Number Theory beyond Frege

On the necessity of open arity

Hannes Hutzelmeyer

Summary

A closer look at mathematical proofs led Gottlob Frege to realize that Aristotle's syllogism logic was not sufficient for many theorems. He developed what today is called first-order predicate logic. It is usually thought that predicate logic is sufficient for the theory of natural numbers. However, this **first step** of modern logic development again is not sufficient. One needs another step, especially to allow for so-called **open arity of arrays**. This **second step** cannot be done in general in object-language based on predicate logic but only by metalanguage. Therefore one needs something like the FUME-method (put forward by the author) which allows for a precise treatment of both language levels. Dot-dot-dot … is not admissible in predicate logics as it needs some kind of recursion. In metalanguage, however, one has to introduce some basic recursion right from the setup (but it is much weaker than primitive recursion).

For natural numbers two examples are given, one for a concrete version of Robinson arithmetic and one for recursive arithmetic. Without the second step to metalanguage one **cannot express** some of the most important so-called theorems of number theory in a direct fashion, leave alone prove them. Actually some are not theorems but metatheorems. The examples comprise Chinese remainders, Gödel's beta-function, little Gauss's summing up of numbers, Euclid's unlimited primes and the canonical representation of a natural number (fundamental theorem of natural arithmetic).

After one has included the second step which allows one to talk about open arities in metalanguage one can tackle the problem of talking about number-arrays in object language. One can do this to a certain extent by **coding** number-arrays by (usually) two numbers. This can be done even in Robinson arithmetic using 'Gödel's beta-function'. But one has to make use of the second step before one can return to object-language. Of course, the introduction of **two tiers**, i.e. object-language **and** metalanguage, is necessary for many other areas of mathematics, if not to say, most of them.

Contact: Hutzelmeyer@pai.de
https://pai.de

1 Beyond the conventional paradigm of logic of mathematics

It all started in the year of 1879 when Gottlieb Frege put forward his revolutionary 'Begriffsschrift'. Until then the syllogism logic of Aristotle had been considered to be sufficient as the basis of logical reasoning and therefore also of mathematics. Besides the usual logical characters = ≠ ¬ ∨ ∧ → ↔ quantors ∃ ∀ and variables like e.g. A_1 or A_{13} were introduced together with the rules for omnition ∀A_1[...] and entition ∃A_2[...] as well as ***relation-constant*** and ***function-constant*** strings that allowed for expressing mathematical sentences in a proper fashion. Freges notation differs from this modern form, but that is irrelevant.

The author was confronted with this status when started studying physics, mathematics and philosophy of science in the year of 1960. For a long time he did not enter the field of **number theory** *, however, he always had a bad feeling about theorems of number theory, that he could not relate to the axiomatic approach to, say Robinson arithmetic. The problem to start with is not the proving of theorems of number theory. The first problem is* ***just to write down sentences*** *that are called theorems of number theory. Mathematicans and logicians have constructed complicated systems of so-called classical and intuitionistic logic, theory of types, axiomatic set theory and so on. But are these methods really sufficient for expressing basic sentences of number theory, leave alone proving them in a purely deductive fashion from basically true sentences or axioms ? The author contradicts this question and shows a way out by the FUME-method. He claims that you need both* ***object language and metalanguage*** *being formulated with the rigor of formal logic and some basic recursion thrown in. The examples of section 3 to 7 will - hopefully - clarify his reasoning. The problem is called* ***open arity****. It is not the only reason for the FUME-method, but it is a particularily striking one. 'Dot-dot-dot' ... is just not a legitmate language element in a precise language. For a short introduction to the FUME-method download file* Snark1.1 .pdf *from* https://pai.de .

Heuristically speaking, **sequences** consist of some kind of infinitely many constituents with a definite start and no end, with a line-arrangement, one constituent put behind another. An **array** is a finite ordered collection of constituents with a start and an end (where the constituents are separated by a special character). It has an **arity** given by a natural number that is the count of its constituents. An example for an array is the alphabet of letters separated by commas '*a, b, … z*' with arity *26*, but also the simple array of zeros separated by semicolons '*0;0;0;0;0*' is an example with arity *5*.

Of course this examples are not satisfactory, one needs a precise description for arrays. The FUME-method will be applied as one obviously needs a language that allows for some recursion. If the constituents are taken from a **calcule** of the object language **Funcish**, one has to define arrays in meta-language **Mencish**. The systems of Funcish are called **calcules** by the author, they are not to be confused with various calculus-systems or the calculus of real numbers. There are **concrete** and **abstract** calcules.

The **font-method** is used to distiguish between the various levels of languages: *Times New Roman* of all styles for normal text in English e.g. , *Symbol* and *Arial* boldface italics for metalanguage Mencish e.g. ***number-array(A1)*** and normal *Symbol* and *Arial* for object language Funcish e.g. ∀A_1[(A_1+0)=A_1] .

The other frontier where usual predicate logic is not sufficient for mathematics is connected with higher than first-order logic. Axiomatic set theoty claims that all of infinity mathematics is covered by it. The author, however, has some doubts. Anyhow, the conventional approach to real numbers necessitates second-order logics (for some transcendency axiom, be it Dedekind cuts, interval nesting, Cauchy series or whatever). In group theory second-order is just around the corner, as factors, subgroups, normal subgroups, kernels etc. are not first-order entities.

Mathematicians usually do not even mention that there might be a problem at the foundations. And physicists happily use transcendental mathematics although no one has ever measured anything but a rational number. How about dimensionless constants in physics? Sommerfeld's fine-structure constant, is it a real number and is there a deeper reason for its size. You see, once one is thinking about ***transcendental*** *numbers, one is entering the field of theology, which shows that the name of this numbers has been chosen perfectly!*

2 Metalingual introduction of number-arrays and more

In metalanguage Mencish there are straightforward metaproperties of strings like ***number*** , ***number-array***, ***variable*** , ***sentence*** or ***formula*** and metafunctions for string-replacement $(A;A\int A)$ and character-deletion $(A\partial A)$, the relevant examples are given in appendix A. One can define ***number-array*** strings by the simple recursion in metalanguage Mencish

number-array **::** ***number*** ¦ ***number-array*** ; ***number***

However, one has to find a way to talk about ***number-array*** strings in Funcish. This will be possible by **coding** ***number-array*** strings by ***number*** strings. That is what it is all about. The following metadefinitions are a little abbreviated, but straightforward, the necessary recursions are admissible in Mencish. For definiteness it is done for the concrete calcule ALPHA of Robinson decimal[1)] natural arithmetic (as described in the next section). However, the only feature that is used are the decimal numbers themselves, so that the metadefinitions can be transferred to other concrete arithmetic calcules like e.g. LAMBDA of decimal pinition arithmetic (which allows for primitive recursive functions):

$A''(A)$ — 0 — if not ***number***
succession — 1 2 3 … — else decimal succession, recursively defined as follows:
$A''(0)=1$ A $A''(1)=2$ … $A''(8)=9$ $A''(9)=10$
$A''(A_1 0)=A_1 1$ $A''(A_1 1)=A_1 2$ … $A''(8)=A_1 9$
$A''(A_1 9)=A''(A_1)0$ *with concatenation*

$A\square\square(A)$ — 1 2 3 … — decimal length, count of ***char***, recursively defined as follows:
length — $A\square\square(A_2)=1$ if $\mathbf{char}(A_2)$ $A\square\square(A_1 A_2)=A''(A\square\square(A_1))$ with $\mathbf{char}(A_2)$

$A\Diamond\Diamond(A)$ — 0 — if not ***number-array***
arity — 1 2 3 … — else decimal arity, defined as follows (count of ***semicolon*** strings):
$A\square\square(;((((((((((A_1\partial 0)\partial 1)\partial 2)\partial 3)\partial 4)\partial 5)\partial 6)\partial 7)\partial 8)\partial 9))$

$A\nabla\nabla(A;A)$ — *0* — if A_1 not ***number-array*** or if A_2 not ***number***
projects array-constituent — or if A_2 ***number*** but not less than $A\Diamond\Diamond(A_1)$
number — else constituent at position [3)] A_2, recursively defined as follows:

$A{<<}A$ — *false* — if A_1 or A_2 are not ***number number*** strings
minority — if ***number*** strings, recursively defined as follows:
$A_1 << A''(A_1)$ $[A_1<<A_2]\rightarrow[A_1<<A''(A_2)]$
$\neg[A_1<<A_1]$ $[A_1<<A_2]\rightarrow[\neg[A_2<<A_1]]$

$$\forall A_1[\forall A_2[[[[\mathbf{number\text{-}array}(A_1)]\wedge[\mathbf{number}(A_2)]]\wedge[A_2<<A\Diamond\Diamond(A_1)]]\rightarrow$$
$$[[[A_2=0]\wedge[A\nabla\nabla(A_1;A_2)=A_1]]\vee[[[0<<A_2]\wedge[\mathbf{number}(A\nabla\nabla(A_1;A_2))]]\wedge$$
$$[\exists A_3[\exists A_4[[[\mathbf{number\text{-}array}(A_3)]\wedge[A\Diamond\Diamond(A_3)=A_2]]\wedge$$
$$[[A_1=A_3;A\nabla\nabla(A_1;A_2);A_4]\vee[A_1=A_3;A\nabla\nabla(A_1;A_2)]]]]]]$$

And one defines ***distinct-variable-array*** and ***omny*** strings with a little more complicated recursion using binary metarelation $A\supset A$, that states that string A_1 is suitably containing string A_2 .

$$\forall A_1[[\mathbf{distinct\text{-}variable\text{-}array}(A_1)]\leftrightarrow[[\mathbf{variable}(A_1)]\vee[\exists A_2[\exists A_3[[[$$
$$[\mathbf{distinct\text{-}variable\text{-}array}(A_2)]\wedge[\mathbf{variable}(A_3)]]\wedge[\neg[A_2\supset A_3]]]\wedge[A_1=A_2;A_3]]]]]]$$

omni **::** ∀ ***variable*** [¦ ***omni*** ∀ ***variable*** [

$$\forall A_1[[\mathbf{omny}(A_1)]\leftrightarrow[[\mathbf{omni}(A_1)]\wedge[\mathbf{distinct\text{-}variable\text{-}array}((((A_1;[\forall\int;)\partial[)\partial\forall))]]]$$

[1)] using decimal numbers is just for convenience

[2)] $A''(A)$, $A\square\square(A)$, $A\Diamond\Diamond(A)$, $A\nabla\nabla(A;A)$, $A{<<}A$ with double symbols defined with decimal numbers correspond to general $A'(A)$, $A\square(A)$, $A\Diamond(A)$, $A\nabla(A;A)$, $A<A$ with double symbols defined with petit numbers

[3)] an array has **constituents**, **place** numbers constituent from left *1* to arity *a*, **position** numbers from *0* to *a-1*

3 *Robinson natural numbers arithmetic and Gödel's beta-function*

In the following the concrete calcules <u>ALPHA</u> of Robinson arithmetic and <u>LAMBDA</u> of pinition arithmetic this will be investigated with respect to arrays. One cannot directly talk about ***number-array*** strings of unspecified arity within Funcish as one cannot express it e.g. in <u>ALPHA</u> with dot-dot-dot and one cannot name a ***variable*** $A_?$ so that the arity is properly represented: ~~∀A_1[∀A_2[… [∀$A_?$[...] …]]~~

Concrete calcule <u>ALPHA</u> of Robinson decimal natural arithmetic uses the following alphabet which is not the shortest one, but it is tried keep as close to conventional logic language as possible:

Arial 8, petit-number for variables										*Arial 12, normal size numbers for decimal individuals*									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
Symbol 12, general logic symbols,														*special calcule symbols*					
=	≠	¬	∨	∧	→	↔	∃	∀	[	]	(	)	;	′	+	×	<	A	

The ontological basis of concrete calcules <u>ALPHA</u> of decimal Robinson arithmetic consists of ***decimal-number*** strings (0 1 2 …), unary succession ***function-constant*** A′ , binary addition ***function-constant*** (A+A) , binary multiplication ***function-constant*** (A×A) and binary minority ***relation-constant*** A<A . The start of derivations of ***THEOREM*** strings is given by so-called ***Basiom*** strings (corresponding to ***Axiom*** strings of abstract calcules). In the usual fashion there are:

Start-existence, injectivity, unary and multary induction of succession A′ .
Right zero and right iteration of addition (A+A)
Right zero and right iteration of multiplication (A×A)
Diagonal succession, iteration succession, non-reflexitivity and antisymmetry of minority A<A .

The so-called **'chinese remainder theorem'** is actually a <u>metatheorem</u> ; it is necessary for Gödel's beta-function; both necessitate open arities.

<u>Chinese remainder metatheorem</u> : if the constituents of a ***number-array*** ***A_2*** of arity***A_1*** are pairwise coprime and if they are larger than the corresponding constituents of a ***number-array*** ***A_3*** of same arity, then there is exactly one ***number*** ***A_{10}*** (less than the product of the constituents of ***A_2***) such that every constituent of ***A_3*** is obtained as remainder of the division of ***A_{10}*** by the corresponding one of ***A_2***. This flowery wording has to be translated into precise metalanguage [1)]. Some string manipulations of section 2 are needed: rela***tion-constant*** ***A <<A*** and ***function-constant*** strings ***A◊◊(A)*** , ***A∇∇(A;A)*** , ***(A;A∫A)*** and ***(A∂A)*** .

```
∀A1[ ∀A2[ ∀A3[ [ [ [ [ [ [ [ number(A1) ] ∧ [ 1<<A1 ] ] ∧ [ number-array(A2) ] ] ∧
[ number-array(A3) ] ] ∧ [ A◊◊(A2) = A1) ] ] ∧ [ A◊◊(A3) = A1) ] ] ∧
[ ∀A4[ ∀A5[ ∀A6[ [ [ [ [ [ [ number(A4) ] ∧ [ number(A5) ] ] ∧ [ number(A6) ] ] ∧ [ A4 <<A1) ] ] ∧
[ A5 = A∇∇(A2;A4) ] ] ∧ [ A6 = A∇∇(A3;A4) ] ] → [ [ [ 1<<A5 ] ∧ [ A6<<A5] ] ∧
[ ∀A7[ ∀A8[ [ [ [ [ number(A7) ] ∧ [ number(A8) ] ] ∧ [ A7 <<A4) ] ] ∧ [ A8= A∇∇(A2;A7) ] ] →
[ TRUTH( ∀A1[∀A2[∀A3[[[A5 =(A1×A2)]∧[A8 =(A1×A3)]]→[A1=1]]]] ) ] ] ] ] ] ] ] ] ] ] ∧
[ ∀A9[ [ [ number(A9) ] ∧ [ TRUTH(                                  pairwise coprime
A9 = ( ((((((((((A1 ∂0) ∂1) ∂2) ∂3) ∂4) ∂5) ∂6) ∂7) ∂8) ∂9) ; ; ∫ ( ) (1×( A2 ; ; ∫ )×)) ) ] ] →
[ ∃A10[ [ [ number(A10) ] ∧ [ A10 <<A9 ] ] ∧                        product of constituents of number-array
[ ∀A11[ ∀A12[ ∀A13[ [ [ [ [ [ [ number(A11) ] ∧ [ number(A12) ] ] ∧
[ number(A13) ] ] ∧ [ A11 <<A1 ] ] ∧ [ A12 = A∇∇(A2;A11) ] ] ∧ [ A13 = A∇∇(A3;A11) ] ] →
[ TRUTH( ∃A1[A10 =(A13+(A12×A1))] ) ] ] ] ] ] ] ] ] ] ]           limited as A1< A10
```

Obviously there is no chance to write this down in object-language! The 'Chinese remainder' is not a ***THEOREM*** of calcule <u>ALPHA</u> but a <u>metatheorem</u> of its <u>metacalcule</u> <u>***ALPHA***</u> .

[1)] Both, object-language Funcish and metalanguage Mencish obey the so-called 'Calculation Criterion of Truth' : a computer can decide if a certain step of reasoning is in accordance with the rules.

In conventional notation: Gödel's beta-function $gbeta(x,y,z)=divrem(x,y(z+1)+1)$ with the division remainder function allows for coding an array of numbers with arity a by two codes x and y with positions z from 0 to $a-1$ or places from 1 to a . Just like above: the so-called **'Gödel's betafunction theorem'** is actually a metatheorem .

Gödel's beta-function metatheorem : a ***number-array*** $\boldsymbol{A_5}$ of arity $\boldsymbol{A_4}$ can be coded by two ***number*** strings $\boldsymbol{A_1}$ and $\boldsymbol{A_2}$ such that every constituent of the ***number-array*** can be obtained using a suitable ternary ***UNEX-formulo*** (see Appendix A) ***AXFOgbeta*** [1] that represents Gödel's beta-function in calcule ALPHA and that has free ***variable*** strings A_0 for result, A_1 , A_2 as codes and A_3 as position inside the array, $A_3 < \boldsymbol{A_1}$.

$$\forall A_4[\ \forall A_5[\ [\ [\ [\ \mathbf{number}(A_4)\ ] \wedge [\ \mathbf{number\text{-}array}(A_5)\ ]\ ] \wedge [\ A_4 = A\Diamond\Diamond(A_5)\ ]\ ] \rightarrow$$
$$[\ \exists A_1[\ \exists A_2[\ [\ [\ \mathbf{number}(A_1)\ ] \wedge [\ \mathbf{number}(A_2)\ ]\ ] \wedge$$
$$[\ \forall A_3[\ \forall A_6[\ [\ [\ [\ [\ \mathbf{number}(A_3)\ ] \wedge [\ A_3 \ll A_4\ ]\ ] \wedge [\ \mathbf{number}(A_6)\ ]\ ] \wedge [\ A_6 = A\nabla\nabla(A_3;A_4)\ ]\ ] \rightarrow$$
$$[\ \mathbf{TRUTH}(\ ((((\ \mathbf{AXFOgbeta}\ ;\ A_1 \int A_1\ )\ ;\ A_2 \int A_2\ )\ ;\ A_3 \int A_3\ )\ ;\ A_0 \int A_6\ )\ )\ ]\ ]\ ]\ ]\ ]\ ]\ ]\ ]$$

It is proven by taking

$$\mathbf{AXFOgbeta} = \exists A_{20}[[(((A_2\times A_3')'\times A_{20})+A_0)=A_1]\wedge[A_0<(A_2\times A_3')']]$$

and applying the Chinese remainder metatheorem . The auxiliary bound ***variable*** A_{20} is chosen such that it does not easily collide with free ***variable*** strings when the ***AXFOgbeta*** is inserted in a ***phrase*** string; obviously A_{20} is limited by A_1 .

Based on Gödel's beta-function metatheorem one can talk talk about ***number-array*** strings of any arity in the following way **within** concrete calcule ALPHA of decimal Robinson arithmetic. Interpret ***variable*** A_4 as arity, A_1 and A_2 as codes, A_3 as position within ***number-array*** and A_0 as unique result:

$$\forall A_1[\forall A_2[\forall A_3[\forall A_4[[[0<A_4]\wedge[A_3<A_4]]\rightarrow[\forall A_0[[\mathbf{AXFOgbeta}]\rightarrow[\ \dots\ ]]]]]]]$$

If one does not like the idea of **two** code ***number*** strings one can combine them into one ***number*** by so-called anti-diagonal pair coding that also can be represented in calcule ALPHA , conventionally written as pair of row and column $p=adp(j,k)=j+((j+k)(j+k+1))/2$ and its inverse functions for row $j=adr(p)=p-(ada(p)(ada(p)+1))/2$ and for column $k=adc(p)=((ada(p)+1)(ada(p)+2))/2-(p+1)$ with corresponding ***UNEX-formulo*** strings, including auxiliary function $ada(p)=(brt(8p+1)-1)/2$ with entire square-root function $brt(n)$. Five more ***extra-individual-constant*** strings with bound ***variable*** strings that do not collide in the following applications (see binary metarelation $\boldsymbol{A \sim A}$ of appendix A) .

binary ***UNEX-formulo*** : for antidiagoanl pair

$\mathbf{AXFOadp} = (A_0+A_0)=(A_1+((A_1+A_2)\times(A_1+A_2')))$ *simple, necessary for bisection*

unary ***UNEX-formulo*** : for entire square root, antidiagonal auxiliary, row and column

$$\mathbf{AXFObrt} = [(A_0\times A_0)=A_1]\vee[[\exists A_{32}[(A_0\times A_0)<(A_1+A_{32})]]\wedge[\exists A_{32}[(A_1+A_{32})<(A_0'\times A_0')]]]$$

$$\mathbf{AXFOada} = [((A_0+A_0)'\times(A_0+A_0)')=(8\times A_1)']\vee$$
$$[[\exists A_{32}[((A_0+A_0)'\times(A_0+A_0)')<(A_1+A_{32})]]\wedge[\exists A_{32}[(A_1+A_{32})<((A_0+A_0)'\times(A_0+A_0)')]]]$$

$$\mathbf{AXFOadr} = \exists A_{33}[[\ (\mathbf{AXFOada};\ A_0 \int A_{33})\ ]\wedge[((A_0+A_0)+(A_{33}\times A_{33}'))=(A_1+A_1)]]$$

$$\mathbf{AXFOadc} = \exists A_{33}[[\ (\mathbf{AXFOada};\ A_0 \int A_{33})\ ]\wedge[((A_0+A_0)+(A_1'+A_1'))=(A_{33}'+A_{33}'')]]$$

Inserting this properly in ***AXFOgbeta*** gives the desired (but somewhat lengthy) result.

[1] ***AXFOgbeta*** is an ***extra-individual-constant*** that is used like a makro in programming languages, just a name for a string that is to be expanded wherever it appears (one has to take care that no collision of bound ***variable*** strings appear)

4 Recursive natural numbers arithmetic

The choice for a concrete calcule of recursive natural arithmetic is the concrete calcule <u>LAMBDA</u> of decimal pinitive arithmetic. It uses the following alphabet which is not the shortest possible one, but it is tried keep as close to conventional logic language as possible:

Arial 8, petit-number for variables										*Arial 12, normal size numbers for decimal individuals*									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
Symbol 12, general logic symbols,														*special calcule symbols*					
=	≠	¬	∨	∧	→	↔	∃	∀	[	]	(	)	;	∗	#	≤	Λ		□

List of 38 (plus 1 ***extra*** *) characters for ontological* ***basis*** *of calcule* <u>LAMBDA</u>

sort :: Λ
sort-array :: ***sort*** ¦ ***sort-array*** ; ***sort***
decimal :: ***number ::*** 0 ¦ 1 ¦ 2 ¦ … *correct definition see section 5*

basis-ingredient :: ***sort*** ¦ ***decimal*** ¦ ***basis-function-constant*** ¦ ***basis-relation-constant***
basis-function-constant :: Λ( ) ¦ Λ(***sort-array***) ¦ (Λ∗Λ) *pinitive functions, decimal synaption*
basis-relation-constant :: #Λ ¦ Λ≤Λ *pinity, minority*

pinon-catena :: ***pinon*** ¦ ***pinon-catena pinon***
pinon-array :: ***pinon*** ¦ ***pinon-array*** ; ***pinon***
pinon :: 0 ¦ 1 ¦ 2 ***pinon pinon*** ¦ 8 ***pinon pinon-catena*** 9 *only 4 cases*

pinon strings are natural numbers that **code** primitive recursive functions, when they replace Λ in ***basis-function-constant*** string Λ( ) or Λ(***sort-array***) resp. : 0 codes the zero function, 1 codes succession. The third case 2 ***pinon pinon*** codes straight recursion, where the left ***pinon*** of intrinsic arity *m* gives the initial value and the right ***pinon*** of intrinsic arity *n* gives the iteration function (the intrinsic arity of the new ***pinon*** is *max(m+1,n-1)*). The last case 8 ***pinon pinon-catena*** 9 codes composition of functions with any intrinsic arity: the left ***pinon*** is the function where the ***pinon*** strings of the ***pinon-array*** are plugged in. The PINITOR calculator that does the calculating is not described here, neither the basic true sentences, that include a **schema** of sentences (or as the author prefers to call it a **mater** of sentences) meaning that they are enumerasbly infinite many (by the way: for a proper introduction of sentence schemata one has to use metalanguage).

The ***basis-function-constant*** (Λ∗Λ) gives the decimal synaption of two strings, which is basically concatenation, except that no leading 0 is admissible. Actually the definition among the ***basis-ingredient*** strings is redundant, as it can be given by a ***pinon*** . The same is true for ***basis-relation-constant*** #Λ and Λ≤Λ as they can be defined using ***pinon*** Λpiny and Λemiy resp. as codes of characteristic functions.

Primitive recursive functions are obtained by ***pinon*** strings, these precede as codes the ***basis-function-constant*** strings Λ( ) and Λ(***sort-array***) . If a number is not a ***pinon*** string the primitive function with this code is simply put to 0 for all input.. Many examples are given in the publication 'Programming primitive recursive functions and beyond' that can be downloaded as file <u>C6-C7-Pinon.pdf</u> on the homepage <u>https://pai.de</u> of the author. Very few examples for coding of primitive recursive functions by decimal numbers are given here:

It is a funny observation that pinitive functions have a Janus face. They have been designed to represent primitive recursive functions, e.g.

22011(Λ_1;Λ_2) the addition of two numbers with ***pinon*** Λadd=22011 e.g. 22011(1;1)=2

But the following is defined too and gives a funny function:

Λ_1(0) the value for all codes at 0 where the result is put to 0 if Λ_1 is not a ***pinon*** code.

By the way: it will turn out that one can talk about ***number-array*** *strings within* <u>LAMBDA</u> *; however, this calcule has the shortcoming that it necessitates enumerably many* ***basis-function-constant*** *strings, as there is no limit on the arity for the* ***sort-array*** *strings of primitive recursive functions.*

The strange functions that can be obtained by putting variables into code position can be generalized to so-called **processive** functions. Composition of functions produces so-called ***scheme*** strings (not to confuse with **schemata** (or matres) of sentences, conventionally they are called 'general terms'). One realizes that ***scheme*** strings that are obtained from ***function-constant*** strings by inserting ***number*** and ***variable*** strings and compositions thereof represent functions. The world of processive functions is very rich, e.g. it comprises straightforwardly **Ackermann function** and other **hyperexponentiations** .

There is a straightforward way in calcule LAMBDA to talk about ***number-array*** strings Λ_3 of arity given by ***number*** Λ_1 . They can be represented by code ***number*** string Λ_2 , is expressed by the metatheorem :

$$\forall\Lambda_3[\ \forall\Lambda_4[\ [\ [\ [\ [\mathbf{number}(\Lambda_3)\ ]\wedge[\ 0\ll\Lambda_3\ ]\ ]\wedge[\ \mathbf{number\text{-}array}(\Lambda_4)\ ]\ ]\wedge[\ \Lambda\Diamond\Diamond(\Lambda_4)=A_3)\ ]\ ]\rightarrow$$
$$[\ \exists\Lambda_1[\ [\mathbf{number}(\Lambda_1)\ ]\wedge$$
$$[\ \forall\Lambda_2[\ \forall\Lambda_5[\ [\ [\ [\ [\mathbf{number}(\Lambda_2)\ ]\wedge[\Lambda_2\ll\Lambda_3\ ]\ ]\wedge[\mathbf{number}(\Lambda_5)\ ]\ ]\wedge[\ \Lambda_5=\Lambda\nabla\nabla(\Lambda_4;\Lambda_3)\ ]\ ]\rightarrow$$
$$[\ \mathbf{TRUTH}(\ \Lambda_5=\Lambda_2(\Lambda_3)\ )\ ]\ ]\ ]\ ]\ ]\ ]\ ]\ ]$$

The proof is quite trivial, one can program a unary primitive recursive function, given any finite count of values of a given arity for the low end of the value table.

Based on this metatheorem one can talk talk about ***number-array*** strings of any arity in the following way **within** concrete calcule LAMBDA of decimal pinitive arithmetic:

$$\forall\Lambda_1[\forall\Lambda_2[\forall\Lambda_3[[[0<\Lambda_3]\wedge[\Lambda_2<\Lambda_3]]\rightarrow[\ \dots\ \Lambda_1(\Lambda_2)\ \dots\ ]]]]$$

As opposed to the preceding section one can talk about the constituent of an ***number-array*** string in a direct way. The reason for this is that concrete calcule LAMBDA allows for primitive recursion and one does not have to take refuge to representation of functions using Gödel's beta-function technique.

But still one has to go the detour in metalanguage in order to correctly refer to ***number-array*** strings as one can only express in metalanguage what is meant by a ***number-array*** string.

5 *Little Gauss's theorem*

Everybody knows the anectode of **little Gaus**s reinventing the method of summing up numbers that was found by Indian mathematician Aryabhata in 499 AD: conventionally written with dot-dot-dot:
(1+2+3+4+ … +n)=n(n+1)/2

How to **express** it in connection with concrete calcule ALPHA of decimal Robinson arithmetic? And another question is, how to prove it? It is not a ***THEOREM*** but a schema (or as the author prefers to call it 'mater') of ***THEOREM*** strings. Therefore it has to be expressed differently:

a) metatheorem of Little Gauss

that is producing successively the trivial ***THEOREM*** strings:

(2×(1+2))=(2×(2+1))
(2×((1+2)+3))=(3×(3+1))
(2×(((1+2)+3)+4)=(4×(4+1))
…

$$\forall A_1[\ \forall A_2[\ [\ [\ [\ [\ [\mathbf{number}(A_1)\ ]\wedge[\mathbf{number\text{-}array}(A_2)\ ]\ ]\wedge$$
$$[\ \exists A_3[\ A_2=1;A_3\ ]\ ]\ ]\wedge[\ \exists A_3[\ A_2=A_3;A_1\ ]\ ]\ ]\wedge$$
$$[\ \forall A_3[\ \forall A_4[\ [\ [\ [\mathbf{number}(A_3)\ ]\wedge[\mathbf{number}(A_4)\ ]\ ]\wedge[\ \forall A_5[\ \forall A_6[\ [\ [\ [\ A_2=A_3;A_4\ ]\vee$$
$$[\ A_2=A_3;A_4;A_6\ ]\ ]\vee[\ A_2=A_5;A_3;A_4\ ]\ ]\vee[\ A_2=A_5;A_3;A_4;A_6\ ]\ ]\ ]\ ]\ ]\rightarrow[\ A_4=A''(\ A_3)\ ]\ ]\ ]\ ]\ ]\rightarrow$$
$$[\ \mathbf{TRUTH}(\ (2\times(\ (((((((((A_2\partial 9)\partial 8)\partial 7)\partial 6)\partial 5)\partial 4)\partial 3)\ 2)\partial 1)\partial 0)\ ;\ ;\ \int(\ )\ (0+(A_2\ ;\ ;\ \int\ )+\ )\ )\ )=$$
$$(A_1\times(A_1+1))\ )\ ]\ ]\ ]$$

initial final
successive
multi-addition

The proof is based on induction for the ***scheme*** $(A_1\times(A_1+1))$ where the start is $A_1=1$ and the induction is based on $((A_1+1)\times((A_1+1)+1))=((A_1\times(A_1+1))+(2\times A_1))$

b) ***THEOREM*** with Gödel's beta-function

One can give a representation of the ***Successive-number-array*** starting from 1 up to arity A_4 using Gödel's beta-function-technique (the existence of A_1 and A_2 are guaranteed by Gödel's beta-function metatheorem (it may be made unique by choosing the smallest A_1) . The first auxiliary ***THEOREM*** states that one can represent the ascending array***Successive-number-array*** by Gödel's beta-function codes:

$\forall A_4[\exists A_1[\exists A_2[\forall A_3[[A_3<A_4]\rightarrow[(\mathbf{\Lambda XFOgbeta;}\ A_0 \int A_3')]]]]]$

and a representation of the successive-sum array thereof

$\forall A_4[\exists A_1[\exists A_2[[((\mathbf{\Lambda XFOgbeta}\ ;\ A_3\int 0)\ ;A_0\int 1)\ ]\wedge$
$[\forall A_3[[A_3<A_4]\rightarrow[\forall A_0[[\mathbf{\Lambda XFOgbeta}]\wedge[((\ \mathbf{\Lambda XFOgbeta}\ ;\ A_3 \int A_3')\ ;\ A_0 \int (A_0+A_3''))]]]]]]]]$

And one can thus state ***THEOREM*** of little Gauss:

$\forall A_4[\exists A_1[\exists A_2[[[((\mathbf{\Lambda XFOgbeta}\ ;\ A_3\int 0)\ ;\ A_0\int 1)\ ]\wedge$
$[\forall A_3[[A_3<A_4]\rightarrow[\forall A_0[[\mathbf{\Lambda XFOgbeta}]\wedge[((\ \mathbf{\Lambda XFOgbeta}\ ;\ A_3 \int A_3')\ ;\ A_0 \int(A_0+A_3''))]]]]]]\wedge$
$[\forall A_0[[(\mathbf{\Lambda XFOgbeta}\ ;\ A_3\int A_4)\ ]\wedge[(A_0+A_0)=(A_4'\times A_4'')]]]]]]$

And one can prove it based on Gödel's beta-function metatheorem and the induction for the ***scheme*** $(A_3\times(A_3+1))$.

c) ***THEOREM*** in a concrete calcule with recursive arithmetic

It is a different story in the concrete calcule LAMBDA of decimal pinitive arithmetic where one has the tools of primitive recursion. The ***number-array*** 1;2;3;4; … ;Λ_1 is coded by arity Λ_1 and ***pinon*** $\Lambda_2=1$. Given two strings [1] and ***Λzllisu*** and ***Λzrblisp*** one can construct a ***pinon*** for every ***Λ2*** by concatenating them to ***Λzllisu Λ2 Λzrblisp*** . For a given arity this ***pinon*** sums up the constituents and there is a ***pinon*** ***Λcarl*** for carlation, conventionally written as (x(x+1))/2 . The ***THEOREM*** of little Gauss reads:

$\forall A_1[\ \mathbf{\Lambda zllisu}\ 1\mathbf{\Lambda zrblisp}(A_1))=\mathbf{\Lambda carl}\ (A_1)]$

This means: once one has realized that ***number-array*** strings can be represented by their arity and a code, one can express the ***THEOREM*** of little Gauss perfectly in LAMBDA and it can be proven within LAMBDA too.

[1] the ***extra-individual-constant*** strings are again used like a makro in programming languages just names for strings that are to be expanded wherever they appear in synaptions

6 *Euclid's theorem of unlimited primes*

Contrary to the preceding section it is not problem to express the ***THEOREM*** of unlimited primes properly in concrete calcule ALPHA of decimal Robinson arithmetic. One starts off with unary ***formula AFAprime***

$$\boldsymbol{AFAprime} = [1<A_1]\wedge[\forall A_{31}[\forall A_{32}[[A_1=(A_{31}\times A_{32})]\rightarrow[[A_{31}=1]\vee[A_{31}=A_1]]]]]$$

that defines prime ***number*** strings and then one can express the ***THEOREM*** :

$$\forall A_1[[\boldsymbol{AFAprime}\,]\rightarrow[\exists A_2[[(\boldsymbol{AFAprime};\, A_1\!\int\! A_2)\,]\wedge[A_1<A_2]]]]$$

However the proof needs **arrays of open arity**. This means that for a proof one has to use the second step and move from object-language to metalanguage (and back). The translation of the ***THEOREM*** into a metatheorem and the arrangements for the proof are a bit tedious but trivial. ***Successive-prime-array*** strings come handy, example 2;3;5;7;11;13

$$\forall A_1[\,[\,\textit{Successive-prime-array}(A_1)\,]\leftrightarrow[\,[\,[\,\textit{number-array}(A_1)\,]\wedge[\,\exists A_7[\,A_1=2;A_7\,]\,]\,]\wedge$$
$$[\,\forall A_2[\,\forall A_3[\,[\,[\,[\,\textit{number}(A_2)\,]\wedge[\,\textit{number}(A_3)\,]\,]\wedge[\,\forall A_4[\,\forall A_5[\,[\,[\,[\,[\,A_1=A_2;A_3\,]\vee$$
$$[\,A_1=A_2;A_3;A_5\,]\,]\vee[\,A_1=A_4;A_2;A_3\,]\,]\vee\,[\,A_1=A_4;A_2;A_3;A_5\,]\,]\,]\,]\,]\,]\rightarrow[\,[\,[\,[\,\textit{TRUTH}(\,($$
$$\textit{AFAprime}\,;\,A_1\!\int\! A_2)\,)\,]\wedge[\,\textit{TRUTH}(\,(\,\textit{AFAprime}\,;\,A_1\!\int\! A_3)\,)\,]\,]\wedge[\,A_2<<A_3)\,]\,]\wedge[\,\forall A_6[\,[\,\textit{TRUTH}($$
$$(\,\textit{AFAprime}\,;\,A_1\!\int\! A_6)\,)\,]\rightarrow[\,[\,[\,[\,A_6=A_2]\vee[\,A_3=A_6]\,]\vee[\,A_6<<A_2]\,]\vee[\,A_3<<A_6]\,]\,]\,]\,]\,]\,]\,]\,]\,]$$

a) metatheorem

$$\forall A_1[\,[\,[\,\textit{number}(A_1)\,]\wedge[\,\textit{TRUTH}(\,(\textit{AFAprime};\,A_1\!\int\! A_1)\,)\,]\,]\rightarrow$$
$$[\,\exists A_2[\,[\,[\,\textit{number}(A_2)\,]\wedge[\,\textit{TRUTH}(\,(\textit{AFAprime};\,A_1\!\int\! A_2)\,)\,]\,]\wedge[\,A_1<<A_2]\,]\,]\,]$$

For the proof construct A_4 from ***Successive-prime-array*** as successor of the product of its constituents. *Metalingual proofs can be lengthy (and a bit boring in its details), so just a sketch is given as usual:*

$[\,\textit{Successive-prime-array}(A_3)\,]\wedge[\,[\,A_3=2;3\,]\vee$
$[\,\exists A_5[\,[\,\textit{Successive-prime-array}(A_5)\,]\wedge[\,A_3=A_5\,A_2\,]\,]\,]\,]$

$$A_4=(\,(\,((((((((((A_3\partial 9)\partial 8)\partial 7)\partial 6)\partial 5)\partial 4)\partial 3)\,2)\partial 1)\partial 0)\,;\,;\int(\,)\,(1\times(\,A_2\,;\,;\int\,)\times\,)\,)\partial)'$$

b) ***THEOREM*** with Gödel's beta-function

The idea is to use ***number-array*** strings as e.g. in conventional notation: *1*, *2*, *6*, *30*, *210*, *2310* that are are generated by successive products of prime ***number*** strings. For a given prime ***number*** A_1 one can find the corresponding ***number-array*** that ends with the constituent that is the product of all preceding primes, its successor is a prime ***number*** greater than the considered one. This can be done using Gödel's beta-function technique with codes A_4 , A_5 and arity A_6'' with the quaternary ***formula*** :

$$\boldsymbol{AFAeupr}=[[((((\boldsymbol{AXFOgbeta};\,A_1\!\int\! A_4);\,A_2\!\int\! A_5);\,A_3\!\int\! 0);\,A_0\!\int\! 1)\,]\wedge \qquad \textit{starts at } 1$$
$$[\exists A_7[[((((\boldsymbol{AXFOgbeta};\,A_1\!\int\! A_4);\,A_2\!\int\! A_5);\,A_3\!\int\! A_6);\,A_0\!\int\! A_7)\,]\wedge \qquad A_6''\textit{ is arity}$$
$$[\exists A_8[[((((\boldsymbol{AXFOgbeta};\,A_1\!\int\! A_4);\,A_2\!\int\! A_5);\,A_3\!\int\! A_6');\,A_0\!\int\! A_8)\,]\wedge[A_8=(A_7\times A_1)]]]]]]\wedge \qquad \textit{ends for } A_1$$
$$[\forall A_9[[A_9<A_6]\rightarrow[\forall A_{10}[\forall A_{11}[[[((((\boldsymbol{AXFOgbeta};\,A_1\!\int\! A_4);\,A_2\!\int\! A_5);\,A_3\!\int\! A_9);\,A_0\!\int\! A_{10})\,]\wedge \qquad \textit{all prime}$$
$$[((((\boldsymbol{AXFOgbeta};\,A_1\!\int\! A_4);\,A_2\!\int\! A_5);\,A_3\!\int\! A_9');\,A_0\!\int\!(A_{10}\times A_{11}))\,]]\rightarrow[[(\,\boldsymbol{AFAprime};\,A_1\!\int\! A_{11})\,]\wedge$$
$$[[1<A_9]\rightarrow[\forall A_{12}[[[(((\boldsymbol{AXFOgbeta};\,A_1\!\int\! A_4);\,A_2\!\int\! A_5);\,A_3\!\int\! A_9'');\,A_0\!\int\!(A_{11}\times A_{12}))\,]\rightarrow[A_{11}<A_{12}]]\wedge$$
$$[\forall A_{13}[[[A_{13}<A_{12}]\wedge[(\,\boldsymbol{AFAprime};\,A_1\!\int\! A_{13})\,]]\rightarrow[\neg[A_{12}<A_{13}]]]]]]]]]]] \qquad \textit{consecutive}$$

For the proof take the construction of a prime ***number*** A_0' greater than A_1 :

$\forall A_1[[\boldsymbol{AFAprime}\,]\rightarrow[\exists A_4[\exists A_5[\exists A_6[[(((\boldsymbol{AFAeupr};\,A_1\!\int\! A_4);\,A_2\!\int\! A_5);\,A_3\!\int\! A_6)\,]\wedge$
$[\forall A_0[[(((\boldsymbol{AXFOgbeta};\,A_1\!\int\! A_4);\,A_2\!\int\! A_5);\,A_3\!\int\! A_6)\,]\rightarrow[[(\,\boldsymbol{AFAprime};\,A_1\!\int\! A_0')\,]\wedge[A_1<A_0']]]]]]]]$

c) It is a different story in the concrete calcule LAMBDA of decimal pinitive arithmetic where one has the tools of primitive recursion. There one can express the ***Successive-prime-array*** by means of code and perform the proof within the calcule.

7 Fundamental theorem of natural arithmetic

The **Fundamental theorem of natural arithmetic** (canonical representation of a natural numbers by unique prime-power decomposition) is illustrated by the example $504=(((((2\times 2)\times 2)\times 3)\times 3)\times 7)$. It cannot be expressed immediately in concrete calcule ALPHA of decimal Robinson arithmetic as a ***THEOREM*** . First one has to take refuge to the corresponding metatheorem:

a) fundamental metatheorem of natural arithmetic

Ascending-prime-array strings come handy, example $2;2;2;3;3;7$,

$$\forall A_1[[\textbf{Ascending-prime-array}(A_1)]\leftrightarrow[[\textbf{number-array}(A_1)]\wedge[[(\textbf{AFAprime};\ A_1\int A_1)]\vee$$
$$[[\forall A_2[\forall A_3[[[[\textbf{number}(A_2)]\wedge[\textbf{number}(A_3)]]\wedge[\forall A_4[\forall A_5[[[[A_1=A_2;A_3]\vee$$
$$[A_1=A_2;A_3;A_5]]\vee[A_1=A_4;A_2;A_3]]\vee[A_1=A_4;A_2;A_3;A_5]]]]]]]\rightarrow[[[\textbf{TRUTH}($$
$$(\textbf{AFAprime};\ A_1\int A_2))]\wedge[\textbf{TRUTH}((\textbf{AFAprime};\ A_1\int A_3))]]\wedge[[A_2=A_3)]\vee[A_2\ll A_3]]]]]]]]]]$$

for expressing the metatheorem :

$$\forall A_1[[[\textbf{number}(A_1)]\wedge[1\ll A_1]]\rightarrow[\exists A_2[[[\textbf{Ascending-prime-array}(A_2)]\wedge[\textbf{TRUTH}(A_1=$$
$$((((((((((((A_2\partial 9)\partial 8)\partial 7)\partial 6)\partial 5)\partial 4)\partial 3)2)\partial 1)\partial 0);;\int()\ (1\times(A_2;;\int)\times))\partial_0)]]\wedge$$
$$[\forall A_3[[[\textbf{Ascending-prime-array}(A_3)]\wedge[\textbf{TRUTH}(A_1=$$
$$((((((((((((A_3\partial 9)\partial 8)\partial 7)\partial 6)\partial 5)\partial 4)\partial 3)2)\partial 1)\partial 0);;\int()\ (1\times(A_3;;\int)\times))\partial_0)]]\rightarrow$$
$$[A_2=A_3]]]]]]$$

The first part states the existence and the second part takes care of uniqueness:The proof necessitates induction, preferably in the form of infinite descent.

b) fundamental ***THEOREM*** in Robinson natural arithmetic with Gödel's beta-function

The idea is to use ***number-array*** strings of products of successive powers of ascending primes, for the above example: $1;8;72;504$, the last one being the ***number*** in question. Firstly the binary ***formula*** prime-power-pair ***AFApripopair*** is defined which is true if the first argument A_1 is a prime ***number*** and the second argument A_2 is a power thereof, e.g. 5 and 125 are such a pair.

$$\textbf{AFApripopair} = [[1<A_1]\wedge[\forall A_{31}[\forall A_{32}[[A_1=(A_{31}\times A_{32})]\rightarrow[[A_{31}=1]\vee[A_{31}=A_1]]]]]]\wedge$$
$$[\forall A_{33}[\forall A_{34}[[A_2=(A_{33}\times A_{34})]\rightarrow[[1<A_{33}]\rightarrow[\exists A_{35}[A_{33}=(A_{35}\times A_1)]]]]]]$$

The fundamental ***THEOREM*** of natural arithmetic in concrete calcule ALPHA looks a little bit complicated (and extends to about 30 lines if one expands ***formula*** strings ***AXFOgbeta*** and ***AFApripopair***), where the first part states the existence and the second part takes care of uniqueness:

$\forall A_1[[1<A_1]\rightarrow[\exists A_4[\exists A_5[\exists A_6[[[[[((((\textbf{AXFOgbeta};\ A_1\int A_4);\ A_2\int A_5);\ A_3\int 0);\ A_0\int 1)]\wedge$	*start* 1
$[((((\textbf{AXFOgbeta};\ A_1\int A_4);\ A_2\int A_5);\ A_3\int A_6);\ A_0\int A_1)]]\wedge$	*end with* A_1
$[\forall A_7[[A_7<A_6]\rightarrow[\forall A_8[\forall A_9[[[((((\textbf{AXFOgbeta};\ A_1\int A_4);\ A_2\int A_5);\ A_3\int A_7);\ A_0\int A_8)]\wedge$	*successive values*
$[((((\textbf{AXFOgbeta};\ A_1\int A_4);\ A_2\int A_5);\ A_3\int A_7');\ A_0\int A_9)]]\rightarrow$	*are*
$[\exists A_{10}[\exists A_{11}[[[((\textbf{AFApripopair};\ A_1\int A_{10})\ A_2\int A_8)]\wedge((\textbf{AFApripopair};\ A_1\int A_{11})\ A_2\int A_9)]]\wedge$	*prime-powers*
$[[0<A_7]\rightarrow[A_{10}<A_{11}]]]]]]]]]]]\wedge$	*ascending primes*
$[\forall A_{24}[\forall A_{25}[\forall A_{26}[[[[((((\textbf{AXFOgbeta};\ A_1\int A_{24});\ A_2\int A_{25});\ A_3\int 0);\ A_0\int 1)]\wedge$	*and the*
$[((((\textbf{AXFOgbeta};\ A_1\int A_{24});\ A_2\int A_{25});\ A_3\int A_{26});\ A_0\int A_1)]]\wedge$	*decomposition*
$[\forall A_7[[A_7<A_{26}]\rightarrow[\forall A_8[\forall A_9[[[((((\textbf{AXFOgbeta};\ A_1\int A_{24});\ A_2\int A_{25});\ A_3\int A_7);\ A_0\int A_8)]\wedge$	*is unique*
$[((((\textbf{AXFOgbeta};\ A_1\int A_{24});\ A_2\int A_{25});\ A_3\int A_7');\ A_0\int A_9)]]\rightarrow$	
$[\exists A_{10}[\exists A_{11}[[[((\textbf{AFApripopair};\ A_1\int A_{10})\ A_2\int A_8)]\wedge((\textbf{AFApripopair};\ A_1\int A_{11})\ A_2\int A_9)]]\wedge$	(*one could pick*
$[[0<A_7]\rightarrow[A_{10}<A_{11}]]]]]]]]]]]]\rightarrow[[A_6=A_{26}]\wedge[\forall A_{15}[[A_{15}<A_6]\rightarrow[\forall A_0[$	*the smallest* A_4)
$[(((\textbf{AXFOgbeta};\ A_1\int A_4);\ A_2\int A_5);\ A_3\int A_{15})]\wedge[(((\textbf{AXFOgbeta};\ A_1\int A_{24});\ A_2\int A_{25});\ A_3\int A_{26})]]]]]]]]]]]]]]$	

c) fundamental ***THEOREM*** of natural arithmetic in a concrete calcule with recursive arithmetic

It is a different story in the concrete calcule LAMBDA of decimal pinitive arithmetic where one has the tools of primitive recursion. There one can express the ***Ascending-prime-array*** by means of its arity and a ***pinon*** code and perform the proof within the calcule where one has the possibility of limited sums and products as was mentioned at the end of section 5.

8 Open arity in other areas of mathematics and conclusion

Open arity and related features are needed in many other areas of mathematics, e.g.

- axiom schemata of **separation** and **replacement** of axiomatic set theory
- **induction** and **recursion** for functions of **any arity** in number theories.
- an infinite count of functions for proper defintion of recursive functions
- geometrical space of **unspecified dimension** (how to express n-tuples)
- definition and use **polynomials**, say for integer, rational or algebraic arithmetics
- systems of **unspecified finite cardinality** (e.g. finite groups and Galois fields).
- finite and infinite **graph theories** and many more.

All of them can be treated properly by the FUME-method with the two-tiers of languages Funcish and Mencish. Of course common English can be used as an unprecise supralanguage to talk about everything. However, it is important to know about the shortcomings of unprecise language. Supralanguage English (or any other natural language) is but a means to express comments and to reason in a plausibible fashion. The precise talking has to be done in Mencish and Funcish:

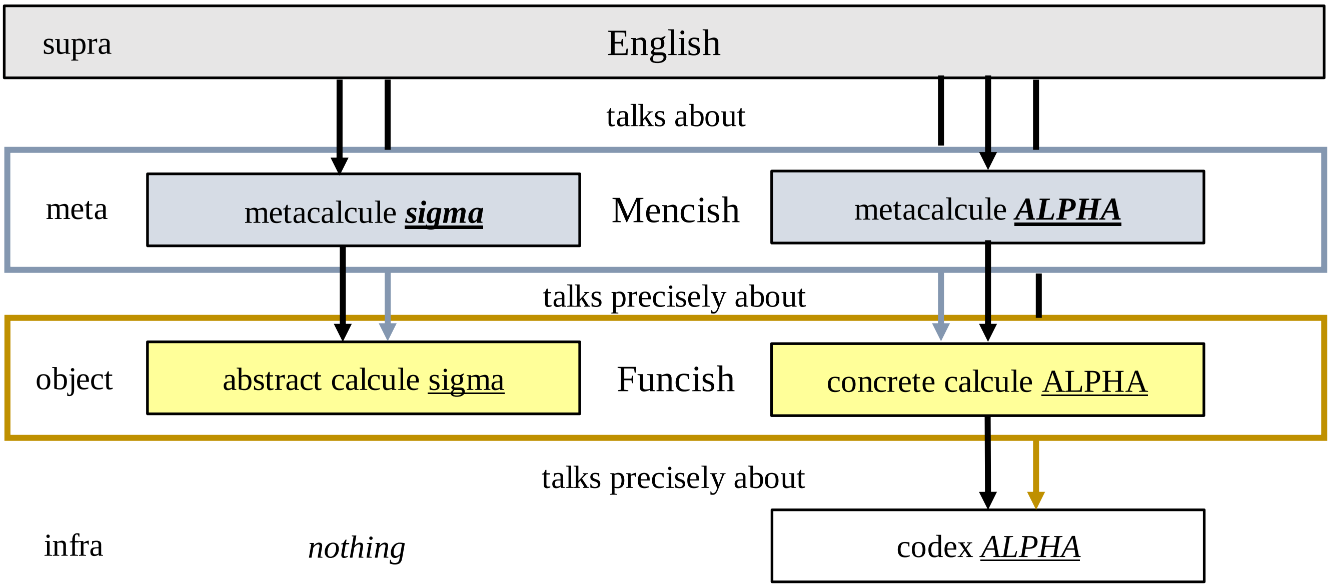

Figure 1 Hierarchy of languages and codices pertinent to the FUME-method for two example calcules, an abstract and a concrete one

A logic with only one tier is not sufficient for the foundation of mathematics. Extending predicate logics to theory of types, introducing axiomatic set theory and other constructions does not solve the problem. One needs at least two tiers, a precise object-language together with a precise metalanguage.

Appendix A Selected basic metaindividuals, metarelations and metafunctions

<u>syntactic metaproperties in general</u> (sort ϕ)

petit-number	string with only 0,1,2,3,4,5,6,7,8,9 (for convention decimals are used)
number	string with only 0,1,2,3,4,5,6,7,8,9 (for convention decimals are used)
number-array	array of ***number*** strings separated by semicolon
variable	***sort*** string followed by ***petit-number***
variable-array	array of ***variable*** strings separated by semicolon
omny	multiple distinct ***omnicle*** strings e.g. $\forall\phi_{2}[\forall\phi_{11}[\forall\phi_{3}[$
pattern	built up from ***function-constant*** strings with ***number*** and ***variable*** strings
term	***pattern*** with ***number*** strings only
scheme	***pattern*** with at least one ***variable*** strings only
phrase	built up from equalities of ***pattern*** strings and from ***relation-constant*** strings using full predicative logic
sentence	***phrase*** with no free ***variable*** strings
formula	***phrase*** with at least one ***variable*** (arity is count of distinct ***variable*** strings), no ϕ_0
formulo	like ***formula*** but with ϕ_0 (which is left out for arity count)

Successive-prime-array array of ***number*** strings, that are successive primes

Ascending-prime-array array of ***number*** strings, that are ascending (not necessarily successive) primes

<u>alethic metaproperties in general</u>

UNEX-formulo	representing a function by a ***formulo*** with unique existence of result ϕ_0 for input
TRUTH	any alethic ***sentence***
THEOREM	quantive alethic ***sentence*** that is not basic
Axiom , ***Basiom***	***sentence*** introduced as basic ***TRUTH*** (in abstract or concrete calcule resp.)

<u>metaindividuals in calcule</u> <u>ALPHA l</u> (sort A)

AXFOgbeta	ternary ***UNEX-formulo*** representing Gödel's beta-function,
AXFOadp	binary ***UNEX-formulo*** representing antidiagonal pair coding
AXFOada	unary ***UNEX-formulo*** representing auxiliary function for antidiagonal pair coding
AXFOadr	unary ***UNEX-formulo*** representing row decoding function of antidiagonal pair
AXFOadc	unary ***UNEX-formulo*** representing column decoding function of antidiagonal pair
AFAprime	unary ***formula*** characterizing ***number*** strings
AFAeupr	unary ***formula*** of products of successive primes, ending at the given argument A_1
AFApripopair	binary ***formula*** , so that A_1 is a prime and A_2 is a power thereof

<u>syntactic binary metarelations in general and in calcule</u> <u>ALPHA</u>

ϕ ≈ ϕ	matching length of strings
ϕ ≺ ϕ	smaller length of strings
ϕ ⊃ ϕ	soutaining (suitably containing, i.e. in a way that avoids disambiguities)
ϕ ⁄ ϕ	suitably-free-in
ϕ ⁄ ϕ	suitably-bound-in
ϕ ~ ϕ	compatible (no collision of bound ***variable*** strings in constructing ***phrase*** strings)
A << A	natural-minority, smaller with respect to numbering by ***number***

<u>syntactic metafunctions in general and in calcule</u> <u>ALPHA</u>

(ϕ&ϕ)	synaption (concatenation except for leading 0)
(ϕ∂ϕ)	character-deletion
(ϕ;ϕ∫ϕ)	string-replacement
A″(A)	succession with respect to ***number*** (10 characters)
A□□(A)	length as ***number***, e.g. ***A□□(*** $\forall A_1[$ ***)*** = 4
A◊◊(A)	arity as ***number***, e.g. ***A◊◊(***$A_{12};A_3;A_1;A_1;A_1$***)*** = 5
A∇∇(A;A)	projection: substring of ***array*** in second place at position with ***number*** in first place

Appendix B Gödel's beta-function and more in abstract Robinson-Crusoe arithmetic

Based on the observation that one only needs the ***UNEX-formulo*** technique for representation of functions in concrete calcule <u>ALPHA</u> of Robinson decimal natural number arithmetic one remembers equation $(x+y)^2=x^2+y^2+2xy$ (in classical notation) to produce an eaven weaker calcule. This time the **abstract** counter piece is introduced. The interesting feature is that one can leave away the binary function **multiplication** ; unary **quadration** is sufficient.

The ontological basis of abstract calcule <u>alphakappa</u> of Robinson-Crusoe natural number arithmetic comprises the following ingredients:

sort ::	$\alpha\kappa$	
basis-individual-constant ::	$\alpha\kappa n$	*nullum*
basis-function-constant ::	$\alpha\kappa'$ ¦ $(\alpha\kappa+\alpha\kappa)$ ¦ $(\alpha\kappa\uparrow)$	*succession, addition, quadration*
basis-relation-constant ::	$\alpha\kappa<\alpha\kappa$	*minority*
extra-individual-constant ::	$\alpha\kappa u=\alpha\kappa n'$	*unus*

Axiom strings

A1 $\forall\alpha\kappa_1[\alpha\kappa_1'\neq\alpha\kappa n]$

A2 $\forall\alpha\kappa_1[\alpha\kappa_2[[\alpha\kappa_1'=\alpha\kappa_2']\rightarrow[\alpha\kappa_1'=\alpha\kappa_2]]]$

A3 $\forall\alpha\kappa_1[[\alpha\kappa_1\neq\alpha\kappa n]\rightarrow[\exists\alpha\kappa_2[\alpha\kappa_1=\alpha\kappa_2']]]$

A4 $\forall\alpha\kappa_1[(\alpha\kappa_1+\alpha\kappa n)=\alpha\kappa_1]$

A5 $\forall\alpha\kappa_1[\alpha\kappa_2[(\alpha\kappa_1+\alpha\kappa_2')=(\alpha\kappa_1+\alpha\kappa_2)']]$

A6 $\forall\alpha\kappa_1[(\alpha\kappa n\uparrow)=\alpha\kappa n]$

A7 $\forall\alpha\kappa_1[(\alpha\kappa_1'\uparrow)=(((\alpha\kappa_1\uparrow)+\alpha\kappa_1)+\alpha\kappa_1)']$

A8 $\forall\alpha\kappa_1[\neg[\alpha\kappa_1<\alpha\kappa n]]$

A9 $\forall\alpha\kappa_1[[\alpha\kappa n=\alpha\kappa_1]\vee[\alpha\kappa n<\alpha\kappa_1]]$

A10 $\forall\alpha\kappa_1[\alpha\kappa_2[[\alpha\kappa_1<\alpha\kappa_2]\leftrightarrow[[\alpha\kappa_1'=\alpha\kappa_2]\vee[[\alpha\kappa_1'<\alpha\kappa_2]]]]$

A11 $\forall\alpha\kappa_1[\alpha\kappa_2[[\alpha\kappa_1<\alpha\kappa_2']\leftrightarrow[[\alpha\kappa_1<\alpha\kappa_2]\vee[[\alpha\kappa_1=\alpha\kappa_2]]]]$

Axiom matres for the unary and multary case of induction:

$\exists\boldsymbol{\alpha\kappa_1}[\,[\,\textbf{\textit{sentence}}(\forall\alpha\kappa_1[\boldsymbol{\alpha\kappa_1}])\,]\rightarrow$
$[\,\textbf{\textit{Axiom}}([[(\boldsymbol{\alpha\kappa_1};\alpha\kappa_1\int\alpha\kappa n)]\wedge[\forall\alpha\kappa_1[[\boldsymbol{\alpha\kappa_1}]\rightarrow[(\boldsymbol{\alpha\kappa_1};\alpha\kappa_1\int\alpha\kappa_1')]]]]\rightarrow[\forall\alpha\kappa_1[\boldsymbol{\alpha\kappa_1}]])\,]\,]$

$\exists\boldsymbol{\alpha\kappa_1}[\,\exists\boldsymbol{\alpha\kappa_2}[\,\exists\boldsymbol{\alpha\kappa_3}[\,[\,[\,[\,\textbf{\textit{formula}}(\boldsymbol{\alpha\kappa_1})\,]\wedge[\,\textbf{\textit{omny}}(\boldsymbol{\alpha\kappa_2})\,]\,]\wedge[\,\textbf{\textit{sentence}}(\boldsymbol{\alpha\kappa_2}\forall\alpha\kappa_1[\boldsymbol{\alpha\kappa_1}]\boldsymbol{\alpha\kappa_3})\,]\,]\rightarrow$
$[\,\textbf{\textit{Axiom}}(\boldsymbol{\alpha\kappa_2}[[(\boldsymbol{\alpha\kappa_1};\alpha\kappa_1\int\alpha\kappa n)]\wedge[\forall\alpha\kappa_1[[\boldsymbol{\alpha\kappa_1}]\rightarrow[(\boldsymbol{\alpha\kappa_1};\alpha\kappa_1\int\alpha\kappa_1')]]]]\rightarrow[\forall\alpha\kappa_1[\boldsymbol{\alpha\kappa_1}]]\boldsymbol{\alpha\kappa_3}\,]\,]\,]\,]$

One uses the following binary ***UNEX-formulo*** for the introduction of multiplication:

$\boldsymbol{\alpha\kappa}\textbf{\textit{XFOmul}} =$ $((\alpha\kappa_1+\alpha\kappa_2)\uparrow)=(((\alpha\kappa_1\uparrow)+(\alpha\kappa_2\uparrow))+(\alpha\kappa_0+\alpha\kappa_0))]$

One has a unary ***formula*** in Robinson-Crusoe arithmetic to express that a ***number*** string is prime:

$\boldsymbol{\alpha\kappa}\textbf{\textit{FAprime}} =$ $\forall\alpha\kappa_{30}[\forall\alpha\kappa_{31}[[[[\alpha\kappa u<\alpha\kappa_{30}]\wedge[\alpha\kappa_{30}<\alpha\kappa_{31}']]\wedge[\alpha\kappa_{31}<\alpha\kappa_1]]\rightarrow$
$[((\alpha\kappa_{30}+\alpha\kappa_{31})\uparrow)\neq(((\alpha\kappa_{30}\uparrow)+(\alpha\kappa_{31}\uparrow))+(\alpha\kappa_1+\alpha\kappa_1)]]]$

As well one can represent Gödel's beta-function in Robinson-Crusoe arithmetic by a ternary ***UNEX-formulo*** using auxiliary bound ***variable*** strings $\alpha\kappa_{21}$ and $\alpha\kappa_{22}$ that are limited by $((\alpha\kappa_1+\alpha\kappa_2)\uparrow)$:
$\exists\alpha\kappa_{21}[((\alpha\kappa_1+\alpha\kappa_2)\uparrow)=(((\alpha\kappa_1\uparrow)+(\alpha\kappa_2\uparrow))+(\alpha\kappa_{21}+\alpha\kappa_{21}))]$
$\exists\alpha\kappa_{22}[((\alpha\kappa_{21}'+\alpha\kappa_{20})=(((\alpha\kappa_{21}'\uparrow)+(\alpha\kappa_{20}\uparrow))+(\alpha\kappa_{22}+\alpha\kappa_{22})]$

$\boldsymbol{\alpha\kappa}\textbf{\textit{XFOgbeta}} =$ $\exists\alpha\kappa_{20}[\exists\alpha\kappa_{21}[\exists\alpha\kappa_{22}[[[[((\alpha\kappa_2+\alpha\kappa_3'\uparrow)=(((\alpha\kappa_2\uparrow)+(\alpha\kappa_3'\uparrow))+(\alpha\kappa_{21}+\alpha\kappa_{21})]\wedge$
$[((\alpha\kappa_{21}'+\alpha\kappa_{20})\uparrow)=(((\alpha\kappa_{21}'\uparrow)+(\alpha\kappa_{20}\uparrow))+(\alpha\kappa_{22}+\alpha\kappa_{22})]]\wedge[(\alpha\kappa_{22}+\alpha\kappa_0)=\alpha\kappa_1]]\wedge[\alpha\kappa_0<(\alpha\kappa_2\times\alpha\kappa_3')']]]]$