@(n,Nat)) A(X) D(@(X,Sets(n)),OneToOne(n,X))`

*Let **n** be a natural number and **X** a class. **X** belongs to Sets(**n**) if and only if **n** and **X** are one-to-one.*

1804(F) Finite definition (class of finite sets)

```
 =
   F K - - - - - - + - -
   i ^ E - + - - - - +
   n X ^ @     @
   i   n   ^ N   ^ S
   t       n a   X e ^
   e         t     t n
                   s
```

```
=(Finite,{X|E(n|@(n,Nat)) @(X,Sets(n))})
```

1805(T) Belonging to Sets of 0 is equality to 0

```
 A - - - + - - - + -
 ^ T D
 X     @       =
         ^ S     ^ 0
         X e 0   X
           t
           s
```

```
A(X) D(@(X,Sets(0)),=(X,0))
```

*Let **X** be a class. **X** belongs to Sets of 0 if and only if **X** is equal to 0.*

1806(T) Sets of 0 is included in Sng of 0

```
 A - + - - - + - -
 ^ @       @
 X   ^ S     ^ S
     X e 0   X n 0
       t       g
       s
```

```
A(X|@(X,Sets(0))) @(X,Sng(0))
```

1808(T) Sets of 0 is equal to Sng of 0

```
 =
   S   S
   e 0 n 0
   t   g
   s
```

```
=(Sets(0),Sng(0))
```

The class of all the sets with 0 elements is equal to Sng of 0.

1811(T) Sets of 1 theorem

```
 =
   S   K - - - - + - -
   e 1 ^ E - + - - - +
   t   X ^ S   =
   s     x   ^   ^ S
             x   X n ^
                   g x
```

```
=(Sets(1),{X|E(x|S(x)) =(X,Sng(x))})
```

*Sets(1) is equal to the class of all Sng(**x**), where **x** is a set.*

1812(T) Sets of 2

```
 =
   S   K - - +
   e 2 ^ O
   t   X n 2 ^
   s     e   X
         T
         o
         O
         n
         e
```

```
=(Sets(2),{X|OneToOne(2,X)})
```

1813(T) Sets of 2 explanation

```
 A - - - + - - - - - +
 ^ T D
 X     @       O
         ^ S   n 2 ^
         X e 2 e   X
           t   T
           s   o
               O
               n
               e
```

```
A(X) D(@(X,Sets(2)),OneToOne(2,X))
```

1816(T) Sets of 2

```
 =
   S   K - - - - - - - - - - - - + - - -
   e 2 ^ E - + - - - - - - + - - - - + -
   t   X ^ S   E - - + - - - + - - - - +
   s     x   ^ ^ X             =
             x y   S   N         ^ D
                     ^   =       X b ^ ^
                     y     ^ ^     l x y
                           x y
```

```
=(Sets(2),{X|E(x|S(x)) E(y|X(S(y),N(=(x,y)))) =(X,Dbl(x,y))})
```

*Sets(2) is equal to the class of all Dbl(**x**, **y**) , where **x**, **y** are different sets.*

1832(T) Dis of X and x, where x is not in X

```
A - - - - - - - - + - + - -
^ T A - - + - - + - - - - +
X   ^ X             D
    x   S   N       i ^ S
          ^   @     s X n ^
          x     ^ ^     g x
                x X
```

A(X) A(x|X(S(x),N(@(x,X)))) Dis(X,Sng(x))

1833(T) TraSet corollary

```
A - - - + - - + - - - + - - - - - - +
^ T D
X     T   X
      r ^   S   A - + - - - - + - - -
      a X     ^ ^ @     A - + - - + -
      S       X x   ^ ^ ^ @     @
      e             x X y   ^ ^   ^ ^
      t                     y x   y X
```

A(X) D(TraSet(X),X(S(X),A(x|@(x,X)) A(y|@(y,x)) @(y,X)))

*Let **X** be a class. **X** is a transitive set if and only if **X** is a set, and for any **x** in **X** and for any **y** in **x**, **y** belongs to **X**.*

1835(T) Union of TraSet is TraSet

```
A - + - - - - - + -
^ T   A - + - - - +
X r ^ ^ T   T
  a X Y r ^ r U
  S     a Y a   ^ ^
  e     S   S   X Y
  t     e   e
        t   t
```

A(X|TraSet(X)) A(Y|TraSet(Y)) TraSet(U(X,Y))

The union of transitive sets is a transitive set.

1837(T) Intersection of TraSet is TraSet

```
A - + - - - - - + -
^ T   A - + - - - +
X r ^ ^ T   T
  a X Y r ^ r I
  S     a Y a   ^ ^
  e     S   S   X Y
  t     e   e
        t   t
```

A(X|TraSet(X)) A(Y|TraSet(Y)) TraSet(I(X,Y))

The intersection of transitive sets is a transitive set.

1844(T) A natural number is not self-containing

```
 A - + - - - + +
 ^ @     N
 n   ^ N   @
     n a     ^ ^
       t     n n
```

```
A(n|@(n,Nat)) N(@(n,n))
```

Let's note that the axiom of regularity has not been used in the proof of this theorem.

1845(T) A natural number and its Sng are disjoint

```
 A - + - - + - +
 ^ @     D
 n   ^ N i ^ S
     n a s n n ^
       t     g n
```

```
A(n|@(n,Nat)) Dis(n,Sng(n))
```

1847(T) Adding a new element to a set

```
 A - + - - - - - + - - - - - - - - - - - - - - - +
 ^ @     A - + - - - - - - - - - - + - - + - - - - -
 n   ^ N ^ @         A - - + - - + - - - - - - + - - -
     n a X   ^ S   ^ X                 @
       t     X e ^ x   S   N             U       S
               t n       ^   @             ^ S   e S
               s         x     ^ ^         X n ^ t u ^
                               x X           g x s c n
```

```
A(n|@(n,Nat)) A(X|@(X,Sets(n))) A(x|X(S(x),N(@(x,X))))
@(U(X,Sng(x)),Sets(Suc(n)))
```

*Let **n** be a natural number and **X** be a set with **n** elements. If the set **x** does not belong to X, then the union of **X** and Sng of **x** has **n** + 1 elements.*

1848(T) Belonging implies not null

```
 A - - - - - + - - - - -
 ^ T A - - - - + - - + -
 x   ^ T C
     y     @     N
             ^ ^   =
             x y     ^ 0
                     y
```

```
A(x) A(y) C(@(x,y),N(=(y,0)))
```

*If **x** belongs to **y**, then **y** is not empty.*

1849(A) Axiom of regularity

```
A - - + - - - - + - - - - - - - + - - - -
^ N       E - + - - - - - - - - - - - +
X   =     ^ @     A - - - - + - - - + -
      ^ 0 x   ^ ^ ^ T V
      X       x X u     N       N
                          @       @
                            ^ ^     ^ ^
                            u X     u x
```

`A(X|N(=(X,0))) E(x|@(x,X)) A(u) V(N(@(u,X)),N(@(u,x)))`

*For any not empty class **X**, there exists **x** in **X** such that for any **u**, **u** does not belong to **X** or **u** does not belong to **x**.*

According to theorem 1844, there are no self-contained natural numbers. Until now, however, we have not been able to prove the same for any set. Now, with the help of the regularity axiom, we will prove this.

1850(T) A set is not self-contained

```
A - + - - + +
^ S   N
X   ^   @
    X     ^ ^
          X X
```

`A(X|S(X)) N(@(X,X))`

2385(T) Axiom of regularity corollary

```
A - - + - - - - + - - + - -
^ N       E - + - - - - + -
X   =     ^ @     =
      ^ 0 x   ^ ^   I     0
      X       x X     ^ ^
                      X x
```

`A(X|N(=(X,0))) E(x|@(x,X)) =(I(X,x),0)`

*For any not empty **X**, there exists **x** in **X** such that the intersection of **X** and **x** is empty.*

2386(T) No mutual belonging

```
A - - - - - - + - - - +
^ T A - - - - - + - + -
a   ^ T N
    b     X
            @     @
              ^ ^   ^ ^
              a b   b a
```

`A(a) A(b) N(X(@(a,b),@(b,a)))`

*For any **a**, **b**, not(**a** belongs to **b** and **b** belongs to **a**).*

1851(T) Existence for Pre (predecessor of a not null natural number)

```
A - - + - - - + - - - - - - - - + - -
^ X               E - - - + - - - - +
m   @     N       ^ T X
      ^ N   =     n     @     =
      m a     ^ 0         ^ N   ^ S
        t     m           n a   m u ^
                            t     c n
```

```
A(m|X(@(m,Nat),N(=(m,0)))) E(n) X(@(n,Nat),=(m,Suc(n)))
```

*For any not null natural number **m**, there exists a natural number **n** such that **m** is the successor of **n**.*

1852(T) Uniqueness for Pre

```
A - - + - - - + - - - - - - - + - - - - - - - - + - - - - -
^ X               A - - + - - - - + - - - - - - - - - - + -
m   @     N       ^ X               A - - + - - - - + - - +
      ^ N   =     n   @     =       ^ X               =
      m a     ^ 0 1     ^ N   ^ S   n   @     =         ^ ^
        t     m         n a   m u ^ 2     ^ N   ^ S     n n
                        1 t     c n       n a   m u ^   1 2
                                  1       2 t     c n
                                                    2
```

```
A(m|X(@(m,Nat),N(=(m,0))))
A(n1|X(@(n1,Nat),=(m,Suc(n1))))
A(n2|X(@(n2,Nat),=(m,Suc(n2)))) =(n1,n2)
```

*If **m**, $\boldsymbol{n_1}$, $\boldsymbol{n_2}$ are natural numbers with **m** not null, **m** = Suc($\boldsymbol{n_1}$), and **m** = Suc($\boldsymbol{n_2}$), then $\boldsymbol{n_1}$ = $\boldsymbol{n_2}$.*

1853(F) Pre definition (predecessor of a not null natural number)

```
A - - + - - - + - - - - + - - + - - +
^ X               X
n   @     N         @       =
      ^ N   =         P   N   ^ S
      n a     ^ 0     r ^ a   n u P
        t     n       e n t     c r ^
                                  e n
```

```
A(n|X(@(n,Nat),N(=(n,0)))) X(@(Pre(n),Nat),=(n,Suc(Pre(n))))
```

*For any not null natural number **n**, Pre(**n**) is a natural number, and **n** = Suc(Pre(**n**)).*

1854(T) Pre of Suc

```
A - + - - - - + +
^ @     =
n   ^ N   P     ^
    n a   r S   n
      t   e u ^
            c n
```

```
A(n|@(n,Nat)) =(Pre(Suc(n)),n)
```

1855(T) 1 is less than or equal to any not null natural number

```
A - - + - - - + - - - +
^ X               N
n   @     N       < 1 ^
      ^ N   =     =   n
      n a     ^ 0
        t     n
```

`A(n|X(@(n,Nat),N(=(n,0)))) N<=(1,n)`

1856(T) Pre of n is difference of n, 1

```
A - - + - - - + - - - + - + -
^ X               =
n   @     N         P   N
      ^ N   =       r ^ - ^ 1
      n a     ^ 0   e n   n
        t     n
```

`A(n|X(@(n,Nat),N(=(n,0)))) =(Pre(n),N-(n,1))`

*For any not null natural number **n**, Pre(**n**) = **n** – 1.*

Sometimes a theorem seems trivial and not absolutely necessary. True, but if this makes it easier to prove another theorem, then maybe it's still worth the effort. This is the case with the following theorem:

1870(T) Successor of n is union of n and singleton of n

```
A - + - - - + - + - +
^ @     =
n   ^ N   S   U
    n a   u ^   ^ S
      t   c n   n n ^
                  g n
```

`A(n|@(n,Nat)) =(Suc(n),U(n,Sng(n)))`

1871(T) Class difference of successor and singleton

```
A - + - - - - - + - + +
^ @     =
n   ^ N   \         ^
    n a     S   S   n
      t     u ^ n ^
            c n g n
```

`A(n|@(n,Nat)) =(\(Suc(n),Sng(n)),n)`

*For any natural number **n**, the class difference of Suc(**n**) and Sng(**n**) is equal to **n**.*

1876(T) A natural number belongs to its successor

```
 A - + - - + - +
 ^ @     @
 n   ^ N   ^ S
     n a   n u ^
       t     c n
```

```
A(n|@(n,Nat)) @(n,Suc(n))
```

1886(T) Extracting an element

```
 A - + - - - - - - + - - - - - - - - - - - +
 ^ @     A - + - - - - - - - + - - + - - - -
 n   ^ N ^ @           E - + - - - - - - + - -
     n a X   ^ S       ^ @       @
       t     X e S     x   ^ ^     \       S
               t u ^       x X       ^ S   e ^
               s c n                 X n ^ t n
                                       g x s
```

```
A(n|@(n,Nat)) A(X|@(X,Sets(Suc(n)))) E(x|@(x,X)) @(\(X,Sng(x)),Sets(n))
```

*Let **n** be a natural number and **X** a class having Suc(**n**) elements. There exists an **x** in **X** such that **X** \ {**x**} has **n** elements.*

1893(T) Sets of union of disjoint sets

```
 A - + - - - - - - - - - + - - - - - - - - - - - - - - - - + -
 ^ @     A - + - - - - - - - - - - - + - - - - - - - - - - - - +
 m   ^ N ^ @     A - + - - - - - - - - - - - + - - - + - - - - -
     m a n   ^ N ^ @         A - + - - - - - + - - - + - - - -
       t     n a X   ^ S   ^ @       C
               t     X e ^ Y   ^ S     D     @
                       t m     Y e ^   i ^ ^   U     S
                       s         t n   s X Y     ^ ^ e N
                                 s               X Y t + ^ ^
                                                     s   m n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat))
A(X|@(X,Sets(m))) A(Y|@(Y,Sets(n)))
C(Dis(X,Y),@(U(X,Y),Sets(N+(m,n))))
```

*Let **m**, **n** be natural numbers, **X** a set with **m** elements, and **Y** a set with **n** elements. If **X**, **Y** are disjoint, then the union of **X**, **Y** has **m** + **n** elements.*

1895(T) A not null natural number is a successor of a natural number

```
 A - - + - - - + - - - - - - - + - -
 ^ X                 E - + - - - - +
 m   @     N         ^ @       =
       ^ N   =       n   ^ N   ^ S
       m a     ^ 0     n a   m u ^
         t     m         t     c n
```

```
A(m|X(@(m,Nat),N(=(m,0)))) E(n|@(n,Nat)) =(m,Suc(n))
```

*For any not null natural number **m**, there exists a natural number **n** such that **m** is equal to Suc of **n**.*

1903(T) An element of a natural number is a natural number

```
 A - + - - - - + - - -
 ^ @     A - + - - + -
 m   ^ N ^ @     @
     m a n   ^ ^   ^ N
       t     n m   n a
                     t
```

```
A(m|@(m,Nat)) A(n|@(n,m)) @(n,Nat)
```

*If **m** is a natural number and **n** is an element of **m**, then **n** is a natural number.*

1906(T) Suc and N<=

```
 A - + - - - + - - - - - - - - +
 ^ @     =
 n   ^ N   S   K - - + - - + -
     n a   u ^ ^ X
       t   c n m   @     N
                     ^ N < ^ ^
                     m a = m n
                       t
```

```
A(n|@(n,Nat)) =(Suc(n),{m|X(@(m,Nat),N<=(m,n))})
```

*The successor of the natural number **n** is equal to the set of all natural numbers **m** such that **m** <= **n**.*

1909(T) Part of Suc is N<=

```
 A - + - - - - - - - - - + - - - + -
 ^ @     A - + - - - - - - - + - - +
 m   ^ N ^ @     D
     m a n   ^ N   P         N
       t     n a   a S   S   < ^ ^
               t   r u ^ u ^ = m n
                   t c m c n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) D(Part(Suc(m),Suc(n)),N<=(m,n))
```

*Let **m**, **n** be natural numbers. Suc(**m**) is included in Suc(**n**) if and only if **m** <= **n**.*

1913(T) A natural number is the class difference of Suc and Sng

```
 A - + - - + - - + - +
 ^ @     =
 m   ^ N   ^ \
     m a   m   S   S
       t       u ^ n ^
               c m g m
```

```
A(m|@(m,Nat)) =(m,\(Suc(m),Sng(m)))
```

1915(T) Natural number explanation with N<=

```
A - + - - + - - - - - - - - + - - - +
^ @     =
n   ^ N   ^ K - - + - - - + - - - + -
    n a   n ^ X
      t     m   @     X
                  ^ N   N     N
                  m a   < ^ ^   =
                    t   = m n     ^ ^
                                  m n
```

`A(n|@(n,Nat)) =(n,{m|X(@(m,Nat),X(N<=(m,n),N(=(m,n))))})`

*A natural number **n** is equal to the class of all natural numbers less than **n**.*

1916(T) Predecessor of a not null natural number belongs to that number

```
A - - + - - - + - - - + +
^ X               @
n   @     N         P   ^
      ^ N   =       r ^ n
      n a     ^ 0   e n
        t     n
```

`A(n|X(@(n,Nat),N(=(n,0)))) @(Pre(n),n)`

1917(T) Part and Pre

```
A - - + - - - + - - - - - - - + - - - + -
^ X               A - + - - - - + - - - +
m   @     N       ^ @     C
      ^ N   =     n   ^ N   P     @
      m a     ^ 0     n a   a ^ ^   P   ^
        t     m         t   r m n   r ^ n
                            t       e m
```

`A(m|X(@(m,Nat),N(=(m,0)))) A(n|@(n,Nat)) C(Part(m,n),@(Pre(m),n))`

*Let **m** be a not null natural number and **n** a natural number. If **m** is included in **n**, then the predecessor of **m** belongs to **n**.*

1921(T) X included in 0 iff X = 0

```
A - - - + - - + -
^ T D
X     P     =
      a ^ 0   ^ 0
      r X     X
      t
```

`A(X) D(Part(X,0),=(X,0))`

*Let **X** be a class. **X** is included in 0 if and only if **X** is equal to 0.*

1923(T) Part is N<=

```
A - + - - - - - - - + - - + -
^ @     A - + - - - - + - - +
m   ^ N ^ @     D
    m a n   ^ N   P     N
      t     n a   a ^ ^ < ^ ^
              t   r m n = m n
                  t
```

`A(m|@(m,Nat)) A(n|@(n,Nat)) D(Part(m,n),N<=(m,n))`

*Let **m**, **n** be natural numbers. **m** is included in **n** if and only if **m** <= **n**.*

For natural numbers, inclusion is equivalent to the relation “less than or equal to”.

1924(T) A natural number is included in its successor

```
A - + - - + - +
^ @     P
n   ^ N a ^ S
    n a r n u ^
      t t   c n
```

`A(n|@(n,Nat)) Part(n,Suc(n))`

1925(T) A natural number is not equal to its successor

```
A - + - - - + - +
^ @     N
n   ^ N   =
    n a     ^ S
      t     n u ^
              c n
```

`A(n|@(n,Nat)) N(=(n,Suc(n)))`

1926(T) An element of a natural number cannot contain that number

```
A - + - - - - + - - + -
^ @     A - + - - - - +
m   ^ N ^ @     N
    m a n   ^ ^   @
      t     n m     ^ ^
                    m n
```

`A(m|@(m,Nat)) A(n|@(n,m)) N(@(m,n))`

1927(T) Transitivity of belonging for natural numbers

```
A - + - - - - - - - - - - - - - + - - - - - - + -
^ @     A - + - - - - - - - - - - - + - + - - - -
m   ^ N ^ @     A - + - - - - - - - - - + - - - +
    m a n   ^ N ^ @     C
      t     n a p   ^ N   X             @
              t     p a     @     @       ^ ^
                      t       ^ ^   ^ ^   m p
                              m n   n p
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) A(p|@(p,Nat)) C(X(@(m,n),@(n,p)),@(m,p))
```

*Let **m, n, p** be natural numbers. If **m** belongs to **n** and **n** belongs to **p**, then **m** belongs to **p**.*

1928(T) If m belongs to n, then Suc(m) is included in n

```
A - + - - - - - - - + - - - + -
^ @     A - + - - - - + - - - +
m   ^ N ^ @     C
    m a n   ^ N   @     P
      t     n a     ^ ^ a S   ^
              t     m n r u ^ n
                        t c m
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) C(@(m,n),Part(Suc(m),n))
```

1929(T) A natural number is included in the set of natural numbers

```
A - + - - + -
^ @     P
m   ^ N a ^ N
    m a r m a
      t t   t
```

```
A(m|@(m,Nat)) Part(m,Nat)
```

1930(T) Nat is a transitive set

```
T
r N
a a
S t
e
t
```

```
TraSet(Nat)
```

1931(T) Suc of x belongs to Nat implies x belongs to Nat

```
A - + - - - + - - - + -
^ S   C
x   ^   @       @
    x     S   N   ^ N
          u ^ a   x a
          c x t     t
```

```
A(x|S(x)) C(@(Suc(x),Nat),@(x,Nat))
```

1932(T) A natural number is not a successor set

```
A - + - - - +
^ @     N
m   ^ N   S
    m a   u ^
      t   c m
          S
          e
          t
```

```
A(m|@(m,Nat)) N(SucSet(m))
```

1933(T) A natural number is not included in an element of it

```
A - + - - - - + - - + -
^ @     A - + - - - - +
m   ^ N ^ @     N
    m a n   ^ ^   P
      t     n m   a ^ ^
                  r m n
                  t
```

```
A(m|@(m,Nat)) A(n|@(n,m)) N(Part(m,n))
```

1936(T) GU of Suc

```
A - + - - - - + +
^ @     =
m   ^ N   G     ^
    m a   U S   m
      t     u ^
            c m
```

```
A(m|@(m,Nat)) =(GU(Suc(m)),m)
```

*Let **m** be a natural number. The general union of the Suc(**m**) is equal to **m**.*

1939(T) General union of Nat is Nat

```
=
  G   N
  U N a
    a t
    t
```

```
=(GU(Nat),Nat)
```

1940(T) N<= of m, n implies Fun of m, n, Ident(m)

```
A - + - - - - - - - + - - + - - +
^ @     A - + - - - - + - - + - -
m   ^ N ^ @     C
    m a n   ^ N   N     F
      t     n a   < ^ ^ u ^ ^ I
              t   = m n n m n d ^
                              e m
                              n
                              t
```

`A(m|@(m,Nat)) A(n|@(n,Nat)) C(N<=(m,n),Fun(m,n,Ident(m)))`

1941(T) N<= of m, n implies Inj of m, n, Ident(m)

```
A - + - - - - - - - + - - + - - +
^ @     A - + - - - - + - - + - -
m   ^ N ^ @     C
    m a n   ^ N   N     I
      t     n a   < ^ ^ n ^ ^ I
              t   = m n j m n d ^
                              e m
                              n
                              t
```

`A(m|@(m,Nat)) A(n|@(n,Nat)) C(N<=(m,n),Inj(m,n,Ident(m)))`

1944(T) Bij of m, n, Ident(m) iff m = n

```
A - + - - - - - - - + - - + - + -
^ @     A - + - - - - + - - - - +
m   ^ N ^ @     D
    m a n   ^ N   B         =
      t     n a   i ^ ^ I     ^ ^
              t   j m n d ^   m n
                        e m
                        n
                        t
```

`A(m|@(m,Nat)) A(n|@(n,Nat)) D(Bij(m,n,Ident(m)),=(m,n))`

1978(T) Bij of Suc of m, Suc of n, F

```
A - + - - - - - - - - + - - - - - - + - -
^ @     A - + - - - - - - + - - - - - + -
m   ^ N ^ @     A - - - - - + - - - - - -
    m a n   ^ N ^ B           E - - - - +
      t     n a F i S   S   ^ ^ T B
              t   j u ^ u ^ F G   i ^ ^ ^
                    c m c n       j m n G
```

`A(m|@(m,Nat)) A(n|@(n,Nat)) A(F|Bij(Suc(m),Suc(n),F)) E(G) Bij(m,n,G)`

*For any natural numbers **m**, **n**, and for any **F** with Bij(Suc(**m**), Suc(**n**), **F**), there exists **G** such that Bij(**m**, **n**, **G**).*

1986(T) A Bij between two natural numbers implies their equality

```
A - + - - - - - - - + - - - + -
^ @     A - + - - - - + - - - +
m   ^ N ^ @     A - - - + - - -
    m a n   ^ N ^ B       =
      t     n a F i ^ ^ ^   ^ ^
              t   j m n F   m n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) A(F|Bij(m,n,F)) =(m,n)
```

*Let **m**, **n** be natural numbers, and **F** a class. If Bij(**m**, **n**, **F**), then **m** is equal to **n**.*

1987(T) Belonging to Sets of two natural numbers implies their equality

```
A - + - - - - - - - - - - + - - - - - + -
^ @     A - + - - - - - - - - - - + - - +
m   ^ N ^ @     A - - + - - - + - - - - -
    m a n   ^ N ^ X                 =
      t     n a X   @       @         ^ ^
              t       ^ S     ^ S     m n
                      X e ^   X e ^
                        t m     t n
                        s       s
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) A(X|X(@(X,Sets(m)),@(X,Sets(n)))) =(m,n)
```

1988(T) Sets of two different natural numbers are disjoint

```
A - + - - - - - - - - + - - - + - -
^ @     A - + - - - - - + - - - - +
m   ^ N ^ @     C
    m a n   ^ N   N       D
      t     n a     =     i S   S
              t       ^ ^ s e ^ e ^
                      m n   t m t n
                            s   s
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) C(N(=(m,n)),Dis(Sets(m),Sets(n)))
```

*Let **m**, **n** be natural numbers. if **m**, **n** are different, then Sets(**m**) and Sets(**n**) are disjoint.*

1992(T) Extracting an element

```
A - + - - - - - - + - - - - - - - - - - +
^ @     A - + - - - - - - + - - + - - - -
n   ^ N ^ @         A - + - - - - - + - -
    n a X   ^ S     ^ @     @
      t     X e S   x   ^ ^   \       S
              t u ^     x X     ^ S   e ^
              s c n             X n ^ t n
                                  g x s
```

```
A(n|@(n,Nat)) A(X|@(X,Sets(Suc(n)))) A(x|@(x,X)) @(\(X,Sng(x)),Sets(n))
```

*Let **n** be a natural number and **X** a class belonging to Sets(Suc(**n**)). For any **x** in **X**, the class difference of **X**, Sng(**x**) belongs to Sets(**n**).*

1996(T) 0 belongs to Sets of 0

```
 @
   0 S
     e 0
     t
     s
```

```
@(0,Sets(0))
```

1997(T) 0 is a finite set

```
 @
   0 F
     i
     n
     i
     t
     e
```

```
@(0,Finite)
```

1998(T) Finite explanation

```
 A - - - + - - - - - - - + - -
 ^ T D
 X     @     E - + - - - - +
         ^ F ^ @     @
         X i m   ^ N   ^ S
           n     m a   X e ^
           i       t     t m
           t             s
           e
```

```
A(X) D(@(X,Finite),E(m|@(m,Nat)) @(X,Sets(m)))
```

*Let **X** be a class. **X** is finite if and only if there exists a natural number **m** such that **X** belongs to Sets(**m**).*

1999(T) Belonging to Sets implies belonging to Finite

```
 A - + - - - - - - + - - -
 ^ @     A - + - - - + -
 m   ^ N ^ @       @
     m a X   ^ S     ^ F
       t     X e ^   X i
               t m     n
               s       i
                       t
                       e
```

```
A(m|@(m,Nat)) A(X|@(X,Sets(m))) @(X,Finite)
```

*For any natural number **m**, and for any **X** in Sets(**m**), **X** is finite.*

2006(T) Belonging to Sets implies a subset is finite

```
A - + - - - - - + - - - - - - -
^ @     A - + - - - - - + - - -
m   ^ N ^ @       A - + - - + -
    m a X   ^ S   ^ P     @
      t     X e ^ Y a ^ ^   ^ F
              t m   r Y X   Y i
              s     t         n
                              i
                              t
                              e
```

`A(m|@(m,Nat)) A(X|@(X,Sets(m))) A(Y|Part(Y,X)) @(Y,Finite)`

*For any natural number **m**, for any **X** in Sets(**m**), and for any **Y** included in **X**, **Y** is finite.*

2015(T) N<= explanation

```
A - + - - - - - - - - + - - - - - + - - -
^ @     A - + - - - + - - - - - - - - + -
m   ^ N ^ @     D
    m a n   ^ N   N     E - + - - - - - +
      t     n a   < ^ ^ ^ @     =
              t   = n m d   ^ N   ^ N
                            d a   m + ^ ^
                              t       n d
```

`A(m|@(m,Nat)) A(n|@(n,Nat)) D(N<=(n,m),E(d|@(d,Nat)) =(m,N+(n,d)))`

*Let **m**, **n** be natural numbers. **n** <= **m** if and only if there exists a natural number **d** such that **m** is equal to **n** + **d**.*

2024(T) Suc(m) <= n implies n is not null

```
A - + - - - - - - - - + - - - - -
^ @     A - + - - - - - + - - + -
m   ^ N ^ @     C
    m a n   ^ N   N       N
      t     n a   < S   ^   =
              t   = u ^ n     ^ 0
                    c m       n
```

`A(m|@(m,Nat)) A(n|@(n,Nat)) C(N<=(Suc(m),n),N(=(n,0)))`

*Let **m**, **n** be natural numbers. If Suc(**m**) is less than or equal to **n**, then **n** is not null.*

2026(T) Associativity N+ N- (by misusing the term associativity)

```
A - + - - - - - - - - - - - - - - - + - - - - - + - -
^ @     A - + - - - - - - - - + - - - - + - - - - + -
m   ^ N ^ @     A - + - - - + - - - - - - + - - - - +
    m a n   ^ N ^ @     C
      t     n a p   ^ N   N     =
              t     p a   < ^ ^   N         N
                      t   = p n   + ^ N     - N     ^
                                    m - ^ ^   + ^ ^ p
                                        n p     m n
```

A(m|@(m,Nat)) A(n|@(n,Nat)) A(p|@(p,Nat)) C(N<=(p,n),=(N+(m,N-(n,p)),N-(N+(m,n),p)))

*Let **m**, **n**, **p** be natural numbers. If **p** <= **n**, then **m** + (**n** - **p**) = (**m** + **n**) – **p**.*

2027(T) If p <= n then Suc(n – p) = Suc(n) – p

```
A - + - - - - - - - - + - - - + - - - + -
^ @     A - + - - - + - - - - - + - - - +
n   ^ N ^ @     C
    n a p   ^ N   N     =
      t     p a   < ^ ^   S       N
              t   = p n   u N     - S   ^
                          c - ^ ^   u ^ p
                              n p   c n
```

A(n|@(n,Nat)) A(p|@(p,Nat)) C(N<=(p,n),=(Suc(N-(n,p)),N-(Suc(n),p)))

2029(T) Union and Sets

```
A - + - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - - - + - -
^ @     A - + - - - - - - - - - - - - - - + - - - - - - - - - - - - - - - + -
m   ^ N ^ @     A - + - - - - - - - - - - - - - - - - - + - - - - - - - - - +
    m a n   ^ N ^ @     A - + - - - - - - - - - - + - - - - - + - - - - - - -
      t     n a p   ^ N ^ @       A - + - - - - - - + - - - - - + - - - - - -
              t     p a X   ^ S   ^ @       C
                      t     X e ^ Y   ^ S     @           @
                              t m     Y e ^     I     S     U     S
                              s         t n       ^ ^ e ^     ^ ^ e N
                                        s         X Y t p     X Y t - N     ^
                                                      s           s   + ^ ^ p
                                                                        m n
```

A(m|@(m,Nat))
A(n|@(n,Nat))
A(p|@(p,Nat))
A(X|@(X,Sets(m)))
A(Y|@(Y,Sets(n)))
C(@(I(X,Y),Sets(p)),@(U(X,Y),Sets(N-(N+(m,n),p))))

*Let **m**, **n**, **p** be natural numbers and **X**, **Y** be classes. If **X** belongs to Sets(**m**), **Y** belongs to Sets(**n**), and the intersection of X, Y belongs to Sets(**p**), then the union of **X**, **Y** belongs to Sets((**m** + **n**) - **p**).*

2030(T) Dom and Ran are sets implies the graph is a set

```
A - + - - - - + - - + - +
^ G   C
G r ^   X             S
  a G     S     S       ^
  p         D     R     G
  h         o ^   a ^
            m G   n G
```

A(G|Graph(G)) C(X(S(Dom(G)),S(Ran(G))),S(G))

2031(T) Theorem of replacement

```
A - + - - - - - - - - + - - - - - - - -
^ S   A - - - - - - - - - - - + - - - +
X   ^ ^ T A - + - - - - - + - - - + - -
    X Y   ^ F   C
          F u ^   X                 S
            n F     =       =         ^
            c         ^ D     ^ R     Y
                      X o ^   Y a ^
                        m F     n F
```

```
A(X|S(X)) A(Y) A(F|Func(F)) C(X(=(X,Dom(F)),=(Y,Ran(F))),S(Y))
```

2074(T) Adding two N<= member by member

```
A - + - - - - - - - - - - - - - - - - + - - - - - - + - - - -
^ @     A - + - - - - - - - - - - - - - + - - - - - - - - + -
m   ^ N ^ @     A - + - - - - - - - - - - - + - - - - + - - -
1   m a n   ^ N ^ @     A - + - - - - - - - - + - - - - - - +
    1 t 1   n a m   ^ N ^ @     C
            1 t 2   m a n   ^ N   X             N
                    2 t 2   n a     N     N     < N     N
                            2 t     < ^ ^ < ^ ^ = + ^ ^ + ^ ^
                                    = m n = m n     m m   n n
                                      1 1   2 2     1 2   1 2
```

```
A(m1|@(m1,Nat)) A(n1|@(n1,Nat)) A(m2|@(m2,Nat)) A(n2|@(n2,Nat))
C(X(N<=(m1,n1),N<=(m2,n2)),N<=(N+(m1,m2),N+(n1,n2)))
```

*Let **m1**, **n1**, **m2**, **n2** be natural numbers. If **m1** <= **n1** and **m2** <= **n2**, then **m1** + **m2** <= **n1** + **n2**.*

2077(T) Belonging and N<=

```
A - + - - - - - - - + - - - + - - - + -
^ @     A - + - - - - + - - - + - - - +
m   ^ N ^ @     D
    m a n   ^ N   @     X
      t     n a     ^ ^   N     N
              t     m n   < ^ ^   =
                          = m n     ^ ^
                                    m n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) D(@(m,n),X(N<=(m,n),N(=(m,n))))
```

*Let **m**, **n** be natural numbers. **m** belongs to **n** if and only if **m** <= **n** and **m**, **n** are different.*

2097(T) Class difference between Nat and a natural number

```
A - + - - - - + - - - - - - + -
^ @     =
n   ^ N   \     K - - + - - - +
    n a     N ^ ^ X
      t     a n x   @     N
            t         ^ N < ^ ^
                      x a = n x
                        t
```

```
A(n|@(n,Nat)) =(\(Nat,n),{x|X(@(x,Nat),N<=(n,x))})
```

*Let **n** be a natural number. Then Nat**n** = {**x** | **x** $\in$ Nat $\wedge$ **n** <= **x**}.*

2080(T) Lemma for subpath

```
A - + - - - - - - - - + - - - - - - - - - + - - - - +
^ @     A - + - - - - - - - - + - - - - + - - - - - -
m   ^ N ^ @     A - + - - - - - - - - + - - - - + - -
    m a n   ^ N ^ @     A - + - - - - - - - - - - + -
      t     n a p   ^ ^ ^ @     C
              t     p m q   ^ ^   N         @
                            q n   < N     ^   N     ^
                                  = + ^ ^ m   + ^ ^ m
                                      p n     p q
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) A(p|@(p,m)) A(q|@(q,n))
C(N<=(N+(p,n),m),@(N+(p,q),m))
```

*Let **m**, **n** be natural numbers, **p** an element of **m**, and **q** an element of **n**. If **p** + **n** <= **m**, then **p** + **q** belongs to **m**.*

2201(R) Path definition

```
A - - - + - - + - - - - - - - - - - - + -
^ T D
F     P   X
      a ^   F   E - - + - - - + - - - - +
      t F   u ^ ^ X               =
      h     n F n   @     N         D   ^
            c         ^ N   =       o ^ n
                      n a   ^ 0   m F
                        t     n
```

```
A(F) D(Path(F),X(Func(F),E(n|X(@(n,Nat),N(=(n,0)))) =(Dom(F),n)))
```

*Let **F** be a class. **F** is a path if and only if Func(**F**) and there exists a not null natural number **n** such that Dom of **F** is equal to **n**.*

A graph function is a path if its domain is a not null natural number.

2202(R) Circ definition

```
A - - - + - - + - - + - - + - - +
^ T D
F     C   X
      i ^   P   =
      r F   a ^   V     V
      c     t F   a ^ 0 a ^ P
            h     l F   l F r D
                            e o ^
                              m F
```

```
A(F) D(Circ(F),X(Path(F),=(Val(F,0),Val(F,Pre(Dom(F))))))
```

*Let **F** be a class. **F** is a circuit (Circ of **F**) if and only if **F** is a path and Val(**F**, 0) = Val(**F**, Pre(Dom(**F**))).*

A circuit is a path having the same first and last value.

2205(R) SubPath definition

```
A - + - - - - - + - - - - - - - - - - - - + - - - - - - + - - + - - - - - -
^ P   A - + - - - + - - - - - + - - - - - - + - - - - - - - - - - + - - -
F a ^ ^ P   D
  t F G a ^   S     E - - + - - - - + - - - - - - - - - - - - - - - - + -
  h     t G   u ^ ^ ^ X                       A - + - - - - - + - - - - +
        h     b F G m   @       N             ^ @       =
              P           ^ D   < N       D   x   ^ D     V     V
              a           m o ^ = + ^ D   o ^     x o ^   a ^ ^ a ^ N
              t             m G     m o ^ m G       m F   l F x l G + ^ ^
              h                       m F                             m x
```

```
A(F|Path(F)) A(G|Path(G))
D(SubPath(F,G),E(m|X(@(m,Dom(G)),N<=(N+(m,Dom(F)),Dom(G)))) A(x|@(x,Dom(F)))
=(Val(F,x),Val(G,N+(m,x))))
```

*Let **F**, **G** be paths. **F** is a subpath of **G** (SubPath of **F**, **G**) if and only if there exists **m** with **m** in Dom(**G**) and **m** + Dom(**F**) <= Dom(**G**), such that for any **x** in Dom(**F**), Val(**F**, **x**) = Val(**G**, **m** + **x**).*

2207(T) Corollary of SubPath definition

```
A - + - - - - - - - - - - - - + - - - - - + - - - - - - - -
^ P   A - + - - - - + - - - - - - - - - - - - + - - - - - +
F a ^ ^ P   A - + - - - - - - - - - - + - - - - - - + - - -
  t F G a ^ ^ @       A - + - - - - - - - - - - - - + - -
  h     t G m   ^ D   ^ @       C
        h       m o ^ n   ^ D     N             @
                  m G     n o ^   < N       D     N     D
                            m F   = + ^ D   o ^   + ^ ^ o ^
                                      m o ^ m G     m n m G
                                        m F
```

```
A(F|Path(F)) A(G|Path(G)) A(m|@(m,Dom(G))) A(n|@(n,Dom(F)))
C(N<=(N+(m,Dom(F)),Dom(G)),@(N+(m,n),Dom(G)))
```

*Let **F**, **G** be paths, **m** an element of Dom(**G**), and **n** an element of Dom(**F**). If **m** + Dom(**F**) <= Dom(**G**), then **m** + **n** belongs to Dom(**G**).*

2208(T) SubPath is transitive

```
A - + - - - - - - - - - + - - - - - + -
^ P   A - + - - - - - - - + - + - - - -
F a ^ ^ P   A - + - - - - - - - - + - - +
  t F G a ^ ^ P   C
  h     t G H a ^   X             S
        h     t H     S     S     u ^ ^
              h       u ^ ^ u ^ ^ b F H
                      b F G b G H P
                      P     P     a
                      a     a     t
                      t     t     h
                      h     h
```

```
A(F|Path(F)) A(G|Path(G)) A(H|Path(H))
C(X(SubPath(F,G),SubPath(G,H)),SubPath(F,H))
```

*Let **F**, **G**, **H** be classes with Path(**F**), Path(**G**), and Path(**H**). If SubPath(**F**,**G**) and SubPath(**G**,**H**), then SubPath(**F**,**H**).*

2209(T) 0 belongs to all natural numbers except 0

```
A - - + - - - + - - - +
^ X             @
n   @     N       0 ^
      ^ N   =       n
      n a     ^ 0
        t     n
```

```
A(n|X(@(n,Nat),N(=(n,0)))) @(0,n)
```

2210(T) SubPath is reflexive

```
A - + - + +
^ P   S
F a ^ u ^ ^
  t F b F F
  h   P
      a
      t
      h
```

```
A(F|Path(F)) SubPath(F,F)
```

2214(T) SubPath is antisymmetric

```
A - + - - - - - - + - - - + - + -
^ P   A - + - - - - + - + - - - +
F a ^ ^ P   C
  t F G a ^   X             =
  h     t G     S     S       ^ ^
        h       u ^ ^ u ^ ^   F G
                b F G b G F
                P     P
                a     a
                t     t
                h     h
```

```
A(F|Path(F)) A(G|Path(G)) C(X(SubPath(F,G),SubPath(G,F)),=(F,G))
```

2215(T) A path is a set

```
A - + - +
^ P   S
F a ^   ^
  t F   F
  h
```

```
A(F|Path(F)) S(F)
```

2216(T) Reduction for N<=

```
A - + - - - - - - - + - - - - - - - + - - - -
^ @     A - + - - - - + - - - - - - - - - + -
m   ^ N ^ @     D
    m a n   ^ N   N     A - + - - - - + - - +
      t     n a   < ^ ^ ^ @     N
              t   = m n d   ^ N < N     N
                            d a = + ^ ^ + ^ ^
                              t     m d   n d
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) D(N<=(m,n),A(d|@(d,Nat)) N<=(N+(m,d),N+(n,d)))
```

*Let **m**, **n** be natural numbers. **m** <= **n** if and only if, for any natural number **d**, **m** + **d** <= **n** + **d**.*

2218(T) N- of Suc of n and 1 is n

```
A - + - - - - + - +
^ @     =
n   ^ N   N       ^
    n a   - S   1 n
      t     u ^
            c n
```

```
A(n|@(n,Nat)) =(N-(Suc(n),1),n)
```

*For any natural number **n**, Suc(**n**) – 1 = **n**.*

The proof of the following theorem is not based on the axiom of regularity, and the theorem shows that natural numbers are not self-contained. So, even if we give up the axiom of regularity, the natural numbers are not self-contained.

2219(T) No mutual belonging for natural numbers

```
A - + - - - - - - - - + - - - - +
^ @     A - + - - - - - + - - + -
m   ^ N ^ @     V
    m a n   ^ N   N       N
      t     n a     @       @
              t       ^ ^     ^ ^
                      m n     n m
```

A(m|@(m,Nat)) A(n|@(n,Nat)) V(N(@(m,n)),N(@(n,m)))

*If **m**, **n** are natural numbers, then **m** does not belong to **n** or **n** does not belong to **m**.*

2222(T) N<= explanation

```
A - + - - - - - - - + - - - - +
^ @     A - + - - - - + - - + -
m   ^ N ^ @     D
    m a n   ^ N   N     N
      t     n a   < ^ ^   @
              t   = m n     ^ ^
                            n m
```

A(m|@(m,Nat)) A(n|@(n,Nat)) D(N<=(m,n),N(@(n,m)))

*Let **m**, **n** be natural numbers. **m** <= **n** if and only if **n** does not belong to **m**.*

2223(F) PathInc definition (increment of a path)

```
A - + - - - - - - + - - - - - - - - - - - - - - + - - - - - - -
^ P   A - + - - - - + - - - - - - - - - - - - - - - - - - - + -
G a ^ ^ @     =
  t G m   ^ N   P     K - - - - - - - - - - - - - - + - - - - -
  h       m a   a ^ ^ ^ E - + - - - - - - - + - - - - - - + - -
            t   t G m p ^ @     E - - + - - - + - - - - - - - +
                h       d   ^ N ^ X               =
                I           d a y   S   @           ^ P
                n             t       ^   P     ^   p a N       ^
                c                     y   a ^ ^ G     i + ^ ^ y
                                          i d y       r   d m
                                          r
```

A(G|Path(G)) A(m|@(m,Nat))
=(PathInc(G,m),{p|E(d|@(d,Nat)) E(y|X(S(y),@(Pair(d,y),G)))
=(p,Pair(N+(d,m),y))})

2226(T) PathInc is a graph

```
A - + - - - - - - + -
^ P   A - + - - - - +
G a ^ ^ @     G
  t G m   ^ N r P
  h       m a a a ^ ^
            t p t G m
              h h
                I
                n
                c
```

```
A(G|Path(G)) A(m|@(m,Nat)) Graph(PathInc(G,m))
```

2230(T) Dom of PathInc

```
A - + - - - - - - - + - - - - - - + - - - - -
^ P   A - + - - - - - + - - - - - - - - - - - +
G a ^ ^ @     =
  t G s   ^ N   D       K - - - - - - - + - - -
  h       s a   o P     ^ E - + - - - - - - + -
            t   m a ^ ^ m ^ @       =
                  t G s   d   ^ D     ^ N
                  h             d o ^   m + ^ ^
                  I               m G       d s
                  n
                  c
```

```
A(G|Path(G)) A(s|@(s,Nat))
=(Dom(PathInc(G,s)),{m|E(d|@(d,Dom(G))) =(m,N+(d,s))})
```

*Let **G** be a path and **s** a natural number. Dom(PathInc(**G**, **s**)) is the class of those **m** for which there exists **d** in Dom(**G**), so that **m** = **d** + **s**.*

2232(T) Belonging of natural number and addition

```
A - + - - - - - - - - - - - - - - - - - + - - + - +
^ @     A - + - - - - - - - - - + - - - - - - - - + -
m   ^ N ^ @     A - + - - - + - - - + - - - - - - -
    m a n   ^ N ^ @     D
      t     n a d   ^ N   @     @
              t     d a     ^ ^   N     \
                      t     d n   + ^ ^   N     ^
                                    d m   + ^ ^ m
                                            m n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) A(d|@(d,Nat)) D(@(d,n),@(N+(d,m),\(N+(m,n),m)))
```

*Let **m**, **n**, **d** be natural numbers. **d** belongs to **n** if and only if **d** + **m** belongs to (**m** + **n**) \ **m** (where "\" is the class difference).*

2235(T) Belonging to \ of N+

```
A - + - - - - - - - - - - - - - - + - + - - - - - - - - - +
^ @     A - + - - - - - - - - - - - + - - - - + - - - - -
m   ^ N ^ @     A - + - - + - - - - - - - - - - - + - - -
    m a n   ^ N ^ S   D
      t     n a s   ^   @             E - + - - - - - + -
              t     s     ^ \         ^ @     =
                          s   N     ^ d   ^ ^   ^ N
                              + ^ ^ m     d n   s + ^ ^
                                m n                 d m
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) A(s|S(s)) D(@(s,\(N+(m,n),m)),E(d|@(d,n))
=(s,N+(d,m)))
```

*Let **m**, **n** be natural numbers, and **s** a set. **s** belongs to (**m** + **n**) \ **m** if and only if there exists **d** in **n** such that **s** = **d** + **m**.*

2236(T) Dom of PathInc

```
A - + - - - - - - - + - - - - - + -
^ P   A - + - - - - - + - - + - - - +
G a ^ ^ @     =
  t G m   ^ N   D       \
  h       m a   o P       N       ^
            t   m a ^ ^   + ^ D   m
                  t G m     m o ^
                  h           m G
                  I
                  n
                  c
```

```
A(G|Path(G)) A(m|@(m,Nat)) =(Dom(PathInc(G,m)),\(N+(m,Dom(G)),m))
```

*Let **G** be a path and **m** be a natural number. Dom(PathInc(**G**, **m**)) is equal to (**m** + Dom(**G**)) \ **m**.*

2241(F) PathInc is Func

```
A - + - - - - - - + -
^ P   A - + - - - - +
G a ^ ^ @     F
  t G m   ^ N u P
  h       m a n a ^ ^
            t c t G m
                h
                I
                n
                c
```

```
A(G|Path(G)) A(m|@(m,Nat)) Func(PathInc(G,m))
```

*If **G** is a path and **m** is a natural number, then PathInc(**G**, **m**) is a graph function.*

2244(T) PathInc explanation

```
A - + - - - - - - - - - - - - - - - - - - - - + - - - - - +
^ P   A - + - - - - - - - - - - - - - - + - - - + - - - - -
G a ^ ^ @     A - + - - - - - - - - - + - - - - - - - + - -
  t G m   ^ N ^ @     A - + - - - - - - - + - - - - - - - + -
  h       m a d   ^ N ^ S   D
            t     d a y   ^   @                 @
                    t     y     P         P       P     ^
                                a N     ^ a ^ ^   a ^ ^ G
                                i + ^ ^ y t G m   i d y
                                r   d m   h       r
                                          I
                                          n
                                          c
```

```
A(G|Path(G)) A(m|@(m,Nat)) A(d|@(d,Nat)) A(y|S(y))
D(@(Pair(N+(d,m),y),PathInc(G,m)),@(Pair(d,y),G))
```

*Let **G** be a path, **m**, **d** natural numbers, and **y** a set. The pair of **d** + **m**, **y** belongs to PathInc of **G**, **m** if and only if the pair of **d**, **y** belongs to **G**.*

2245(T) Dom and Dom of PathInc

```
A - + - - - - - - - - - - - - - + - - - - - - - + -
^ P   A - + - - - - - - - - - - - - - - - + - - - - +
G a ^ ^ @     A - + - - - + - - - - - + - - - - -
  t G m   ^ N ^ @     D
  h       m a d   ^ N   @       @
            t     d a     ^ D     N     D
                    t     d o ^   + ^ ^ o P
                            m G     d m m a ^ ^
                                          t G m
                                          h
                                          I
                                          n
                                          c
```

```
A(G|Path(G)) A(m|@(m,Nat)) A(d|@(d,Nat))
D(@(d,Dom(G)),@(N+(d,m),Dom(PathInc(G,m))))
```

*Let **G** be a path, and **m**, **d** be natural numbers. **d** belongs to Dom of **G** if and only if **d** + **m** belongs to Dom of PathInc(**G**, **m**).*

2246(T) Values of a path and its PathInc

```
A - + - - - - - - - - + - - + - - - + - - - -
^ P   A - + - - - - - - - - - - - - - + - - +
G a ^ ^ @     A - + - - - - - + - - - - - + -
  t G m   ^ N ^ @       =
  h       m a d   ^ D     V     V
            t     d o ^   a ^ ^ a P     N
                    m G   l G d l a ^ ^ + ^ ^
                                  t G m   d m
                                  h
                                  I
                                  n
                                  c
```

`A(G|Path(G)) A(m|@(m,Nat)) A(d|@(d,Dom(G))) =(Val(G,d),Val(PathInc(G,m),N+(d,m)))`

*Let **G** be a path, **m** a natural number, and **d** in Dom(**G**). Then Val(**G**, **d**) = Val(PathInc(**G**, **m**), **d** + **m**).*

2247(T) For natural numbers, n is included in m + n

```
A - + - - - - - - - - + -
^ @     A - + - - + - - +
m   ^ N ^ @     P
    m a n   ^ N a ^ N
      t     n a r n + ^ ^
              t t     m n
```

`A(m|@(m,Nat)) A(n|@(n,Nat)) Part(n,N+(m,n))`

*If **m**, **n** are natural numbers, then **n** is included in **m** + **n**.*

2251(T) Reduction for belonging

```
A - + - - - - - - - - - - - + - - - + - - - -
^ @     A - + - - - - - - - - + - - - - - + -
m   ^ N ^ @     A - + - - - - - - - - + - - +
    m a n   ^ N ^ @     D
      t     n a p   ^ N   @     @
              t     p a     ^ ^   N     N
                      t     m n   + ^ ^ + ^ ^
                                    m p   n p
```

`A(m|@(m,Nat)) A(n|@(n,Nat)) A(p|@(p,Nat)) D(@(m,n),@(N+(m,p),N+(n,p)))`

*Let **m**, **n**, **p** be natural numbers. **m** belongs to **n** if and only if **m** + **p** belongs to **n** + **p**.*

2252(T) Belonging to sum seen through difference

```
A - + - - - - - - - - - - - + - - - - - + - - - - + -
^ @     A - + - - - - - - - - - - - - - - + - - - - +
m   ^ N ^ @     A - - + - - - + - - + - - - - - + - -
    m a n   ^ N ^ X             D
      t     n a d   @     N       @         @
              t       ^ N < ^ ^     ^ N       N     ^
                      d a = m d     d + ^ ^   - ^ ^ n
                        t               m n     d m
```

`A(m|@(m,Nat)) A(n|@(n,Nat)) A(d|X(@(d,Nat),N<=(m,d))) D(@(d,N+(m,n)),@(N-(d,m),n))`

*Let **m, n, d** be natural numbers with **m** <= **d**. **d** belongs to **m** + **n** if and only if **d** – **m** belongs to **n**.*

2253(T) Belonging to sum explanation

```
A - + - - - - - - - - - - - - - + - - - - + - - + - - - - + -
^ @     A - + - - - - - - - - - - + - - - - - - - - - - - - +
m   ^ N ^ @     A - + - - - + - - - - - + - - - - + - - + - -
    m a n   ^ N ^ @     D
      t     n a d   ^ N   @         V
              t     d a     ^ N       @     X
                      t     d + ^ ^     ^ ^   N     @
                                m n     d m   < ^ ^   N     ^
                                              = m d   - ^ ^ n
                                                        d m
```

`A(m|@(m,Nat)) A(n|@(n,Nat)) A(d|@(d,Nat)) D(@(d,N+(m,n)),V(@(d,m),X(N<=(m,d),@(N-(d,m),n))))`

*Let **m, n, d** be natural numbers. **d** belongs to **m** + **n** if and only if **d** belongs to **m** or (**m** <= **d** and **d** – **m** belongs to **n**).*

2254(F) Concat definition (Concatenation of two paths)

```
A - + - - - - - + - - + - - - +
^ P   A - + - - - + - - - + - -
F a ^ ^ P   =
  t F G a ^   C     U
  h     t G   o ^ ^   ^ P
        h     n F G   F a ^ D
              c         t G o ^
              a         h   m F
              t         I
                        n
                        c
```

`A(F|Path(F)) A(G|Path(G)) =(Concat(F,G),U(F,PathInc(G,Dom(F))))`

*Let **F**, **G** be paths. Concat of **F**, **G** is the union of **F** and PathInc(**G**, Dom(**F**)).*

2255(T) m and the class difference of N+(m, n) and m are disjoint

```
A - + - - - - - - + - - + - +
^ @     A - + - - - - - - + -
m   ^ N ^ @     D
    m a n   ^ N i ^ \
      t     n a s m   N     ^
              t       + ^ ^ m
                        m n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) Dis(m,\(N+(m,n),m))
```

2256(T) Dom and Dom of PathInc are disjoint

```
A - + - - - - - + - - - - +
^ P   A - + - - - - - + - -
F a ^ ^ P   D
  t F G a ^ i D   D
  h     t G s o ^ o P
        h     m F m a ^ D
                    t G o ^
                    h   m F
                    I
                    n
                    c
```

```
A(F|Path(F)) A(G|Path(G)) Dis(Dom(F),Dom(PathInc(G,Dom(F))))
```

2257(T) Concat is Func

```
A - + - - - - - + -
^ P   A - + - - - +
F a ^ ^ P   F
  t F G a ^ u C
  h     t G n o ^ ^
        h   c n F G
              c
              a
              t
```

```
A(F|Path(F)) A(G|Path(G)) Func(Concat(F,G))
```

2260(T) Addition is union of class difference

```
A - + - - - - - - - + - - + - - + - +
^ @     A - + - - - - + - - - - - + -
m   ^ N ^ @     =
    m a n   ^ N   N     U
      t     n a   + ^ ^   ^ \
              t     m n   m   N     ^
                              + ^ ^ m
                                m n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) =(N+(m,n),U(m,\(N+(m,n),m)))
```

*If **m**, **n** are natural numbers, then **m** + **n** is equal to the union of **m** and (**m** + **n**) \ **m**.*

2261(T) Dom of Concat is the addition of the domains

```
A - + - - - - - - + - - - + - -
^ P   A - + - - - - + - - - - +
F a ^ ^ P   =
  t F G a ^   D       N
  h     t G   o C     + D   D
        h     m o ^ ^   o ^ o ^
                n F G   m F m G
                c
                a
                t
```

```
A(F|Path(F)) A(G|Path(G)) =(Dom(Concat(F,G)),N+(Dom(F),Dom(G)))
```

2264(T) Addition is null iff the numbers are null

```
A - + - - - - - - - - + - - - - + - - - -
^ @     A - + - - - - - + - - - - - - - + -
m   ^ N ^ @     D
    m a n   ^ N   =         X
      t     n a     N     0   =     =
              t     + ^ ^       ^ 0   ^ 0
                      m n       m     n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) D(=(N+(m,n),0),X(=(m,0),=(n,0)))
```

2265(T) The domain of a path is not null

```
A - + - - - + -
^ P   N
F a ^   =
  t F     D   0
  h       o ^
          m F
```

```
A(F|Path(F)) N(=(Dom(F),0))
```

2266(T) Concat is a path

```
A - + - - - - - + -
^ P   A - + - - - +
F a ^ ^ P   P
  t F G a ^ a C
  h     t G t o ^ ^
        h   h n F G
              c
              a
              t
```

```
A(F|Path(F)) A(G|Path(G)) Path(Concat(F,G))
```

2267(T) Val of Concat first

```
A - + - - - - - - - + - - - + - - - + -
^ P   A - + - - - - - - - - - + - - - -
F a ^ ^ P   A - + - - - - - - - + - - +
  t F G a ^ ^ @       =
  h     t G d   ^ D     V         V
        h       d o ^   a C     ^ a ^ ^
                  m F   l o ^ ^ d l F d
                          n F G
                          c
                          a
                          t
```

`A(F|Path(F)) A(G|Path(G)) A(d|@(d,Dom(F))) =(Val(Concat(F,G),d),Val(F,d))`

*Let **F**, **G** be paths. If **d** belongs to Dom of **F**, then Val(Concat(**F**, **G**), **d**) is equal to Val(**F**, **d**).*

2269(T) Val of Concat second

```
A - + - - - - - - - - - - - + - - - - + - - -
^ P   A - + - - - - + - - - - + - - - - - + -
F a ^ ^ P   A - + - - - - - - - - + - - - - +
  t F G a ^ ^ @       =
  h     t G d   ^ D     V               V
        h       d o ^   a C     N       a ^ ^
                  m G   l o ^ ^ + ^ D   l G d
                          n F G   d o ^
                          c         m F
                          a
                          t
```

`A(F|Path(F)) A(G|Path(G)) A(d|@(d,Dom(G)))`
`=(Val(Concat(F,G),N+(d,Dom(F))),Val(G,d))`

*Let **F**, **G** be paths. If **d** belongs to Dom of **G**, then Val(Concat(**F**, **G**), **d** + Dom(**F**)) is equal to Val(**G**, **d**).*

2270(T) The first path of Concat is a subpath of Concat

```
A - + - - - - + - + -
^ P   A - + - - - - +
F a ^ ^ P   S
  t F G a ^ u ^ C
  h     t G b F o ^ ^
        h   P   n F G
            a   c
            t   a
            h   t
```

`A(F|Path(F)) A(G|Path(G)) SubPath(F,Concat(F,G))`

2271(T) Belonging to an addition with a not null

```
A - + - - - - - - - - - - - - + - + -
^ @     A - + - - - - + - - - - - - +
m   ^ N ^ @     C
    m a n   ^ N   N       @
      t     n a     =       ^ N
              t       ^ 0   m + ^ ^
                      n         m n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) C(N(=(n,0)),@(m,N+(m,n)))
```

*Let **m**, **n** be natural numbers. If **n** is not null, then **m** belongs to **m** + **n**.*

2272(T) The second path of Concat is a subpath of Concat

```
A - + - - - - - - - + -
^ P   A - + - + - - +
F a ^ ^ P   S
  t F G a ^ u ^ C
  h     t G b G o ^ ^
        h   P   n F G
            a   c
            t   a
            h   t
```

```
A(F|Path(F)) A(G|Path(G)) SubPath(G,Concat(F,G))
```

2273(T) If X is included in Dom then Dom of DomInt is X

```
A - + - - - - + - - - + - -
^ G   A - + - - - - - - + +
G r ^ ^ P       =
  a G X a ^ D     D       ^
  p     r X o ^   o D     X
  h     t   m G   m o ^ ^
                    m G X
                    I
                    n
                    t
```

```
A(G|Graph(G)) A(X|Part(X,Dom(G))) =(Dom(DomInt(G,X)),X)
```

*Let **G** be a graph and **X** be a part of Dom(**G**). Then Dom of DomInt(**G**, **X**) is equal to **X**.*

2274(F) PathDec definition (decreasing path)

```
A - + - - - - + - - + - - - - - - - - - - - - - - - - - - + - - - - - - -
^ P   A - + - - - - - + - - - - - - - + - - - - - - - - - - - - - - - + -
H a ^ ^ @       =
  t H m   ^ D     P     K - - - - - - - - - - - - - - - - - - + - - - - -
  h       m o ^   a ^ ^ ^ E - - + - - - + - - - - - - + - - - - - - + - -
            m H   t H m p ^ X             E - - + - - - + - - - - - - - +
                  h       d   @     N     ^ X               =
                  D             ^ N < ^ ^ y   S   @           ^ P
                  e             d a = m d       ^   P     ^   p a N     ^
                  c               t             y   a ^ ^ H     i - ^ ^ y
                                                    i d y       r   d m
                                                    r
```

```
A(H|Path(H)) A(m|@(m,Dom(H))) =(PathDec(H,m),{p|E(d|X(@(d,Nat),N<=(m,d)))
E(y|X(S(y),@(Pair(d,y),H))) =(p,Pair(N-(d,m),y))})
```

*Let **H** be a path and **m** an element of Dom(**H**). PathDec(**H**, **m**) is equal to the class of those **p** for which there exists a natural number **d** with **m** <= **d**, and there exists a set **y** with Pair(**d**, **y**) belonging to **H**, such that **p** = Pair(**d** - **m**, **y**).*

2277(T) PathDec is a graph

```
A - + - - - - + - - + -
^ P   A - + - - - - - +
H a ^ ^ @       G
  t H m   ^ D   r P
  h       m o ^ a a ^ ^
            m H p t H m
                h h
                  D
                  e
                  c
```

```
A(H|Path(H)) A(m|@(m,Dom(H))) Graph(PathDec(H,m))
```

2280(T) Belonging to N- explanation

```
A - + - - - - + - - - - - - - - + - - - - - +
^ @     A - + - - - - - - - - - - + - - - + -
m   ^ N ^ @     A - + - - - + - - - - - + - -
    m a n   ^ ^ ^ @     D
      t     n m d   ^ N   @         @
                    d a     ^ N       N     ^
                      t     d - ^ ^   + ^ ^ m
                                m n     d n
```

```
A(m|@(m,Nat)) A(n|@(n,m)) A(d|@(d,Nat)) D(@(d,N-(m,n)),@(N+(d,n),m))
```

*Let **m**, **d** be natural numbers and **n** in **m**. **d** belongs to **m** – **n** if and only if **d** + **n** belongs to **m**.*

2281(T) A path is a graph

```
A - + - +
^ P   G
F a ^ r ^
  t F a F
  h   p
      h
```

```
A(F|Path(F)) Graph(F)
```

2282(T) PathDec explanation

```
A - + - - - - + - - - - - - - - - - - - - + - - - - - - - - - - - +
^ P   A - + - - - - - - - - - - - - - - - - - + - - - - - - - - + - -
H a ^ ^ @       A - + - - - - - - - + - - - - - - - + - - - - + - - -
  t H m   ^ D   ^ S   A - + - - - - + - - - - - - - - - - - - - - + -
  h       m o ^ x   ^ ^ S   D
            m H     x y   ^   @               X
                          y     P     P         @     @
                                a ^ ^ a ^ ^     ^ N   P         ^
                                i x y t H m     x a   a N     ^ H
                                r     h           t   i + ^ ^ y
                                      D               r   x m
                                      e
                                      c
```

```
A(H|Path(H)) A(m|@(m,Dom(H))) A(x|S(x)) A(y|S(y))
D(@(Pair(x,y),PathDec(H,m)),X(@(x,Nat),@(Pair(N+(x,m),y),H)))
```

*Let **H** be a path, **m** an element of Dom(**H**), and **x**, **y** sets. Pair(**x**, **y**) belongs to PathDec(**H**, **m**) if and only if **x** is a natural number and Pair(**x** + **m**, **y**) belongs to **H**.*

2285(T) Dom of PathDec

```
A - + - - - - + - - - + - - - + -
^ P   A - + - - - - - - - + - - - - +
H a ^ ^ @       =
  t H m   ^ D     D       N
  h       m o ^   o P     - D   ^
            m H   m a ^ ^   o ^ m
                    t H m   m H
                    h
                    D
                    e
                    c
```

```
A(H|Path(H)) A(m|@(m,Dom(H))) =(Dom(PathDec(H,m)),N-(Dom(H),m))
```

*Let **H** be a path and **m** an element of Dom(**H**). Then Dom(PathDec(**H**, **m**)) is equal to Dom(**H**) – **m**.*

2287(T) PathDec is Func

```
A - + - - - - + - - + -
^ P   A - + - - - - - +
H a ^ ^ @       F
  t H m   ^ D   u P
  h       m o ^ n a ^ ^
            m H c t H m
                  h
                  D
                  e
                  c
```

```
A(H|Path(H)) A(m|@(m,Dom(H))) Func(PathDec(H,m))
```

2288(T) Val of PathDec

```
A - + - - - - + - - - - - + - - - - + - - - + - - -
^ P   A - + - - - - - - - - - + - - - - + - - - - - +
H a ^ ^ @       A - + - - - - - - - - - - + - - - + -
  t H m   ^ D   ^ @           =
  h       m o ^ x   ^ D           V         V
            m H     x o P         a P     ^ a ^ N
                      m a ^ ^     l a ^ ^ x l H + ^ ^
                        t H m       t H m         x m
                        h           h
                        D           D
                        e           e
                        c           c
```

```
A(H|Path(H)) A(m|@(m,Dom(H))) A(x|@(x,Dom(PathDec(H,m))))
=(Val(PathDec(H,m),x),Val(H,N+(x,m)))
```

*Let **H** be a path, **m** an element of Dom(**H**), and **x** an element of Dom(PathDec(**H**, **m**)). Then Val(PathDec(**H**, **m**), **x**) is equal to Val(**H**, **x** + **m**).*

2289(T) Belonging implies N- is not null

```
A - + - - - - + - - - + - -
^ @     A - + - - - - - + -
m   ^ N ^ @     N
    m a n   ^ ^   =
      t     n m     N     0
                    - ^ ^
                      m n
```

```
A(m|@(m,Nat)) A(n|@(n,m)) N(=(N-(m,n),0))
```

*If **m** is a natural number and **n** is an element of **m**, then **m** – **n** is not null.*

2290(T) PathDec is Path

```
A - + - - - - + - - + -
^ P   A - + - - - - - +
H a ^ ^ @       P
  t H m   ^ D   a P
  h       m o ^ t a ^ ^
            m H h t H m
                  h
                  D
                  e
                  c
```

```
A(H|Path(H)) A(m|@(m,Dom(H))) Path(PathDec(H,m))
```

2291(T) PathDec is a subpath

```
A - + - - - - + - - + - +
^ P   A - + - - - - - + -
H a ^ ^ @       S
  t H m   ^ D   u P     ^
  h       m o ^ b a ^ ^ H
            m H P t H m
                a h
                t D
                h e
                  c
```

```
A(H|Path(H)) A(m|@(m,Dom(H))) SubPath(PathDec(H,m),H)
```

2292(T) N<= explanation with intersection

```
A - + - - - - - - - + - - - + - +
^ @     A - + - - - - + - - - + -
m   ^ N ^ @     D
    m a n   ^ N   N     =
      t     n a   < ^ ^   I     ^
              t   = m n     ^ ^ m
                            m n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) D(N<=(m,n),=(I(m,n),m))
```

*Let **m, n** be natural numbers. **m** <= **n** if and only if the intersection of **m, n** is equal to **m**.*

2293(T) Dom of DomInt of a path and an element of its Dom

```
A - + - - - - + - - - + - -
^ P   A - + - - - - - - + +
F a ^ ^ @       =
  t F m   ^ D     D       ^
  h       m o ^   o D     m
            m F   m o ^ ^
                    m F m
                    I
                    n
                    t
```

```
A(F|Path(F)) A(m|@(m,Dom(F))) =(Dom(DomInt(F,m)),m)
```

*Let **F** be a path and **m** an element of Dom(**F**). Then Dom(DomInt(**F**, **m**)) is equal to **m**.*

2294(T) DomInt is a path

```
A - + - - - - - + - - - - - - - + -
^ P   A - - + - - - - + - - - - +
H a ^ ^ X                 P
  t H m   @       N       a D
  h         ^ D     =     t o ^ ^
            m o ^     ^ 0 h m H m
              m H     m     I
                            n
                            t
```

```
A(H|Path(H)) A(m|X(@(m,Dom(H)),N(=(m,0)))) Path(DomInt(H,m))
```

*Let **H** be a path and **m** a not null element of Dom(**H**). Then DomInt(**H**, **m**) is a path.*

2295(T) Val of DomInt

```
A - + - - - - - - - + - - - - + - - - + -
^ F   A - - - - - - - + - - - - + - - - -
F u ^ ^ T A - + - - - - - - - - - + - - +
  n F X   ^ @           =
  c       x   ^ D         V         V
              x o D       a D     ^ a ^ ^
                m o ^ ^   l o ^ ^ x l F x
                  m F X     m F X
                  I         I
                  n         n
                  t         t
```

```
A(F|Func(F)) A(X) A(x|@(x,Dom(DomInt(F,X)))) =(Val(DomInt(F,X),x),Val(F,x))
```

*Let **F** be a graph function, **X** a class, and **x** an element of Dom(DomInt(**F**, **X**)). Then Val(DomInt(**F**, **X**), **x**) is equal to Val(**F**, **x**).*

2296(T) Val of DomInt of a path and an element of its domain

```
A - + - - - - - + - - - - - - - - - - - + - - - + -
^ P   A - - + - - - - + - - - - + - - - - + - - - -
H a ^ ^ X                 A - + - - - - - - + - - +
  t H m   @       N       ^ @     =
  h         ^ D     =     x   ^ ^   V         V
            m o ^     ^ 0     x m   a D     ^ a ^ ^
              m H     m             l o ^ ^ x l H x
                                      m H m
                                      I
                                      n
                                      t
```

```
A(H|Path(H)) A(m|X(@(m,Dom(H)),N(=(m,0)))) A(x|@(x,m))
=(Val(DomInt(H,m),x),Val(H,x))
```

*Let **H** be a path, **m** a not null element of Dom(**H**), and **x** an element of **m**. Then Val(DomInt(**H**, **m**), **x**) is equal to Val(**H**, **x**).*

2297(T) DomInt is a subpath

```
A - + - - - - - + - - - - - - + - +
^ P   A - - + - - - - + - - - - + -
H a ^ ^ X                 S
  t H m   @       N       u D     ^
  h         ^ D     =     b o ^ ^ H
            m o ^     ^ 0 P m H m
              m H     m   a I
                          t n
                          h t
```

```
A(H|Path(H)) A(m|X(@(m,Dom(H)),N(=(m,0)))) SubPath(DomInt(H,m),H)
```

2298(T) Dom of Concat of DomInt and PathDec is Dom

```
A - + - - - - - + - - - - - - - - + - - + - - +
^ P   A - - + - - - - + - - - - - - + - - + - -
H a ^ ^ X                 =
  t H m   @       N         D               D
  h         ^ D     =       o C             o ^
            m o ^     ^ 0   m o D     P     m H
              m H     m       n o ^ ^ a ^ ^
                              c m H m t H m
                              a I     h
                              t n     D
                                t     e
                                      c
```

```
A(H|Path(H)) A(m|X(@(m,Dom(H)),N(=(m,0))))
=(Dom(Concat(DomInt(H,m),PathDec(H,m))),Dom(H))
```

2303(T) Value of PathInc

```
A - + - - - - - - - - - - - + - - - - + - - - + - - -
^ P   A - + - - - - - - + - - + - - - - + - - - - - +
H a ^ ^ @     A - + - - - - - - - - - - - + - - - + -
  t H m   ^ N ^ @               =
  h       m a x   ^ \             V         V
            t     x   N       ^   a P     ^ a ^ N
                      + ^ D   m   l a ^ ^ x l H - ^ ^
                        m o ^     t H m         x m
                          m H       h
                                    I
                                    n
                                    c
```

```
A(H|Path(H)) A(m|@(m,Nat)) A(x|@(x,\(N+(m,Dom(H)),m)))
=(Val(PathInc(H,m),x),Val(H,N-(x,m)))
```

*Let **H** be a path, **m** a natural number, and **x** an element of (**m** + Dom(**H**)) \ **m**. Then Val(PathInc(**H**, **m**), **x**) is equal to Val(**H**, **x** – **m**).*

2304(T) Val of Concat of DomInt and PathDec

```
A - + - - - - - + - - - - - - - - - - - - + - - + - - - + -
^ P   A - - + - - - - + - - - - + - - - - - + - - + - - - -
H a ^ ^ X                 A - + - - - - - - - - - - + - - +
  t H m   @       N       ^ @     =
  h         ^ D     =     x   ^ ^   V                 V
            m o ^     ^ 0     x m   a C             ^ a ^ ^
              m H     m             l o D     P     x l H x
                                      n o ^ ^ a ^ ^
                                      c m H m t H m
                                      a I     h
                                      t n     D
                                        t     e
                                              c
```

```
A(H|Path(H)) A(m|X(@(m,Dom(H)),N(=(m,0)))) A(x|@(x,m))
=(Val(Concat(DomInt(H,m),PathDec(H,m)),x),Val(H,x))
```

*Let **H** be a path, **m** a not null element of Dom(**H**), and **x** an element of **m**. Then Val(Concat(DomInt(**H**, **m**), PathDec(**H**, **m**)), **x**) is equal to Val(**H**, **x**).*

2307(T) Belonging implies belonging to sum

```
A - + - - - - - - - - - - - - + - - - + -
^ @     A - + - - - - - - - - - - - - - +
m   ^ N ^ @     A - + - - - + - - + - - -
    m a n   ^ N ^ @     C
      t     n a x   ^ N   @     @
              t     x a     ^ ^   ^ N
                      t     x m   x + ^ ^
                                      m n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) A(x|@(x,Nat)) C(@(x,m),@(x,N+(m,n)))
```

*Let **m**, **n**, **x** be natural numbers. If **x** belongs to **m**, then **x** belongs to **m** + **n**.*

2313(T) Class difference of a natural number is included in Nat

```
A - + - - - - - + - -
^ @     A - - - - + -
m   ^ N ^ T P
    m a x   a \     N
      t     r   ^ ^ a
            t   m x t
```

```
A(m|@(m,Nat)) A(x) Part(\(m,x),Nat)
```

*Let **m** be a natural number and **x** be a class. Then **m** \ **x** is included in Nat.*

2319(T) Val of Concat second

```
A - + - - - - - - - - - + - - - + - - - + - - - - - - - +
^ P   A - + - - - - - - - - + - - - - - - + - - + - - - -
F a ^ ^ P   A - + - - - - - - - - - - - - - + - - - + - -
  t F G a ^ ^ @                   =
  h     t G x   ^ \                 V         V
        h       x   N         D     a C     ^ a ^ N
                    + D   D   o ^   l o ^ ^ x l G - ^ D
                      o ^ o ^ m F     n F G         x o ^
                      m F m G         c             m F
                                      a
                                      t
```

```
A(F|Path(F)) A(G|Path(G)) A(x|@(x,\(N+(Dom(F),Dom(G)),Dom(F))))
=(Val(Concat(F,G),x),Val(G,N-(x,Dom(F))))
```

*Let **F**, **G** be paths and **x** an element of (Dom(**F**) + Dom(**G**)) \ Dom(**F**). Then Val(Concat(**F**, **G**), **x**) is equal to Val(**G**, **x** – Dom(**F**)).*

2322(T) Reduction for belonging with N-

```
A - + - - - - - - - - + - - - - - - - - - - - + - - - - - + -
^ @     A - - + - - + - - - - - - - + - - - - - - - - + - - +
n   ^ N ^ X             A - - + - - - + - - + - - - + - - - -
    n a p   @     N     ^ X             D
      t       ^ N < ^ ^ m   @     N       @     @
              p a = p n       ^ N < ^ ^     ^ ^   N     N
                t             m a = p m     m n   - ^ ^ - ^ ^
                                t                   m p   n p
```

```
A(n|@(n,Nat)) A(p|X(@(p,Nat),N<=(p,n))) A(m|X(@(m,Nat),N<=(p,m))) D(@(m,n),@(N-
(m,p),N-(n,p)))
```

*Let **n**, **p**, **m** be natural numbers with **p** <= **n** and **p** <= **m**. **m** belongs to **n** if and only if **m** – **p** belongs to **n** – **p**.*

2324(T) N+, N- associativity

```
A - + - - - - - - - - - - - - + - - - - - - + - - - - + -
^ @     A - + - - - - - - - - - + - - - - - - + - - - - +
m   ^ N ^ @     A - - + - - - - - + - - - + - - - + - - -
    m a n   ^ N ^ X                 =
      t     n a x   @     N           N         N
              t       ^ N < N     ^   - N     ^ - ^ N
                      x a = + ^ ^ x     - ^ ^ n   x + ^ ^
                        t     m n         x m         m n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) A(x|X(@(x,Nat),N<=(N+(m,n),x)))
=(N-(N-(x,m),n),N-(x,N+(m,n))
```

*Let **m**, **n**, **x** be natural numbers with **m** + **n** <= **x**. Then (**x** – **m**) – **n** is equal to **x** – (**m** + **n**).*

2329(T) Addition, union and class difference for natural numbers

```
A - + - - - - - - - - - - - - + - - - - + - - + - + - - - + - - - + -
^ @     A - + - - - - - - - - - + - - - - - - - + - - - - - + - - - +
m   ^ N ^ @     A - + - - - - - - + - - - - - - - - - - - - - + - - -
    m a n   ^ N ^ @     =
      t     n a p   ^ N   N         U
              t     p a   + N     ^   U             \
                      t     + ^ ^ p     ^ \           N         N
                              m n       m   N     ^   + N     ^ + ^ ^
                                            + ^ ^ m     + ^ ^ p   m n
                                              m n         m n
```

A(m|@(m,Nat)) A(n|@(n,Nat)) A(p|@(p,Nat))
=(N+(N+(m,n),p),U(U(m,\(N+(m,n),m)),\(N+(N+(m,n),p),N+(m,n))))

*Let **m**, **n**, **p** be natural numbers. Then*
*(**m** + **n**) + **p** = (**m** U ((**m** + **n**) \ **m**)) U (((**m** + **n**) + **p**) \ (**m** + **n**)).*

2330(T) Val of Concat

```
A - + - - - - - - - - - - - - + - - - - - - - - + - - - - - + - - - -
^ P   A - + - - - - - - - - - - - + - - - - - - - + - - - - - - + - -
F a ^ ^ P   A - + - - - - - - - - - - + - - - - - - + - - - - - - + -
  t F G a ^ ^ P   A - + - - - - - - - - - - - - - - - + - - - - - - +
  h     t G H a ^ ^ @                   =
        h     t H x   ^ N                 V             V
              h       x + N         D     a C         ^ a C         ^
                          + D   D   o ^   l o C     ^ x l o ^ C     x
                            o ^ o ^ m H     n o ^ ^ H     n F o ^ ^
                            m F m G         c n F G       c   n G H
                                            a c           a   c
                                            t a           t   a
                                              t               t
```

A(F|Path(F)) A(G|Path(G)) A(H|Path(H)) A(x|@(x,N+(N+(Dom(F),Dom(G)),Dom(H))))
=(Val(Concat(Concat(F,G),H),x),Val(Concat(F,Concat(G,H)),x))

*Let **F**, **G**, **H** be paths and **x** an element of (Dom(**F**) + Dom(**G**)) + Dom(**H**).*
*Then Val(Concat(Concat(**F**, **G**), **H**), **x**) is equal to Val(Concat(**F**, Concat(**G**, **H**)), **x**).*

2331(T) Concat is associative

```
A - + - - - - - - - - - + - - - + - - -
^ P   A - + - - - - - - - + - - - - + -
F a ^ ^ P   A - + - - - - - + - - - - +
  t F G a ^ ^ P   =
  h     t G H a ^   C         C
        h     t H   o C     ^ o ^ C
              h     n o ^ ^ H n F o ^ ^
                    c n F G   c   n G H
                    a c       a   c
                    t a       t   a
                      t           t
```

A(F|Path(F)) A(G|Path(G)) A(H|Path(H))
=(Concat(Concat(F,G),H),Concat(F,Concat(G,H)))

2332(R) SPath definition (simple path)

```
A - - - + - - + - +
^ T D
F     S   X
      P ^   P   S
      a F   a ^ F ^
      t     t F u F
      h     h   n
                c
```

```
A(F) D(SPath(F),X(Path(F),SFunc(F)))
```

F *is SPath if and only if* ***F*** *is Path and* ***F*** *is SFunc.*

2333(T) SPath corollary

```
A - - - + - - + - - - - - - - - - - - - + -
^ T D
F     S   X
      P ^   S   E - - + - - - + - - - - +
      a F   F ^ ^ X               =
      t     u F n   @     N         D   ^
      h     n         ^ N   =       o ^ n
            c         n a     ^ 0   m F
                        t     n
```

```
A(F) D(SPath(F),X(SFunc(F),E(n|X(@(n,Nat),N(=(n,0)))) =(Dom(F),n)))
```

Let ***F*** *be a class.* ***F*** *is SPath if and only if* ***F*** *is SFunc and there exists a not null natural number* ***n****, such that Dom(****F****) =* ***n****.*

2335(T) Concat of two SPath with disjoint Ran is SPath

```
A - + - - - - - - + - - - - + -
^ S   A - + - - - - - + - - - +
F P ^ ^ S   C
  a F G P ^   D         S
  t     a G   i R   R   P C
  h     t     s a ^ a ^ a o ^ ^
        h       n F n G t n F G
                        h c
                          a
                          t
```

```
A(F|SPath(F)) A(G|SPath(G)) C(Dis(Ran(F),Ran(G)),SPath(Concat(F,G)))
```

2336(T) m belongs to n iff Suc(**m**) <= **n**

```
A - + - - - - - - - + - - - + -
^ @     A - + - - - - + - - - +
m   ^ N ^ @     D
    m a n   ^ N   @     N
      t     n a     ^ ^ < S   ^
              t     m n = u ^ n
                          c m
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) D(@(m,n),N<=(Suc(m),n))
```

*Let **m**, **n** be natural numbers. **m** belongs to **n** if and only if Suc(**m**) <= **n**.*

2341(T) n belongs to m implies N- of m, Suc(n) belongs to m

```
A - + - - - - + - - + - - +
^ @     A - + - - - - - + -
m   ^ N ^ @     @
    m a n   ^ ^   N       ^
      t     n m   - ^ S   m
                    m u ^
                      c n
```

```
A(m|@(m,Nat)) A(n|@(n,m)) @(N-(m,Suc(n)),m)
```

*Let **m** be a natural number and **n** be an element of **m**. Then **m** – Suc(**n**) belongs to **m**.*

2342(F) Permut definition (permutation of m objects)

```
A - - + - - - + - - - + - - - - - - - - + - - - +
^ X                 =
m   @     N           P   K - - + - - - + - - - + -
      ^ N   =         e ^ ^ X
      m a     ^ 0     r m F   S   X
        t     m       m       F ^   =       =
                      u       u F     D   ^   R   ^
                      t       n       o ^ m   a ^ m
                              c       m F     n F
```

```
A(m|X(@(m,Nat),N(=(m,0))))
=(Permut(m),{F|X(SFunc(F),X(=(Dom(F),m),=(Ran(F),m)))})
```

*Let **m** be a not null natural number. Permut(**m**) is the class of those **F** for which **F** is SFunc, Dom(**F**) = **m**, and Ran(**F**) = **m**.*

2343(F) LastPerm definition (last permutation of m objects)

```
A - - + - - - + - - - + - - - - + - - - - - + - -
^ X                 =
m   @     N           L   K - - - - - + - - - - - -
      ^ N   =         a ^ ^ E - + - - - - + - - - +
      m a     ^ 0     s m p ^ @     =
        t     m       t     x   ^ ^   ^ P
                      P         x m   p a ^ N
                      e                 i x - ^ S
                      r                 r     m u ^
                      m                         c x
```

```
A(m|X(@(m,Nat),N(=(m,0))))
=(LastPerm(m),{p|E(x|@(x,m)) =(p,Pair(x,N-(m,Suc(x))))})
```

*Let **m** be a not null natural number. LastPerm(**m**) is the class of those **p** for which there exists **x** in **m** such that **p** is equal to Pair(**x**, **m** – Suc(**x**)).*

2346(T) LastPerm explanation

```
A - - + - - - + - - - - - - - - - - - - - + - - - + - - - + - -
^ X             A - + - - - - - - + - - - - - + - - - - - - +
m   @     N     ^ S   A - + - - - - + - - - - - - - - + - - - -
      ^ N   =     x   ^ ^ S   D
      m a     ^ 0     x y   ^   @           X
        t     m             y     P     L     @     =
                                  a ^ ^ a ^     ^ ^   ^ N
                                  i x y s m     x m   y - ^ S
                                  r     t               m u ^
                                        P                 c x
                                        e
                                        r
                                        m
```

```
A(m|X(@(m,Nat),N(=(m,0)))) A(x|S(x)) A(y|S(y))
D(@(Pair(x,y),LastPerm(m)),X(@(x,m),=(y,N-(m,Suc(x)))))
```

*Let **m** be a not null natural number and **x**, **y** be sets. Pair(**x**, **y**) belongs to LastPerm(**m**) if and only if **x** belongs to **m** and **y** is equal to **m** – Suc(**x**).*

2347(T) LastPerm is a graph

```
A - - + - - - + - - - +
^ X             G
m   @     N     r L
      ^ N   =   a a ^
      m a     ^ 0 p s m
        t     m   h t
                    P
                    e
                    r
                    m
```

```
A(m|X(@(m,Nat),N(=(m,0)))) Graph(LastPerm(m))
```

2350(T) Dom of LastPerm of m is m

```
A - - + - - - + - - - - + +
^ X             =
m   @     N       D     ^
      ^ N   =     o L   m
      m a     ^ 0 m a ^
        t     m     s m
                    t
                    P
                    e
                    r
                    m
```

```
A(m|X(@(m,Nat),N(=(m,0)))) =(Dom(LastPerm(m)),m)
```

2355(T) Suc(m) = m + 1

```
A - + - - - + - + -
^ @     =
m   ^ N   S   N
    m a   u ^ + ^ 1
      t   c m   m
```

`A(m|@(m,Nat)) =(Suc(m),N+(m,1))`

*Let **m** be a natural number. Then Suc(**m**) = **m** + 1.*

2357(T) Difference of succesors

```
A - + - - - - - - - - - + - - - + - - - + -
^ @     A - - + - - + - - - - - - + - - +
m   ^ N ^ X             =
    m a n   @     N       N         N
      t       ^ N < ^ ^   - S   S   - ^ ^
              n a = n m     u ^ u ^   m n
                t           c m c n
```

`A(m|@(m,Nat)) A(n|X(@(n,Nat),N<=(n,m))) =(N-(Suc(m),Suc(n)),N-(m,n))`

*Let **m**, **n** be natural numbers with **n** <= **m**. Then Suc(**m**) – Suc(**n**) = **m** – **n**.*

2361(T) LastPerm is Func

```
A - - + - - - + - - - +
^ X               F
m   @     N       u L
      ^ N   =     n a ^
      m a     ^ 0 c s m
        t     m     t
                    P
                    e
                    r
                    m
```

`A(m|X(@(m,Nat),N(=(m,0)))) Func(LastPerm(m))`

2362(T) Val of LastPerm

```
A - - + - - - + - - - - + - - - + - - + - -
^ X               A - + - - - - - + - - - +
m   @     N       ^ @     =
      ^ N   =     x   ^ ^   V       N
      m a     ^ 0     x m   a L   ^ - ^ S
        t     m             l a ^ x   m u ^
                              s m       c x
                              t
                              P
                              e
                              r
                              m
```

`A(m|X(@(m,Nat),N(=(m,0)))) A(x|@(x,m)) =(Val(LastPerm(m),x),N-(m,Suc(x)))`

*Let **m** be a not null natural number and **x** an element of **m**. Then Val(LastPerm(**m**), **x**) = **m** - Suc(**x**).*

2365(T) LastPerm is SFunc

```
A - - + - - - + - - - +
^ X             S
m   @     N     F L
      ^ N   =   u a ^
      m a     ^ 0 n s m
        t     m   c t
                    P
                    e
                    r
                    m
```

```
A(m|X(@(m,Nat),N(=(m,0)))) SFunc(LastPerm(m))
```

2369(T) LastPerm is equal to its inverse

```
A - - + - - - + - - - + - - +
^ X             =
m   @     N       L   I
      ^ N   =     a ^ n L
      m a     ^ 0 s m v a ^
        t     m   t     s m
                  P     t
                  e     P
                  r     e
                  m     r
                        m
```

```
A(m|X(@(m,Nat),N(=(m,0)))) =(LastPerm(m),Inv(LastPerm(m)))
```

2370(T) Ran of LastPerm of m is m

```
A - - + - - - + - - - - + +
^ X             =
m   @     N       R     ^
      ^ N   =     a L   m
      m a     ^ 0 n a ^
        t     m     s m
                    t
                    P
                    e
                    r
                    m
```

```
A(m|X(@(m,Nat),N(=(m,0)))) =(Ran(LastPerm(m)),m)
```

2385(T) Axiom of regularity corollary

```
A - - + - - - - + - - + - -
^ N       E - + - - - - + -
X   =     ^ @      =
      ^ 0 x   ^ ^   I     0
      X       x X     ^ ^
                      X x
```

`A(X|N(=(X,0))) E(x|@(x,X)) =(I(X,x),0)`

2386(T) No mutual belonging

```
A - - - - - - + - - - +
^ T A - - - - - + - + -
a   ^ T N
    b     X
            @     @
              ^ ^   ^ ^
              a b   b a
```

`A(a) A(b) N(X(@(a,b),@(b,a)))`

2501(F) PNat definition (Pairs of natural numbers)

```
=
  P C
  N a N N
  a r a a
  t t t t
```

`=(PNat,Cart(Nat,Nat))`

Division of natural numbers

We will first show that the set of natural numbers is well ordered. That means any nonempty subset of the set of natural numbers has a least element. For this, we will first prove the existence and then the uniqueness of such an element. We will name this element NMin (from minimum). We will then prove the existence theorem of the division of natural numbers, that is, the theorem that shows us the existence and uniqueness of the quotient (NDiv) and the remainder (NMod). Since the existence and uniqueness of NDiv and NMod must be proved simultaneously, we will use their ordered pair and name it NDivMod. We will demonstrate that NDivMod exists and is unique. NDiv will be the first and NMod the second coordinate of NDivMod.

2755(T) Nat is well ordered

```
A - - + - - - + - - - - + - - - + - - - -
^ X               E - + - - - - - - + -
M   P     N       ^ @     A - + - - - +
    a ^ N   =     m   ^ ^ ^ @     N
    r M a     ^ 0     m M n   ^ ^ < ^ ^
    t   t     M               n M = m n
```

```
A(M|X(Part(M,Nat),N(=(M,0)))) E(m|@(m,M)) A(n|@(n,M)) N<=(m,n)
```

*For any **M**, a not empty part of Nat, there exists **m** in **M** such that for any **n** in **M**, **m** <= **n**.*

2756(T) NMin existence

```
A - - + - - - + - - - - - - - + - - - + - - -
^ X               E - - - + - - - - - - + -
M   P     N       ^ T X
    a ^ N   =     m     @     A - + - - - +
    r M a     ^ 0         ^ ^ ^ @     N
    t   t     M           m M n   ^ ^ < ^ ^
                                  n M = m n
```

```
A(M|X(Part(M,Nat),N(=(M,0)))) E(m) X(@(m,M),A(n|@(n,M)) N<=(m,n))
```

*For any **M**, a not empty part of Nat, there exists **m** such that **m** is in **M** and for any **n** in **M**, **m** <= **n**.*

2757(T) NMin uniqueness

```
A - - + - - - + - - - - - + - - - + - - - - - - - + - - - + - - - - - -
^ X               A - - + - - - - - - + - - - - - - - - - - - - - - + -
M   P     N       ^ X                     A - - + - - - - - - + - - - +
    a ^ N   =     m   @     A - + - - - + ^ X                     =
    r M a     ^ 0 1     ^ ^ ^ @     N     m   @     A - + - - - +   ^ ^
    t   t     M         m M n   ^ ^ < ^ ^ 2     ^ ^ ^ @     N       m m
                        1   1   n M = m n       m M n   ^ ^ < ^ ^   1 2
                                1     1 1       2   2   n M = m n
                                                        2     2 2
```

```
A(M|X(Part(M,Nat),N(=(M,0))))
A(m1|X(@(m1,M),A(n1|@(n1,M)) N<=(m1,n1)))
A(m2|X(@(m2,M),A(n2|@(n2,M)) N<=(m2,n2)))
=(m1,m2)
```

2758(F) NMin definition (minimum of a not null subset of natural numbers)

```
A - - + - - - + - - - - + + - - - + - - + -
^ X               X
M   P     N         @       A - + - - - - +
    a ^ N   =         N   ^ ^ @     N
    r M a     ^ 0     M ^ M n   ^ ^ < N   ^
    t   t     M       i M       n M = M ^ n
                      n               i M
                                      n
```

```
A(M|X(Part(M,Nat),N(=(M,0)))) X(@(NMin(M),M),A(n|@(n,M)) N<=(NMin(M),n))
```

*For any **M** included in Nat and not empty, NMin(**M**) belongs to **M**, and for any **n** in **M**, NMin(**M**) <= **n**.*

2767(T) Factors are not null iff product is not null

```
A - + - - - - - - - - - + - - - - - - - - + - -
^ @     A - + - - - - - - - - - + - - - - - + -
m   ^ N ^ @     D
    m a n   ^ N   X                 N
      t     n a     N       N         =
              t       =       =       N     0
                        ^ 0     ^ 0     * ^ ^
                        m       n         m n
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) D(X(N(=(m,0)),N(=(n,0))),N(=(N*(m,n),0)))
```

*Let **m**, **n** be natural numbers. **m** is not null and **n** is not null if and only if **m·n** is not null.*

2774(T) Existence of division of natural numbers

```
A - + - - - - - - - - - - - - - - - - - - - + - - - - - - - -
^ @     A - - + - - - + - - - - - - - - - - - - - - + - - - - +
m   ^ N ^ X               E - + - - - - - - - - - - - + - - - -
    m a n   @     N       ^ @     E - + - - - - - - - - + - + -
      t       ^ N   =     q   ^ N ^ @     X
              n a     ^ 0     q a r   ^ N   =           @
                t     n         t     r a     ^ N           ^ ^
                                        t     m + N     ^   r n
                                                  * ^ ^ r
                                                    n q
```

```
A(m|@(m,Nat))
A(n|X(@(n,Nat),N(=(n,0))))
E(q|@(q,Nat))
E(r|@(r,Nat))
X(=(m,N+(N*(n,q),r)),@(r,n))
```

*For any natural number **m**, and*
*for any not null natural number **n***
*there exists a natural number **q**, and*
*there exists a natural number **r***
*such that **m** = **n·q** + **r**, and **r** belongs to **n**.*

2777(T) A factor of a prod. of not null nat. numbers is less than or equal to the product

```
A - - + - - - + - - - - - - - - - - - + - + -
^ X               A - - + - - - + - - - - - +
m   @     N       ^ X               N
      ^ N   =     n   @     N       < ^ N
      m a     ^ 0       ^ N   =     = m * ^ ^
        t     m         n a     ^ 0       m n
                          t     n
```

```
A(m|X(@(m,Nat),N(=(m,0)))) A(n|X(@(n,Nat),N(=(n,0)))) N<=(m,N*(m,n))
```

*Let **m**, **n** be not null natural numbers. Then **m** <= **m·n**.*

In theorem 2804 we demonstrate the uniqueness of the division of natural numbers.

2805(T) PNat explanation (pairs of natural numbers)

```
A - - - + - - - - - - - - - - + - - -
^ T D
p     @     E - + - - - - - - - - + -
        ^ P ^ @     E - + - - - - - +
        p N m   ^ N ^ @     =
          a     m a n   ^ N   ^ P
          t       t     n a   p a ^ ^
                          t     i m n
                                r
```

```
A(p) D(@(p,PNat),E(m|@(m,Nat)) E(n|@(n,Nat)) =(p,Pair(m,n)))
```

2806(T) Pair of natural number corollary

```
A - + - - - - - - - - - - - + - - - - - -
^ @     A - + - - - - - - - - - - - + - -
m   ^ N ^ @     E - + - - - - - + - - - +
    m a n   ^ N ^ @     X
      t     n a p   ^ P   =       =
              t     p N     ^ C     ^ C
                      a     m o ^   n o ^
                      t       o p     o p
                              r       r
                              1       2
```

```
A(m|@(m,Nat)) A(n|@(n,Nat)) E(p|@(p,PNat)) X(=(m,Coor1(p)),=(n,Coor2(p)))
```

*For any natural number **m** and for any natural number **n**, there exists a pair of natural numbers **p** such that **m** = Coor1(**p**) and **n** = Coor2(**p**).*

2807(T) Existence of division of natural numbers (corollary)

```
A - + - - - - - - - - - - - - - - - - - - + - - - - - - - - - - -
^ @     A - - + - - - + - - - - - - - - - - - - + - - - - - - - +
m   ^ N ^ X               E - - - + - - - - - - - - + - + - - + -
    m a n   @     N       ^ T X
      t       ^ N   =     p     @     X
              n a   ^ 0         ^ P   =                 @
                t     n         p N   ^ N               C   ^
                                  a     m + N       C     o ^ n
                                  t         * ^ C   o ^   o p
                                              n o ^ o p   r
                                                o p r     2
                                                r   2
                                                1
```

```
A(m|@(m,Nat)) A(n|X(@(n,Nat),N(=(n,0)))) E(p)
X(@(p,PNat),X(=(m,N+(N*(n,Coor1(p)),Coor2(p))),@(Coor2(p),n)))
```

*For any natural number **m** and for any not null natural number **n**, there exists **p** in PNat such that **m** = **n**·Coor1(**p**) + Coor2(**p**) and Coor2(**p**) belongs to **n**.*

2808(T) PNat corollary

```
A - + - - - + - - + - + - - - + - - - + -
^ @     X
p   ^ P   =             X
    p N     ^ P           @       @
      a     p a C   C       C   N   C   N
      t       i o ^ o ^     o ^ a   o ^ a
              r o p o p     o p t   o p t
                r   r       r       r
                1   2       1       2
```

```
A(p|@(p,PNat)) X(=(p,Pair(Coor1(p),Coor2(p))),X(@(Coor1(p),Nat),@(Coor2(p),Nat)))
```

*Let **p** be a pair of natural numbers. Then **p** = Pair(Coor1(**p**), Coor2(**p**)), and Coor1(**p**) belongs to Nat, and Coor2(**p**) belongs to Nat.*

2810(F) NDivMod definition

```
A - + - - - - - - - - - - - - - - + - - - - + - - - - - + - - - + - - - - + - -
^ @     A - - + - - - + - - - - - + - - - - - - + - - - + - - - + - - - - - + +
m   ^ N ^ X               X
    m a n   @     N         @         X
      t       ^ N   =         N     P   =                   @
              n a     ^ 0     D ^ ^ N     ^ N                 C         ^
                t     n       i m n a     m + N         C     o N       n
                              v     t         * ^ C     o N   o D ^ ^
                              M                 n o N   o D ^ ^ r i m n
                              o                   o D ^ ^ r i m n 2 v
                              d                   r i m n 2 v       M
                                                  1 v       M       o
                                                    M       o       d
                                                    o       d
                                                    d
```

```
A(m|@(m,Nat))
A(n|X(@(n,Nat),N(=(n,0))))
X(
  @(NDivMod(m,n),PNat),
  X(
    =(m,N+(N*(n,Coor1(NDivMod(m,n))),Coor2(NDivMod(m,n)))),
    @(Coor2(NDivMod(m,n)),n)
  )
)
```

*Let **m**, **n** be natural numbers with **n** not null. Then:*
*NDivMod(**m**, **n**) belongs to PNat, and*
***m** = **n** · Coor1(NDivMod(**m**, **n**)) + Coor2(NDivMod(**m**, **n**)), and*
*Coor2(NDivMod(**m**, **n**)) belongs to **n**.*

2811(F) NDiv definition (division quotient)

```
A - + - - - - - - - - - - - - + - - - + -
^ @     A - - + - - - + - - - - + - - - +
m   ^ N ^ X               =
    m a n   @     N         N     C
      t       ^ N   =       D ^ ^ o N
              n a     ^ 0   i m n o D ^ ^
                t     n     v     r i m n
                                  1 v
                                    M
                                    o
                                    d
```

A(m|@(m,Nat)) A(n|X(@(n,Nat),N(=(n,0)))) =(NDiv(m,n),Coor1(NDivMod(m,n)))

*Let **m**, **n** be natural numbers, with **n** not null. Then NDiv(**m**, **n**) = Coor1(NDivMod(**m**, **n**)).*

2812(F) NMod definition (division remainder)

```
A - + - - - - - - - - - - - - + - - - + -
^ @     A - - + - - - + - - - - + - - - +
m   ^ N ^ X               =
    m a n   @     N         N     C
      t       ^ N   =       M ^ ^ o N
              n a     ^ 0   o m n o D ^ ^
                t     n     d     r i m n
                                  2 v
                                    M
                                    o
                                    d
```

A(m|@(m,Nat)) A(n|X(@(n,Nat),N(=(n,0)))) =(NMod(m,n),Coor2(NDivMod(m,n)))

*Let **m**, **n** be nat. numbers, with **n** not null. Then NMod (**m**, **n**) = Coor2(NDivMod(**m**, **n**)).*

2813(T) NDiv, NMod (division of natural numbers)

```
A - + - - - - - - - - - - - - - - + - - - - + - - - - + - - - - + - - + - - - + - -
^ @     A - - + - - - + - - - - - - + - - - - + - - - - - - + - - + - - + - - - + +
m   ^ N ^ X               X
    m a n   @     N         X                     X
      t       ^ N   =         @         @           =                     @
              n a     ^ 0       N     N   N     N     ^ N                   N     ^
                t     n         D ^ ^ a   M ^ ^ a     m + N         N       M ^ ^ n
                                i m n t   o m n t         * ^ N     M ^ ^   o m n
                                v         d                 n D ^ ^ o m n   d
                                                              i m n d
                                                              v
```

A(m|@(m,Nat)) A(n|X(@(n,Nat),N(=(n,0)))) X(X(@(NDiv(m,n),Nat),@(NMod(m,n),Nat)), X(=(m,N+(N*(n,NDiv(m,n)),NMod(m,n))),@(NMod(m,n),n)))

*Let **m**, **n** be natural numbers, with **n** not null. Then:*
*NDiv(**m**, **n**) and NMod(**m**, **n**) are natural numbers, and*
***m** = **n** · NDiv(**m**, **n**) + NMod(**m**, **n**), and*
*NMod(**m**, **n**) belongs to **n**.*

Less "natural" properties of natural numbers

We all have an intuitive understanding of natural numbers; we know they are 0, 1, 2, 3, and so on, and we can do calculations with them. However, if we look a little over the theorems about natural numbers, we find many somewhat strange properties. Let's make a short list of some of these properties. We will use the traditional notations.

```
3 + 1 = 3 U {3}
5 ∈ 6
5 ⊆ 9
4 - 2 = {0, 1}
4 \ 2 = {2, 3}
(4 - 2) U (4 \ 2) = 4
```

Let m, n, p be natural numbers and x a set. Then:

```
m ≤ n ↔ (m ∈ n ∨ m = n)
m ∈ n → m ⊆ n
m ⊆ n ↔ m ≤ n
m ≤ n ↔ m ∈ n + 1
x ∈ m → x ∈ Nat
m = Suc(m) \ {m}
m ⊆ Suc(m)
n ∈ m → m ∉ n
(m ∈ n ∧ n ∈ p) → m ∈ p
m ⊆ Nat
U(Nat) = Nat
m ∩ ((m + n) \ m) = 0
m + n = m U ((m + n) \ m)
n ≠ 0 → m ∈ m + n
p ∈ m - n ↔ p + n ∈ m
m ∈ n ↔ m + p ∈ n + p
m ≤ n ↔ m ∩ n = m
m \ n ⊆ Nat
m ∈ n ↔ Suc(m) ≤ n
Nat \ n = {x | x ∈ Nat ∧ n ≤ x}
…
```

Instead of conclusions

Our game with sky blue and yellow pieces has started with classes and sets and continued with the class algebra. Using the fact that an ordered pair of two sets represents a connection between those sets, we have defined the Cartesian product, graphs, relations, and functions. We went through natural numbers with Peano's axioms, mathematical induction, finite recursion, arithmetic of natural numbers, and functions with natural numbers as their domain.

This work ends here, but not our game! It continues by defining integers, rational and real numbers with their properties. It even attacks the analysis of real numbers with its limits! Theorems and theorems, all proven through our system!

Yes, it was a wonderful game that we will continue! If you like the sentence below, you can join our game by creating more beautiful mathematics!

A						+				+		
	T	A					+					+
			T	D								
					S			@				
					u	^	^		T		C	
					c				o	^	o	^
					c				B		n	
					e				e		t	
					s						i	
					s						n	
											u	
											e	
											d	

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.

Index

addition of natural numbers 238
Ant 171, 172, 249
antisymmetric 171
Attachment 61, 68
Axiom of global choice 189
Bij 202, 204, 211
bijective function 202
Cart 152, 153
Cartesian product 152
Choice 188
choice function 188
Circ 277
Class generator to form 66
CmpAb 218, 250
Cnt 198
Commutativity 93
comparable elements 218
complement / Remainder 137
Composite of graphs 162
Concat 285, 287, 288
Const 180, 181
constant graph function 180
Coor1 154
Coor2 155
Cross attachment 62, 93
Cross replacement 62, 96
Dbl 131, 134
Deleting a quantifier 62, 80
Dia 171, 172
Difference of two classes 157
DImg 186
direct image 186
Dis 138, 169
disjoint classes 138
Dom 163, 164
Domain of the graph 163
DomInt 182, 183, 184
Dual quantifier distributivity 62, 83
Duplicability 58
Duplication (conjunction) 62, 71
Duplication (implication) 62, 73
Empty 122
EqCl 216
Equ 215
equivalence class 216
equivalence relation 215
Existential generalization 62, 75
Extracting the condition of the quantifier 76
Extracting the hypothesis 61, 62
Finite 257, 272
finite sets 257
Form to class generator 61, 67
fully ordered 219
Fun 199, 200, 201
Func 173
function 199
FuOrd 219
generalised union 167
generalized intersection 167
GI 167
Graph 158, 159
graph function 173
GU 167
I 137
Ident 177
identic graph function 177
Implicability 60
Inj 202
injective function 202
Inserting a sentence into another sentence 62, 98
Inserting the condition into the quantifier 79
Interchanging quantifiers 62, 89
intersection 137
Inv 161
inverse of a graph 161
IsSng 197
last permutation 300
LastPerm 300, 301
Min 220
MplFun 240
multiplication of natural numbers 242
N- 252
N* 242

N+ 238
N<= 245
Nat 222
natural number 1 238
natural number 2 254
natural numbers 222
NDiv 309
NDivMod 308
Negation 61, 65
NLE 248
NMin 305
NMod 309
OneToOne 187, 204
Ord 215
Pair 150
ParInv 189
Part 129
partial inverse 189
Partial quantifier distributivity 62, 85
Particularization 62, 91
Parts 166
Path 276
PathDec 290
PathInc 280
Permut 300
permutation 300
power set 166
Pre 262
prefix 34
prefix-arg 34
pre-q-list 34
primitive function 23
Quantifier distributivity 62, 88
R 137
Ran 163, 164
Range of the graph 163
RanInt 182, 183
RecFun 232
RecGra 228
RecInt 229
RecNat 237
recursive function 232
recursive graph 228
Reductio ad absurdum 61, 64
Ref 215
reflexive relation 215
Reflexivity 62, 93
Rel 214
relation 214
Replacement 70
Restricted quantifier distributivity 87
reverse image 186
RImg 186
Sets 256
SFunc 179
simple graph function 179
singleton 134
Sng 134
SPath 299
SubPath 277
Suc 221
successor of a set 221
successor set 221
SucFun 235
SucSet 221
Sur 202
surjective function 202
Sym 170
symmetric graph 170
Tautologize 61, 63
the content of a singleton 198
Tra 170
transitive graph 170
transitive set 224
TraSet 224
U 137
union 137
Univ 135
Universal generalization 62, 75
u-prefix 35
u-prefix-arg 35
Val 174
variables 12, 25, 38, 109, 112
Verifying the proof 101
well ordered 219
WeOrd 219