Skolem Revisited
Thoralf's Achilles heel

Hannes Hutzelmeyer

Summary

The author has developed an approach to logics that comprises, but also goes beyond predicate logic. The **FUME** method contains two tiers of precise languages: object-language **Funcish** and metalanguage **Mencish**. It allows for a very wide application in mathematics from geometry, number theory, recursion theory and axiomatic set theory with first-order logic, to higher-order logic theory of real numbers and a precise analysis of foundation of mathematics in general, including theory of types.

A famous paper by Thoralf Skolem of 1934 is usually put at the beginning of publications on non-standard arithmetic. A critical investigation shows that it has serious, if not even insurmountable problems. Firstly one notices that it is based on **second-order logic**, it has unary and binary function-variables, and binary operator-constants (that map two functions to a function). It seems strange that one makes a fundamental statement about first-order logic systems using second order.

In proving *Satz 1* on the asymptotic behavior of arithmetic functions Skolem has some inaccuracies and formal errors. These minor problems can be solved by diligent work. But even if one has replaced his metalingual use of his relation symbols $\boldsymbol{B_i}$ by an **ontologically** correct method there remains secondly the problem of **transitivity** of the minority relation of functions that is neglected by Skolem.

Thirdly, in constructing the strictly ascending function *g* of *Satz 1* use is made of **recursion** by a dot-dot-dot notation. This is not an admissible procedure in object-language, although there may be a correct way to solve the problem in metalanguage.

Therefore one does not only need second-order logic in combination with a precise object-language (in order to avoid ontology problems) but also a precise use of metalanguage (in order to avoid dot-dot-dot) for justifying Skolem's *Satz 1* after one has eventually solved the transitivity problem; Skolem's *Satz 2* would then be valid. However, as long as *Satz 1* is not confirmed it does not pay to treat *Satz 3* and *Satz 4*, leaving open the existence of non-standard models of arithmetic on the basis of Skolem's work.

Warning

The author has taken the liberty of changing the usual meaning of some expressions and to coin some new expressions. He has tried to mark these appearances properly; if he has failed to do so in some places he asks for your understanding. Your comments are welcome.

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

1. FUME-method object-language and metalanguage

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 $= \neq \neg \vee \wedge \ \rightarrow \leftrightarrow$ quantors $\exists$ $\forall$ and variables like e.g. A_1 or A_{13} were introduced together with the rules for omnition $\forall A_1$[...] and entition $\exists A_2$[...] as well as ***relation-constant*** and ***function-constant*** strings that allowed for expressing logic-sentences in a proper fashion. Freges notation differs from this modern form, but that is irrelevant.

The author has put forward a **precise** system of **object-language** and **metalanguage** that overcomes certain difficulties of predicate logic and that can be extended to a full theory of **types** . In order to describe an **object-language** one also needs a **metalanguage**. According to the author's principle metalanguage has to be absolutely precise as well, normal English will not do. The **FUME**-method contains at least three tiers of language:

Funcish	object-language	formalized precise
Mencish	metalanguage	formalized precise
English	supralanguage	common, mostly not precise

'Calcule' is the name given to a mathematical system with the precise language-metalanguage method Funcish-Mencish . 'Calcule' is an expression coined by the author in order to avoid confusion. The word 'calculus' is conventionally used for real number mathematics and various logical systems. As a German translation 'Kalkul' is proposed for 'calcule' versus conventional 'Kalkül' that usually is translated as 'calculus'.

A **concrete** calcule talks about a **codex** of concrete **individuals** (given as strings of characters) and concrete **functions** and **relations** that can be realized by 'machines' (called calculators). An **abstract** calcule talks about **nothing**. It only says: if some entities exist with such and such properties they also have certain other properties. Essentially there are only 'if-then' statements. E.g. 'if there are entities that obey the Euclid axioms the following sentence is true for these entities' .

Calcules with first-order logic FOL are called **haplo-calcules** , calcules with higher-order logic HOL for a theory of types are called **hypso-calcules**. An **abstract** calcule is based on a finite count or on enumerably many axioms as opposed to a **concrete** calcule whose foundation can be put into practice by a machine. ***Axiom*** strings are certain ***sentence*** strings, they can also be provided with a metalingual ***Axiom*** **mater** (rather than the usual 'scheme' or 'schema', as the expression ***scheme*** has a different meaning in Mencish), that produce enumerably many ***Axiom*** strings.

In supralanguage English calcules are given names based on the Greek ***sort*** names. They are constructed from the ***sort*** strings that appear in it, using the Latin names of the Greek letters of object language Funcish. Concrete calcule ***sort*** strings have all-capital-letter or all-capital-letter-onset words, abstract calcules have small-letter words. The first letter of the first ***sort*** name classifies a calcule according to some convention rules.The ***sort*** string names are separated by blank and completely underlined. Higher-order logic calcules are characterized by a final letter python[1)] ϖ separated by a blank. Python is also the final character in relevant ***sort*** strings. Examples:

abstract calcules

. alpha with ***sort*** α

concrete calcule

- ALPHA with ***sort*** A
- ALPHApython with ***sort*** Aϖ

1) Distinguish the Greek characters: *capital pi* Π , *small pi* π and *python* ϖ (pronounce as in 'Monty Python') .

The **fonts-method** allows to distinguish between object-language (Arial and Symbol, normal, e.g. $\forall\alpha_1[\ )$, metalanguage (Arial and Symbol, boldface italics e.g. $\boldsymbol{\alpha_1}$ or ***Axiom***) and supralanguage **English** (Times New Roman).

For quoting Skolem and sometimes for better understanding mathematical expressions are written **conventionally** in supralanguage English using italics (heuristically, not fully precise) e.g. :

- numbers *0 1 2 …* constants *a b …* variables *i j m n t x y …* symbols *+ - / (,) = < >* multiplication *xy*
- a series of unary functions f_0 f_1 f_2 … with values at *x* denoted as $f_0(x)$ $f_1(x)$ $f_2(x)$ …
- the value of binary function *f* with values at *x,y* denoted as *f(x,y)*
- a binary operator *B* with value at unary functions *f* and *g* denoted as *B(f,g)*
  and its value at *x* denoted as *B(f,g)(x)* , composition of *f* and *g* denoted as $(f\,^o g)(x)=f(g(x))$
- the appearance of a variable e.g. *t* in an expression means 'for all *t* ' .

In supralanguage English metacalcules are given names that correspond uniquely to their object-calcules: the Times New Roman fonts are chosen as boldface italic, otherwise they are the same.

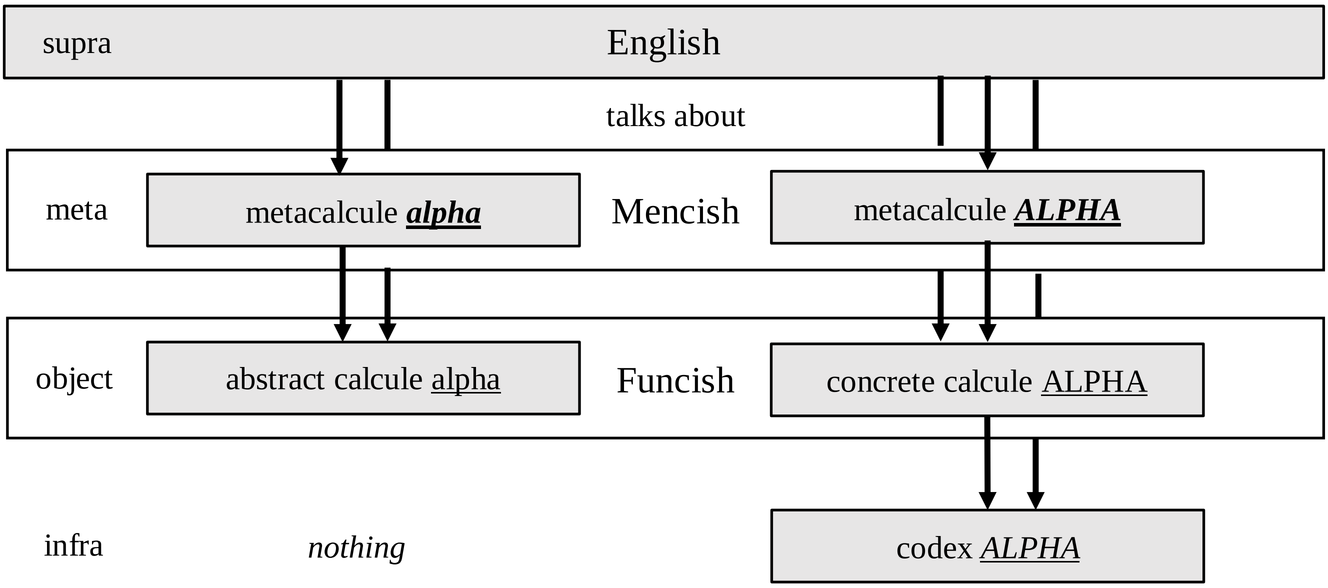

Figure 1 Hierarchy of languages and codices for two example calcules

Metalanguage **Mencish** is chosen with **perfect exactness**, just as object-language Funcish. They both have to meet the **calculation criterion of truth**: every step of reasoning must be such that it can be checked by a calculating machine. Funcish and Mencish sentences and metasentences resp.are understandable without context: 'wherefore by their *words* ye shall know them' (vs. Mathew 7.20).

On first sight Funcish and Mencish look familiar to what one knows from predicate-logic. However, they are especially adapted to a degree of precision so that they can be used universally for all kind of mathematics. And they lend themselves immediately to a treatment by computers, as they have perfect syntax and semantics. It is not the place to go into details. Both Funcish and Mencish have essentially the same syntax. Notice that Funcish has a context-independent notation, which implies that one can determine the **category** of every object uniquely from its syntax. The reader may be puzzled by some expressions that are either newly coined by the author or used slightly different from convention. This is done in good faith; the reason for the so-called **Bavarian notation** is to avoid ambiguities.

The essential parts of a language are its **sentences**. A ***sentence*** is a **string** of **characters** of a given **alphabet** that fulfills certain syntactical and semantical rules. This means that metalanguage talks about the strings of the object-language. The essential parts of the metalanguage are the metasentences (that are strings of characters as well, just in boldface italics). In supralanguage one talks about the metasentences, just as metalanguage talks about object-language. Here it is not discussed in general what an object-language talks **about** .

2. Concrete hypso-calcule ALPHApython *of Monty-arithmetic with second-order logic*

In 1934 Thoralf Skolem put forward his famous paper[1] that shows how to construct non-standard models of arithmetic. Skolem's shortcoming is his use of second-order logic. In the world of denumerability second-order logic has no rightful place! Can one remove this bond ? The language system of the FUME-method is a theory of types and therefore capable of expressing higher-order logic precisely. So the first thing to do is to to express Skolem's theorems *Satz 1* , *Satz 2* , *Satz 3* and *Satz 4* in proper language.

For further discussion one has to take refuge to examples . Although Skolem uses second-order logic, he only regards entities that are at most denumerable. It will be investigated to what extent this justifies Skolem's reasoning. Here we do not use the full machinery of Mencish by whose application everything can be done with absolute preciseness. We just write down the ***Basiom*** strings in the Appendix , that are basic true ***sentence*** strings (that are obtained by observing the defining ***aponom*** strings - not treated here) and correspond to ***Axiom*** strings of abstract calcules.

The concrete calcule ALPHA of arithmetic of naturals can be set up properly, here just its ontological basis is sketched. It is a haplo-calcule as it has first-order logic.

sort ::	A	
individual :: ***natural ::***	0 ¦ 1 ¦ 2 …	***nullum***, ***unus***, ***duo***, …
individual-constant ::	An ¦ Au ¦ Ab …	
basis-function-constant ::	(A+A) ¦ (A×A) ¦	addition, multiplication
	(A÷A)	trunctraction (truncated subtraction)
basis-relation-constant ::	A<A	minority
extra-function-constant ::	A′ ¦ (A÷)	succession, predecssion
extra-relation-constant ::	A≤A	equal-minority

Starting from haplo-calcule ALPHA of arithmetic of naturals the concrete hypso-calcule ALPHApython[2] of Monty-arithmetic of naturals is introduced. It will be the system for a reconstruction of Skolem's ideas in a precise fashion.

ontological basis is

sort ::	Aϖ	
sort-array ::	Aϖ ¦ Aϖ;Aϖ ¦ Aϖ;Aϖ;Aϖ …	unary, binary, ternary, …
type ::	(Aϖ) ¦ (Aϖ;Aϖ) …	property, binary relation, …
	Aϖ(Aϖ) ¦ Aϖ(Aϖ;Aϖ) ¦	unary function, binary function, …
	(Aϖ(Aϖ);Aϖ(Aϖ))	binary unary-function-predicate
basis-function-constant ::	(Aϖ+Aϖ) ¦ (Aϖ×Aϖ) ¦	addition, multiplication
	(Aϖ÷Aϖ)	trunctraction
basis-relation-constant ::	Aϖ<Aϖ	minority
extra-function-constant ::	Aϖ′ ¦ (A÷)	succession, predecession
extra-relation-constant ::	Aϖ≤Aϖ	equal-minority

There are two ways to look at unary arithmetic functions. With second-order logic a unary function is anything formally introduced such that it produces a unique output value for an input value (one can say 'for all functions' and 'there exist a function' . With first-order logic the best one can do is to have a binary ***function-constant*** and treat the first argument as a parameter so that one can say 'for all functions of a series' .

[1] Skolem, Thoralf "Über die Nicht-charakterisierbarkeit der Zahlenreihe mittels endlich oder abzählbar unendlich vieler Aussagen mit ausschliesslich Zahlenvariablen" *Fundamenta Mathematicae T. XXIII* (1934) p. 150-161

[2] pronounce as in Monty Python

3. *Skolem's Satz 1 in precise language*

Skolem's *Satz 1* is translated as follows: For a binary arithmetic function[1] *f(x,y)* (Skolem calls it a series[2] $f_0(t)$, $f_1(t)$, $f_2(t)$ …) there is a strictly ascending function[1] *g(y)* and a binary arithmetic function[1] *t(x,y)* such that for any pair of first arguments *x=a* for *x=b* it holds that the two unary functions of *y* i.e. *f(a,g(y))* and *f(b,g(y))* are either equal or minor or major to each other for all *y* greater than *t(a,b)* .

Observation 1: Skolem does not specify what he considers to be an arithmetic function. It is trivial that the three possibilities are mutually exclusive. It is not stated that there is only one functions *g* , different functions *g* may lead to different ordering of functions of the series. For a chosen unary function *g* there is one binary *t* that gives the lowest limit, all binary *s* with greater values *t(x,y)< s(x,y)*would do as well.

It reads as ***sentence*** in ALPHApython , with series written as Aϖ1(0;Aϖ) , Aϖ1(1;Aϖ) , Aϖ1(2;Aϖ) …

AϖS-Skolem1= ∀Aϖ1(Aϖ;Aϖ)[∃Aϖ1(Aϖ)[
[∀Aϖ1[∀Aϖ2[[Aϖ1<Aϖ2]→[Aϖ1(Aϖ1)<Aϖ1(Aϖ2)]]]]∧[∃Aϖ2(Aϖ;Aϖ)[∀Aϖ1[∀Aϖ2[[
[∀Aϖ3[[Aϖ2(Aϖ1;Aϖ2)<Aϖ3]→[Aϖ1(Aϖ1;Aϖ1(Aϖ3))=Aϖ1(Aϖ2;Aϖ1(Aϖ3))]]]]∨
[∀Aϖ3[[Aϖ2(Aϖ1;Aϖ2)<Aϖ3]→[Aϖ1(Aϖ1;Aϖ1(Aϖ3))<Aϖ1(Aϖ2;Aϖ1(Aϖ3))]]]]∨
[∀Aϖ3[[Aϖ2(Aϖ1;Aϖ2)<Aϖ3]→[Aϖ1(Aϖ2;Aϖ1(Aϖ3))<Aϖ1(Aϖ1;Aϖ1(Aϖ3))]]]]]]]]]

What kind of animals are Aϖ1(Aϖ) so that one can call one of them AϖG(Aϖ) ? Can it be contained in Aϖ1(Aϖ;Aϖ) by a special Aϖg ? What kind of animals are Aϖ2(Aϖ;Aϖ) so that one can call the smallest AϖT(Aϖ;Aϖ) ? Can it be constructed from Aϖ1 . Skolem starts with auxiliary ***sentence*** :

4. *Auxiliary sentence for function pair comparison*

For two unary arithmetic functions *a* and *b* there is a strictly ascending function *g* such that along *g(t)* the **composed** functions $(a^o g)$ and $(b^o g)$ with values *a(g(t))* and *b(g(t))* are equal ***B***:=, minor ***B***:< or major ***B***: > to each other: *a(g(t))* ***B*** *b(g(t))* [3] ; the binary operator **composition** is written as $a(g(t))=(a^o g)(t)$. Actually it is meant that there is a binary operator *G* that maps two functions *a* and *b* to function *g* , i.e. *g=G(a,b)* and thus *g(t)=G(a,b)(t)* . Notice that ***B*** is a strange metalingual animal that has yet to be translated into a proper form so that one can use it in proper language.

AϖS-Skolem1a= ∀Aϖ1(Aϖ)[∀Aϖ2(Aϖ)[
[∃Aϖ3(Aϖ)[[∀Aϖ1[∀Aϖ2[[Aϖ1<Aϖ2]→[Aϖ3(Aϖ1)<Aϖ3(Aϖ2)]]]]∧
[[[∀Aϖ1[Aϖ1(Aϖ3(Aϖ1))=Aϖ2(Aϖ3(Aϖ1))]]]∨[∀Aϖ1[Aϖ1(Aϖ3(Aϖ1))<Aϖ2(Aϖ3(Aϖ1))]]]]∨
[∀Aϖ1[Aϖ2(Aϖ3(Aϖ1))<Aϖ1(Aϖ3(Aϖ1))]]]]]]]

Observation 2: What kind of animal is Aϖ3(Aϖ) and how do you construct it ? Three cases can be defined by three binary unary-function-predicates (that include unlimited enticles ∃Aϖ2[ ). The infinity condition in connection with ∃Aϖ2[cannot be answered in general without having further information on Aϖ1(Aϖ) and ∀Aϖ2(Aϖ) , a problem not considered by Skolem.

∀Aϖ1(Aϖ)[∀Aϖ2(Aϖ)[[EQG(Aϖ1(Aϖ);Aϖ2(Aϖ))]↔
[∀Aϖ1[∃Aϖ2[[Aϖ1<Aϖ2]∧[Aϖ1(Aϖ2)=Aϖ2(Aϖ2)]]]]]]
∀Aϖ1(Aϖ)[∀Aϖ2(Aϖ)[[MIG(Aϖ1(Aϖ);Aϖ2(Aϖ))]↔
[∀Aϖ1[∃Aϖ2[[Aϖ1<Aϖ2]∧[Aϖ1(Aϖ2)<Aϖ2(Aϖ2)]]]]]]
∀Aϖ1(Aϖ)[∀Aϖ2(Aϖ)[[MAG(Aϖ1(Aϖ);Aϖ2(Aϖ))]↔[
[∀Aϖ1[∃Aϖ2[[Aϖ1<Aϖ2]∧[Aϖ2(Aϖ2)<Aϖ1(Aϖ2)]]]]]]

[1] usual sloppy conventional notation, the name of the function is *f* and its values at *x* and *y* are *f(x,y)*

[2] Skolem starts with index *1*

[3] in conventional notation the appearance of a variable *t* in an expression means 'for all *t* ' (in proper notation with ∀)

Skolem then has added the auxiliary ***sentence*** $\boldsymbol{A\varpi}$***S-Skolem1a*** , but is it really a ***THEOREM*** ?

Observation 3: The three cases can be **mutually exclusive** but they are are **not always** mutually exclusive as the example in classical notation shows where *e(x)* denotes the **entirition** function for rounding-down

take for	$A\varpi_1(A\varpi)$	function	*x-3e(x/3)*	*0 1 2 0 1 2 …*
and for	$A\varpi_2(A\varpi)$	function	*2-(x-3e(x/3))*	*2 1 0 2 1 0 …*

Three choices of strictly ascending $A\varpi_3(A\varpi)$ allow for all of the three possibilities:

$A\varpi MIG(A\varpi)$	*3x*	*0 3 6 9 12 15 …*	minority-infinition
$A\varpi EQG(A\varpi)$	*3x+1*	*1 4 7 10 13 16 …*	equality-infinition
$A\varpi MAG(A\varpi)$	*3x+2*	*2 5 8 11 14 17 …*	majority-infinition

Determine $A\varpi_3(A\varpi)$ recursively as $A\varpi_0(A\varpi)$ for any of the three choices, say equality-infinity. It is always defined as it is put to 0 if the predicate is not met.

$$\forall A\varpi_1(A\varpi)[\forall A\varpi_2(A\varpi)[\exists A\varpi_0(A\varpi)[[[\neg[EQG(A\varpi_1(A\varpi);A\varpi_2(A\varpi))]]\rightarrow[\forall A\varpi_1[A\varpi_0(A\varpi_1)=0]]]\wedge$$
$$[[EQG(A\varpi_1(A\varpi);A\varpi_2(A\varpi))]\rightarrow[[\forall A\varpi_1[[[A\varpi_1(A\varpi_1)=A\varpi_2(A\varpi_1)]\wedge$$
$$[\forall A\varpi_2[[A\varpi_1(A\varpi_2)=A\varpi_2(A\varpi_2)]\rightarrow[A\varpi_1\leq A\varpi_2]]]]\rightarrow[A\varpi_0(0)=A\varpi_1]]]\wedge$$
$$[\forall A\varpi_1[\forall A\varpi_2[[[[A\varpi_0(A\varpi_1)<A\varpi_2]\wedge[A\varpi_1(A\varpi_2)=A\varpi_2(A\varpi_2)]]\wedge$$
$$[\forall A\varpi_3[[[A\varpi_0(A\varpi_1)<A\varpi_3]\wedge[A\varpi_1(A\varpi_3)=A\varpi_2(A\varpi_3)]]\rightarrow[A\varpi_2\leq A\varpi_3]]]]\rightarrow[A\varpi_0(A\varpi_1')=A\varpi_2]]]]]]]]$$

Minority-infinity $A\varpi MIG(A\varpi(A\varpi);A\varpi(A\varpi);A\varpi)$ and
majority-infinity $A\varpi MAG(A\varpi(A\varpi);A\varpi(A\varpi);A\varpi)$ are defined accordingly.

Observation 4: This is a ***formulo*** with input $A\varpi_1(A\varpi)$ and $A\varpi_2(A\varpi)$ for output $A\varpi_0(A\varpi)$. But how do you prove that it is a ***UNEX-binary-formulo*** that gives rise to an operator adjection $A\varpi EQG(A\varpi(A\varpi);A\varpi(A\varpi);A\varpi)$? Similarily for adjections $A\varpi MIG(A\varpi(A\varpi);A\varpi(A\varpi);A\varpi)$ and $A\varpi MAG(A\varpi(A\varpi);A\varpi(A\varpi);A\varpi)$. Notice that the adjections produce the zero-function if the conditions are not fulfilled and therefor not ascending. The simple test is e.g. $A\varpi EQG(A\varpi_1(A\varpi);A\varpi_2(A\varpi);1)=0$

Observation 5: One needs an algorithm for the unique definition of $A\varpi_3(A\varpi)$. Firstly define a simple ternary relation **comparity** $CB(A\varpi;A\varpi;A\varpi)$ as a mere junctive abbreviation

$$[CB(A\varpi_1;A\varpi_2;A\varpi_3)]\leftrightarrow$$
$$[[[[A\varpi_3=0]\wedge[A\varpi_1=A\varpi_2]]\vee[[A\varpi_3=1]\wedge[A\varpi_1<A\varpi_2]]]\vee[[1<A\varpi_3]\wedge[A\varpi_2<A\varpi_1]]]$$

Comparing adjection $A\varpi B(A\varpi_1(A\varpi);A\varpi_2(A\varpi);A\varpi_1)=0$ or 1 or 2

Secondly define operator adjections **comparison-selection** $A\varpi G(A\varpi(A\varpi);A\varpi(A\varpi);A\varpi)$ (Skolem's *g(t)*) and **comparison-codification** $A\varpi B(A\varpi(A\varpi);A\varpi(A\varpi);A\varpi)$ (for Skolem's *B*) that also take care of the case that there can be more than one possibility of equality, minority and majority. A first choice would be: prefer firstly equality and secondly minority:

$$\forall A\varpi_1(A\varpi)[\forall A\varpi_2(A\varpi)[\forall A\varpi_1[[$$
$$[[A\varpi EQG(A\varpi_1(A\varpi);A\varpi_2(A\varpi);1)\neq0]\rightarrow$$
$$[[A\varpi G(A\varpi_1(A\varpi);A\varpi_2(A\varpi);A\varpi_1)=A\varpi EQG(A\varpi_1(A\varpi);A\varpi_2(A\varpi);A\varpi_1)]\wedge$$
$$[A\varpi B(A\varpi_1(A\varpi);A\varpi_2(A\varpi);A\varpi_1)=0]]]\vee$$
$$[[[A\varpi EQG(A\varpi_1(A\varpi);A\varpi_2(A\varpi);1)=0]\wedge[A\varpi MIG(A\varpi_1(A\varpi);A\varpi_2(A\varpi);1)\neq0]]\rightarrow$$
$$[[A\varpi G(A\varpi_1(A\varpi);A\varpi_2(A\varpi);A\varpi_1)=A\varpi MIG(A\varpi_1(A\varpi);A\varpi_2(A\varpi);A\varpi_1)\wedge$$
$$[A\varpi B(A\varpi_1(A\varpi);A\varpi_2(A\varpi);A\varpi_1)=1]]]\vee$$
$$[[[A\varpi EQG(A\varpi_1(A\varpi);A\varpi_2(A\varpi);1)=0]\wedge[A\varpi MIG(A\varpi_1(A\varpi);A\varpi_2(A\varpi);1)=0]]\rightarrow$$
$$[[A\varpi G(A\varpi_1(A\varpi);A\varpi_2(A\varpi);A\varpi_1)=A\varpi MAG(A\varpi_1(A\varpi);A\varpi_2(A\varpi);A\varpi_1)\wedge$$
$$[A\varpi B(A\varpi_1(A\varpi);A\varpi_2(A\varpi);A\varpi_1)=2]]]]]]$$

5. *Problems for the auxilary function*

Now there is a uniquely defined function, but there might result problems furtherdown, if one uses this algorithm.

Observation 6: On the way to an ordering of functions that are supposed to constitue non-standard arithmetics it is aimed for some kind of equality and minority of two functions. It is based on a comparison-selection and usual equality and minority of the two functions along this line. With the above method it is not guaranteed that the choice of comparison-code is such that the relation is antisymmetric. One can circumnavigate the problem if one defines minority or majority only with respect to *i* and *j* where *i<j* and adds the reflected case by negation.

i\j	0	1	2	3	4	5	6	7	8	
0		0	1	3	6	10	15	21	28	...
1			2	4	7	11	16	22	29	...
2				5	8	12	17	23	30	...
3					9	13	18	24	31	...
4						14	19	25	32	...
5							20	26	33	...
6								27	34	...
7									35	...
8										...
	...	...	...	...	...	...	...	...	...	...

succession of pairs i≺j , example 3,7
reflect for antisymmetry ~ to ~ ≺ to ≻ ≻ to ≺

0	1	2	3	4	5	6	7	8	
0	1	1	0	4	3	2	2	6	...
2	0	1	2	1	1	5	2	1	...
2	2	0	1	6	5	4	3	2	...
0	1	2	0	2	1	2	2	6	...
3	2	6	1	0	6	0	5	5	...
4	2	5	2	6	0	2	1	3	...
1	5	3	1	0	1	0	0	6	...
1	1	4	1	5	2	0	0	5	...
6	2	1	6	5	4	6	5	0	...
...	...	...	...	...	...	...	...	...	...

distribution of code possibilities
0 for~ 1 for≺ 2 for≻ 3 for~≺ 4 for~≻ 5 for≺≻ 6 for~≺≻

	0	1	2	3	4	5	6	7	8	
0	~	≺	≺	~	≻	≺	≻	≻	≺	...
1	≻	~	≺	≻	≺	≺	≺	≻	≺	...
2	≻	≻	~	≺	≺	≺	~	≻	≺	...
3	~	≺	≻	~	≻	≺	≻	≻	≺	...
4	≺	≻	≻	≺	~	~	~	≺	≻	...
5	≻	≻	≻	≻	~	~	≻	≺	~	...
6	≺	≻	~	≺	~	≺	~	~	≺	...
7	≺	≺	≺	≺	≻	≻	~	~	≻	...
8	≻	≻	≻	≻	≺	~	≻	≺	~	...
	...	...	...	...	...	...	...	...	...	...

distribution of cases
however: 1≺4 and 4≺7 but 1≻7

0	1	2	3	4	5	6	7	8	
0	1	1	0	2	1	2	2	1	...
	0	1	2	1	1	1	2	1	...
		0	1	1	1	0	2	1	...
			0	2	1	2	2	1	...
	2			0	0	0	1	2	...
					0	2	1	0	...
						0	0	1	...
	1			2			0	2	...
								0	...
...	...	...	...	...	...	...	...	...	...

comparison-codification b(i,j)
0 for ~ 1 for ≺ 2 for ≻

Observation 7: Furthermore: with the above method it is not yet proven that the choice of comparison-code is such that the relation is transitive. One has yet to show that there is an algorithm for choosing the code for ambiguous cases so that transitivity is guaranteed!

One can rewrite ***AϖS-Skolem1a*** with AϖG(Aϖ1(Aϖ);Aϖ2(Aϖ);Aϖ) for the desired Aϖ3(Aϖ) as

AϖS-Skolem1aa = ∀Aϖ1(Aϖ)[[∀Aϖ2(Aϖ)[∀Aϖ1[∀Aϖ2[[Aϖ1<Aϖ2]→
[AϖG(Aϖ1(Aϖ);Aϖ2(Aϖ);Aϖ1)<AϖG(Aϖ1(Aϖ);Aϖ2(Aϖ);Aϖ2)]]]]]∧
[∀Aϖ3[[CB(Aϖ1(AϖG(Aϖ1(Aϖ);Aϖ2(Aϖ);Aϖ3)); AϖB(Aϖ1(Aϖ);Aϖ2(Aϖ);Aϖ3));
Aϖ2(AϖG(Aϖ1(Aϖ);Aϖ2(Aϖ);Aϖ3))]]]]]

where AϖG(Aϖ1(Aϖ);Aϖ2(Aϖ);Aϖ3) is Skolem's *G(a,b)(t)* depending on two functions *a* and *b* with values *a(t)* and *b(t)* resp. and a chosen *B* according to preference algorithm, that is now properly expressed by AϖB(Aϖ1(Aϖ);Aϖ2(Aϖ);Aϖ3)

6. Auxiliary theorem of pair ordering

Skolem then adds another ***sentence*** that is actually a ***THEOREM*** , but still one has to improve his choice.

The pairs *(i,j)* of numbers can be arranged as a series by antidiagonal pair-coding *adpair(i,j)=i+((i+j)(i+j+1))/2* . The reverse functions *adrow* and *adcol* are expressed with auxiliary-root function *aux(n)=e(((e(²rt(8n+1)-1)/2)* by means of *adrow(n)=e(n-(aux(n)(aux(n)+1))/2)* and *adcol(n)=e(((aux(n)+1)(aux(n)+2))/2-(n+1))* using entirition *e(x)* .

AϖS-Skolem1b= ∀Aϖ1[∀Aϖ2[∃Aϖ0[(Aϖ0+Aϖ0)=((Aϖ1+Aϖ1)+((Aϖ1+Aϖ2)×(Aϖ1+Aϖ2′)))]]] determines antidiagonal-pair function AϖPAI(Aϖ;Aϖ) or Cantor pairing function and the inverses for row and column AϖADROW(Aϖ) and AϖADCOL(Aϖ) resp. using AϖAUX(Aϖ)

∀Aϖ1[∃Aϖ0[[(8×Aϖ1)′≤((4×(Aϖ0×Aϖ0))′×(4×(Aϖ0×Aϖ0))′)]∧ [((4×(Aϖ0×Aϖ0))′×(4×(Aϖ0×Aϖ0))′)<(8×Aϖ1)′′]]] AϖAUX(Aϖ)

∀Aϖ1[∃Aϖ0[∃Aϖ2[[[(8×Aϖ1)′≤((4×(Aϖ2×Aϖ2))′×(4×(Aϖ2×Aϖ2))′)]∧ [((4×(Aϖ2×Aϖ2))′×(4×(Aϖ2×Aϖ2))′)<(8×Aϖ1)′′]]∧[((Aϖ0+Aϖ0)+(Aϖ2×Aϖ2′))=(Aϖ1+Aϖ1)]]]] for AϖADROW(Aϖ)

∀Aϖ1[∃Aϖ0[∃Aϖ2[[[(8×Aϖ1)′≤((4×(Aϖ2×Aϖ2))′×(4×(Aϖ2×Aϖ2))′)]∧ [((4×(Aϖ2×Aϖ2))′×(4×(Aϖ2×Aϖ2))′)<(8×Aϖ1)′′]]∧[(Aϖ2′×Aϖ2′′)=(2×(Aϖ0+Aϖ1′))]]]] for AϖADCOL(Aϖ)

Actually it would be better to denumerate only the pairs *(i,j)* of numbers with *i<=j* , as the denumeration of pairs will only be applied to pairs where the diagonal-symmetric pairs are either equal or anti-symmetric with pair coding *dspair(i,j)=i+(j/(j+1))/2* and the reverses with
j=dscol(n)=aux(n) and *i=dsrow(n)=n-((aux(n)(aux(n)+1))/2)*

AϖS-Skolem1c= ∀Aϖ1[∃Aϖ0[[(2×Aϖ1)≤(Aϖ0×Aϖ0′)]∧[(Aϖ0×Aϖ0′)<(2×Aϖ1′)]]]

AϖDSCOL(Aϖ)=AϖDSAUX(Aϖ)

(Aϖ1)∀Aϖ1[∃Aϖ0[(2×Aϖ1)=((2×Aϖ0)+(AϖDSAUX(Aϖ1)×(AϖDSAUX(Aϖ1)′)) for AϖDSROW(Aϖ)

Skolem does not treat the problem if the above ***formulo*** strings are ***UNEX-*** which is necessary for introducing the corresponding functions by implicit definition.

∀Aϖ1[[[UNEX-unary-norm-formulo(Aϖ1)] ↔
[[[unary-norm-formulo(Aϖ1)] ∧ [∃Aϖ2[[variable(Aϖ2)] ∧ [¬ [Aϖ1⊃ Aϖ2]]]]] ∧
[TRUTH(∀Aϖ1[∃Aϖ0[[***Aϖ1***]∧[∀***Aϖ2***[[***(Aϖ1;***Aϖ0***∫Aϖ2)***]→[***Aϖ2***=Aϖ0]]]]] ***)]]]***

Implicit definition is justified in second-order logic by second-order ***Axiom***

∀Aϖ1[[UNEX-unary-norm-formulo(Aϖ1)] →
[TRUTH(∃Aϖ1(Aϖ)[[[∀Aϖ1[∀Aϖ0[[***Aϖ1***]↔[Aϖ1(Aϖ1)=Aϖ0]]]]∧
[∀Aϖ2(Aϖ)[∀Aϖ1[∀Aϖ0[[***Aϖ1***]↔[Aϖ2(Aϖ1)=Aϖ0]]]]]]→[∀Aϖ1[Aϖ1(Aϖ1)=Aϖ2(Aϖ1)]]] ***)]]***

Observation 8: This means that one has to make sure that ALPHApython is strong enough to prove that the three ***formulo*** strings are ***UNEX*** . This should be the least problem, Skolem just has not realized that there might be a problem.

7. *Ontological problem and metalingual detours*

Returning to Skolem's proof for ***AϖS-Skolem1*** one has to apply ***AϖS-Skolem1aa*** to Aϖ1(0;Aϖ) , Aϖ1(1;Aϖ) , Aϖ1(2;Aϖ) , … as Skolem's $f_0(t)$, $f_1(t)$, $f_2(t)$ …

$a(g(t))\ \mathbf{B}\ b(g(t))$ hence $a(G(a\ ,b)(t))\ \mathbf{B}(a,b)\ b(G(a,b)(t))$
$f_0(G(f_0, f_1)(t))\ \mathbf{B}(f_0, f_1)\ f_1(G(f_0, f_1)(t))$ hence $(f_0{}^oG(f_0, f_1))(t)\ \mathbf{B}_{0,1}\ (f_1{}^oG(f_0, f_1))(t)$

If one translates from conventional language into Funcish using the FUME -method one has a the first step for $g=G(f_0, f_1)$

AϖG(Aϖ1(0;AϖG(Aϖ1(0;Aϖ);Aϖ1(1;Aϖ);Aϖ1));Aϖ1(1;AϖG(Aϖ1(0;Aϖ);Aϖ1(1;Aϖ);Aϖ1));Aϖ)

One gets a series of series of ***sentence*** strings. For better understanding it is done in conventional language.

Observation 9: However, notice that the animals $\mathbf{B}_{i,j}$ (one of the characters = < >) are not properly expressible in object-language! Notice furthermore that **dot-dot-dot** is not proper object-language either! Be it first-order or higher-order logic.

The first problem cannot be avoided in conventional language. Only with proper FUME and using ternary comparity CB(Aϖ;Aϖ;Aϖ) and comparison-codification AϖB(Aϖ(Aϖ);Aϖ(Aϖ);Aϖ) one can express it properly. The second problem necessitates a detour to metalanguage, as one has no recursion in calcule ALPHApython but only in metacalcule ***ALPHApython*** .

8. *Conventional reconstruction of Skolem's construction*

On page 152 and 153 of Skolem's paper there are inaccuracies and even misleading uses of the same expressions for different entities. This is corrected for in the following four tables. The problem dot-dot-dot like in $(c_0{}^o(c_1{}^o(c_2{}^o\ \ldots\ (c_{n-1}{}^oc_n))))$ is ignored - recursion of composition is not admissible in object-language. The $\mathbf{B}_{i,j}$ problem as described in the preceding section is not resolved either.

$g_{0,0}=G(f_0,f_1)$
$g_{0,1}=G((f_0{}^og_{0,0}),(f_1{}^og_{0,0}))=G((f_0{}^oG(f_0,f_1)),(f_1{}^oG(f_0,f_1)))$
$g_{0,2}=G((f_0{}^og_{0,1}),(f_2{}^og_{0,1}))=G((f_0{}^oG((f_0{}^oG(f_0,f_1)),(f_1{}^oG(f_0,f_1))))),(f_2{}^oG((f_0{}^oG(f_0,f_1)),(f_1{}^oG(f_0,f_1))))))$
…
$g_{0,j}=G((f_0{}^og_{0,j-1}),(f_j{}^og_{0,j-1}))$
$g_{0,j+1}=G((f_0{}^og_{0,j}),(f_{j+1}{}^og_{0,j}))$
…

$(f_0{}^og_{0,0})(t)\ \mathbf{B}_{0,0}\ (f_1{}^og_{0,0})(t)$
$(f_0{}^o(g_{0,0}{}^og_{0,1}))(t)\ \mathbf{B}_{0,1}\ (f_1{}^o(g_{0,0}{}^og_{0,1}))(t)$
$(f_0{}^o(g_{0,0}{}^o(g_{0,1}{}^og_{0,2})))(t)\ \mathbf{B}_{0,2}\ (f_2{}^o(g_{0,0}{}^o(g_{0,1}{}^og_{0,2})))(t)$
…
$(f_0{}^o(g_{0,0}{}^o(g_{0,1}{}^o(g_{0,2}\ \ldots\ {}^og_{0,j})\ \ldots\ )))(t)\ \mathbf{B}_{0,j}\ (f_j{}^o(g_{0,0}{}^o(g_{0,1}{}^o(g_{0,2}\ \ldots\ {}^og_{0,j})\ \ldots\ )))(t)$

$g_{1,1}=G(f_1,f_2)$
$g_{1,2}=G((f_1{}^og_{1,1}),(f_2{}^og_{1,1}))=G((f_1{}^oG(f_1,f_2)),(f_2{}^oG(f_1,f_2)))$
$g_{1,3}=G((f_1{}^og_{1,2}),(f_3{}^og_{1,2}))=G((f_1{}^oG((f_1{}^oG(f_1,f_2)),(f_2{}^oG(f_1,f_2))))),(f_3{}^oG((f_1{}^oG(f_1,f_2)),(f_2{}^oG(f_1,f_2))))))$
…
$g_{1,j}=G((f_1{}^og_{1,j-1}),(f_j{}^og_{1,j-1}))$ $1<n$
$g_{1,j+1}=G((f_0{}^og_{1,j}),(f_{j+1}{}^og_{1,j}))$
…

$(f_1{}^o g_{1,1})(t)$ $\mathbf{B}_{1,1}$ $(f_1{}^o g_{1,1})(t)$
$(f_1{}^o(g_{1,1}{}^o g_{1,2}))(t)$ $\mathbf{B}_{1,2}$ $(f_2{}^o(g_{1,1}{}^o g_{1,2}))(t)$
$(f_1{}^o(g_{1,1}{}^o(g_{1,2}{}^o g_{1,3})))(t)$ $\mathbf{B}_{1,3}$ $(f_3{}^o(g_{1,1}{}^o(g_{1,2}{}^o g_{1,3})))(t)$
…
$(f_1{}^o(g_{1,1}{}^o(g_{1,2}{}^o(g_{1,3} \ldots {}^o g_{1,j}) \ldots)))(t)$ $\mathbf{B}_{1,j}$ $(f_j{}^o(g_{1,1}{}^o(g_{1,2}{}^o(g_{1,3} \ldots {}^o g_{1,j}) \ldots)))(t)$

$g_{i,i}=G(f_i,f_{i+1})$
$g_{i,i+1}=G((f_i{}^o g_{i,i}),(f_{i+1}{}^o g_{i,i}))=G((f_i{}^o G(f_i,f_{i+1})),(f_{i+1}{}^o G(f_i,f_{i+1})))$
$g_{i,i+2}=G((f_i{}^o g_{i,i+1}),(f_{i+2}{}^o g_{i,i+1}))=G((f_i{}^o\ G((f_i{}^o G(f_i,f_{i+1})),(f_{i+1}{}^o G(f_i,f_{i+1})))),(f_{i+2}{}^o\ G((f_i{}^o$ $G(f_i,f_{i+1})),(f_{i+1}{}^o G(f_i,f_{i+1})))))$
…
$g_{i,j}=G((f_1{}^o g_{i,j-1}),(f_j{}^o g_{i,j-1}))$ $m<n$
$g_{i,j+1}=G((f_0{}^o g_{i,j}),(f_{j+1}{}^o g_{i,j}))$
…

$(f_i{}^o g_{i,i})(t)$ $\mathbf{B}_{i,i}$ $(f_i{}^o g_{i,i})(t)$
$(f_i{}^o(g_{i,i}{}^o g_{i,i+1}))(t)$ $\mathbf{B}_{i,i+1}$ $(f_{i+1}{}^o(g_{i,i}{}^o g_{i,i+1}))\ (t)$
$(f_i{}^o(g_{i,i}{}^o(g_{i,i+1}{}^o(g_{i,i+2})))(t)$ $\mathbf{B}_{i,i+2}$ $(f_{i+2}{}^o(g_{i,i}{}^o(g_{i,i+1}{}^o g_{i,i+2})))(t)$
…
$(f_i{}^o(g_{i,i}{}^o(g_{i,i+1}{}^o(g_{i,i+2} \ldots {}^o g_{i,j}) \ldots)))(t)$ $\mathbf{B}_{i,j}$ $((f_{j+1}{}^o(g_{i,i}{}^o(g_{i,i+1}{}^o(g_{i,i+2} \ldots {}^o g_{i,j}) \ldots)))(t)$

Put all together from $g_{0,1}$ to $g_{i,j}$ in diagonal-symmetric ordering succession:

$g(i,j)=(g_{0,1}{}^o(g_{0,2}{}^o(g_{1,2}{}^o(g_{0,3}{}^o(g_{1,3}{}^o(g_{2,3}{}^o \ldots$
$(g_{0,j-1}{}^o(g_{1,j-1}{}^o(g_{2,j-1} \ldots {}^o(g_{j-2,j-1}{}^o(g_{0,j}{}^o(g_{1,j}{}^o(g_{2,j} \ldots {}^o g_{i,j}) \ldots))) \ldots)))) \ldots))))))$

Observation 10: One can include recursion in calcules but this necessitates special prerequisites, it cannot just be added to a given calcule. Notice that simple synaptic recursion, however, is an essential part of metalanguage.

i\j	0	1	2	3	4	5	6	7	8	
0		$g_{(0)}$	$g_{(1)}$	$g_{(3)}$	$g_{(6)}$	$g_{(10)}$	$g_{(15)}$	$g_{(21)}$	$g_{(28)}$	…
1			$g_{(2)}$	$g_{(4)}$	$g_{(7)}$	$g_{(11)}$	$g_{(16)}$	$g_{(22)}$	$g_{(29)}$	…
2				$g_{(5)}$	$g_{(8)}$	$g_{(12)}$	$g_{(17)}$	$g_{(23)}$	$g_{(30)}$	…
3					$g_{(9)}$	$g_{(13)}$	$g_{(18)}$	$g_{(24)}$	$g_{(31)}$	…
4						$g_{(14)}$	$g_{(19)}$	$g_{(25)}$	$g_{(32)}$	…
5							$g_{(20)}$	$g_{(26)}$	$g_{(33)}$	…
6								$g_{(27)}$	$g_{(34)}$	…
7									$g_{(35)}$	…
8										…
	…	…	…	…	…	…	…	…	…	…

$g(n)$

0	1	2	3	4	5	6	7	8	
	$g_{(0,1)}$	$g_{(0,2)}$	$g_{(0,3)}$	$g_{(0,4)}$	$g_{(0,5)}$	$g_{(0,6)}$	$g_{(0,7)}$	$g_{(0,8)}$	…
		$g_{(1,2)}$	$g_{(1,3)}$	$g_{(1,4)}$	$g_{(1,5)}$	$g_{(1,6)}$	$g_{(1,7)}$	$g_{(1,8)}$	…
			$g_{(2,3)}$	$g_{(2,4)}$	$g_{(2,5)}$	$g_{(2,6)}$	$g_{(2,7)}$	$g_{(2,8)}$	…
				$g_{(3,4)}$	$g_{(3,5)}$	$g_{(3,6)}$	$g_{(3,7)}$	$g_{(3,8)}$	…
					$g_{(4,5)}$	$g_{(4,6)}$	$g_{(4,7)}$	$g_{(4,8)}$	…
						$g_{(5,6)}$	$g_{(5,7)}$	$g_{(5,8)}$	…
							$g_{(6,7)}$	$g_{(6,8)}$	…
								$g_{(7,8)}$	…
									…
…	…	…	…	…	…	…	…	…	…

$g(i,j)$

$n=((j(j+1))/2+i)=t(i,j)$
$j=dscol(n)=aux(n)$
$i=dsrow(n)=n-((aux(n)(aux(n)+1))/2)$
$g(n)=g(dsrow(n),dscol(n))$

$g(n,t)=g(n)(t)$
$g(t)=g(t)(t)$

$(f_i{}^o g(i,j)(t)$ $\mathbf{B}_{i,j}$ $((f_{j+1}{}^o g(i,j)(t)$

9. ***Skolem's** Satz 1* **?**

There is still the problem that $\boldsymbol{B}_{i,j}$ cannot be expressed in object-language so that one can apply $f_i(g(t)) \sim f_j(g(t))$ or $f_i(g(t)) \prec f_j(g(t))$ or $f_j(g(t)) \prec f_i(g(t))$ for $t(i,j)<t$

$t(i,j)<t$ implying $f_i(g(t))= f_j(g(t))$ or $f_i(g(t))< f_j(g(t))$ or $f_j(g(t))< f_i(g(t))$

Using FUME-method one can write it down properly as ***sentence*** in <u>ALPHApython</u> as variant of ***sentence AϖS-Skolem1*** :

$f_i(t)$ $g(t)$ B i j $t(i,j)$ t

∀Aϖ1(Aϖ;Aϖ)[∃Aϖ1(Aϖ)[∃Aϖ2(Aϖ;Aϖ))[∀Aϖ1[∀Aϖ2[∀Aϖ3[
[(Aϖ3+Aϖ3)=((Aϖ1+Aϖ1)+((Aϖ1+Aϖ2)×(Aϖ1+Aϖ2′)))]→ [∀Aϖ4[[Aϖ3<Aϖ4]→[CB(
Aϖ1(Aϖ1;Aϖ1(Aϖ4));Aϖ1(Aϖ2;Aϖ1(Aϖ4));AϖB(Aϖ1(Aϖ1(Aϖ1;Aϖ);Aϖ1(Aϖ2;Aϖ);Aϖ4))]]]]]]]]

$f_i(g(t))$ $f_j(g(t))$ *coding* $\boldsymbol{B}_{i,j}$

where the ***relation-constant*** CB(Aϖ;Aϖ;Aϖ) is just a handy abbreviation.

Observation 11: The adjection AϖB(Aϖ(Aϖ);Aϖ(Aϖ);Aϖ) with values 0 for equality, 1 for minority and 2 majority carries the blemish of unsure **antisymmetry**. Furthermore the construction of Aϖ1(Aϖ) (Skolem's unary function *g)* involves metalingual **recursion** within **second-order** logic and thus is not free of doubts either. Therefor *Satz 1* properly expressed by ***AϖS-Skolem1*** cannot be considered to be proven.

10. *Existence of non-standard models of arithmetic ?*

Observation 12: Suppose that Skolem's *Satz 1* can be proven, then *Satz 2* actually becomes a ***THEOREM*** . It can be translated as follows: The arithmetic functions that appear in the set *M* of true statements (which include multinomials) form set ***M*** . The functions that are constructed by application of *Satz 1* form set ***N**** , they are ordered by an equivalence and a minority relation. They contain the set ***N*** of constant functions as a so-called initial part that corresponds one-to-one to the natural numbers. If one assumes that set *M* only contains omnications 'for all … ') of junctive formulae (i.e. only disjunction, conjunction and negation of equalities, no quantors) the true statements of set *M* become true statements for the functions of ***N**** .

It will be a different story with *Satz 3* and *Satz 4* that extend *Satz 2* to the complete set of *M* allowing for all kind of ***sentence*** strings. Besides Kleene's normal form it makes ample use of dot-dot-dot . The author has not looked into this any further and has not investigated if there is a way to justify Skolem's arguing in these ***sentence*** strings. It only pays if one knows that Skolem's *Satz 1* is valid. Therefor *Satz 4* cannot yet be considered to be proven.

Non-standard models of arithmetic are in limbo as far as Skolem's work is concerned.

However, there is a positive aspect of Skolem's paper. For certain examples of series $f_0(t)$, $f_1(t)$, $f_2(t)$ … one can actually construct the strictly ascending unary function *g* and the binary function *t* in such a way that *Satz 1* is a ***THEOREM*** . The shortcoming is that one can add further true statements to the set of *M* with more functions such that they are not true for the examples. The author will put forward that in both cases the functions *g* and *t* can be constructed effectively.

- Concrete calcule <u>ALPHATHETA</u> of Tho-arithmetics that fulfills ***Basiom*** strings ***B1*** to ***B17*** of the Appendix but not ***Basiom B18*** . In this case *g* is the identity function and *t* gives the start for function-equality ~ and function-minority ≺
- Concrete calcule <u>ALPHASIGMA</u> of Sko-arithmetics that fulfills ***Basiom*** strings ***B1*** to ***B18*** of the Appendix but not ***Basiom B19*** . In this case *g* and *t* are based on a prime-number technique.

B1	$\forall A_1[(A_1+0)=A_1]$	additivity right ***nullum***
B2	$\forall A_1[\forall A_2[(A_1+A_2)=(A_2+A_1)]]$	commutativity of addition
B3	$\forall A_1[\forall A_2[\forall A_3[((A_1+A_2)+A_3)=(A_1+(A_2+A_3))]]]$	associativity of addition
B4	$\forall A_1[\forall A_2[[[(A_1+A_2)\div A_2)=A_1]]]$	trunctractivity addition
B5	$\forall A_1[(A_1\times 0)=0]$	multiplicativity right ***nullum***
B6	$\forall A_1[\forall A_2[(A_1\times A_2)=(A_2\times A_1)]]$	commutativity of multiplication
B7	$\forall A_1[\forall A_2[\forall A_3[((A_1\times A_2)\times A_3)=(A_1\times(A_2\times A_3))]]]$	associativity of multiplication
B8	$\forall A_1[\forall A_2[\forall A_3[((A_1+A_2)\times A_3)=((A_1\times A_3)+(A_2\times A_3))]]]$	distributivity right addition, multiplication
B9	$\forall A_1[\forall A_2[\forall A_3[[\neg[[A_1=0]\lor[A_2=0]]]\land[(A_1\times A_2)=(A_1\times A_3)]]\to[A_2=A_3]]]]]]$	bi-injectivity multiplicat.
B10	$\forall A_1[\neg[A_1<A_1]]$	non-self-reflectivity of minority
B11	$\forall A_1[\forall A_2[[A_1<A_2]\to[\neg[A_2<A_1]]]]$	antisymmetry of minority
B12	$\forall A_1[\forall A_2[\forall A_3[[[A_1<A_2]\land[A_2<A_3]]\to[A_1<A_3]]]]$	transitivity of minority
B13	$\forall A_1[0<(A_1+1)]$	minority ***nullum***
B14	$\forall A_1[\forall A_2[\forall A_3[[A_1<A_2]\to[(A_1+A_3)<(A_2+A_3)]]]]$	monotony addition minority
B15	$\forall A_1[\forall A_2[\forall A_3[[A_1<A_2]\to[(A_1\times A_3')<(A_2\times A_3')]]]]$	monotony multiplication minority
B16	$\forall A_1[\forall A_2[[[A_1\leq A_2]\land[A_2<A_1')]]\to[A_1=A_2]]]$	discrety minority
B17	$\forall A_1[\forall A_2[[[A_1\leq A_2]\to[(A_1\div A_2)=0]]\land[[A_2<A_1]\to$ $[(A_2+(A_1\div A_2))=A_1]]]$	additivity trunctraction minority (implying bi-injectivity of addition)
B18	$\forall A_1[\forall A_2[\exists A_0[[[A_2=0]\land[A_0=0]]\lor[[A_2\neq 0]\land$ $[[(A_2\times A_0)\leq A_1]\land[A_1<((A_2\times A_0)+A_2)]]]]]$	divity[1)]
B19	$\forall A_1[\exists A_0[[(A_0\times A_0)\leq A_1]\land[A_1<(A'\times A_0')]]]$	biradicality[2)]

extra-relation-constant :: $\nmid A$ ¦ $A\nparallel A$ ¦ $A\mid A$ ¦ $A\nmid A$ — to be expanded

$[\nmid A_1]\leftrightarrow[\forall A_{12}[\forall A_{13}[[[1<A_{12}]\land[1<A_{12}]]\land[A_1\neq(A_{12}\times A_{13})]]]]$ — primity, $\forall$ limited by A_1 (prime number)

$[\nparallel A_1]\leftrightarrow[\forall A_{12}[\forall A_{13}[A_1\neq(A_{12}''\times A_{13}'')]]]$ — primality (0 or 1 or prime)

$[A_1\mid A_2]\leftrightarrow[\exists A_{11}[[0\neq A_{11}]\land[A_1=(A_{11}\times A_2)]]]$ — divisibility, $\exists$ limited by A_1

$[A_1\nmid A_2]\leftrightarrow[[\nmid A_2]\land[\forall A_3[[A_1\mid A_3]\to[A_3\mid A_2]]]]$ — prime-potency (power of prime)

B20 — prima-predecessity[3)]

$$\forall A_1[\exists A_0[[[[\nparallel A_1]\land[A_0=A_1]]\lor[[\neg[\nparallel A_1]]\land[\forall A_4[[[[A_4<A_1]\land[\nparallel A_4]]\land[\forall A_5[[[A_4<A_5]\land[A_5<A_1]]\to[\neg[\nparallel A_5]]]]]\to[A_0=A_4]]]]]\land[\forall A_6[[[[\nparallel A_1]\land[A_6=A_1]]\lor[[\neg[\nparallel A_1]]\land[\forall A_4[[[[A_4<A_1]\land[\nparallel A_4]]\land[\forall A_5[[[A_4<A_5]\land[A_5<A_1]]\to[\neg[\nparallel A_5]]]]]\to[A_6=A_4]]]]]\to[A_6=A_0]]]]]$$

B21 — prima-successity[4)]

$$\forall A_1[\exists A_0[[[[\nparallel A_1]\land[A_0=A_1]]\lor[[\neg[\nparallel A_1]]\land[\exists A_4[[[[A_1<A_4]\land[\nparallel A_4]]\land[A_0=A_4]]\land[\forall A_5[[[A_5<A_4]\land[A_1<A_5]]\to[\neg[\nparallel A_5]]]]]]]\land[\forall A_6[[[[\nparallel A_1]\land[A_6=A_1]]\lor[[\neg[\nparallel A_1]]\land[\exists A_4[[[[A_1<A_4]\land[\nparallel A_4]]\land[A_6=A_4]]\land[\forall A_5[[[A_5<A_4]\land[A_1<A_5]]\to[\neg[\nparallel A_5]]]]]]]\to[A_6=A_0]]]]]$$

B22 — prime-limity[5)]

$$\forall A_1[\exists A_0[[[A_1\leq 1]\to[A_0=1]]\land[[1<A_1]\to[[[[\forall A_4[[[A_4\leq A_1]\land[\nmid A_4]]\to[[A_0\mid A_4]\land[\neg[A_0\mid(A_4\times A_4)]]]]]]\land[\forall A_4[[[A_0\mid A_4]\land[\nmid A_4]]\to[A_4\leq A_1]]]]\land[A_1<A_0']]\land[\nmid A_0']]]]]$$

B23 — prime-maximality[6)]

$$\forall A_1[\forall A_2[\exists A_0[[[[[A_1\mid A_2]\land[\neg[\nmid A_2]]]\lor[\neg[A_1\mid A_2]]]\land[A_0=1]]\lor[[[[A_1\mid A_0]\land[A_0\nmid A_2]]\land[\neg[A_1\mid(A_0\times A_2)]]]]\land[\forall A_3[[[[A_1\mid A_3]\land[A_3\nmid A_2]]\land[\neg[A_1\mid(A_3\times A_2)]]]\to[A_3=A_0]]]]]]]$$

1) divity means that one can decompose every positive natural number into a multiple of a nonvanishing divisor and a division remainder that is less than the divisor; via bi-injectivity of addition and multiplication the decomposition is unique

2) biradicality means that there is an entire square-root such that the square of its successor is larger than the radicand

3) prima-predecessity means that every positive number is either a prima number or there exists uniquely a predecessive prima number

4) prima-successity means that every number is either a prima number or there exists uniquely a successive prima number

5) prime-limity means that for every number greater *1* there exists the product of all smaller or equal prime numbers, for numbers *0* and *1* take *1* as product

6) prime-maximality means that for prime divisors of a number exists a maximum power that divides the number; it is an antecessor of the so-called 'fundamental theorem of arithmetic', but it lacks the multiple product.