Some canonical calculi and logical languages
The concept of hypercalculus

András Máté

27.09.2024

A language of first-order logic (informal description)

A language of first-order logic (informal description)

Primitive symbols: variables, predicates of any arity, name functors of any arity

A language of first-order logic (informal description)

Primitive symbols: variables, predicates of any arity, name functors of any arity

This language will be the *maximal* first-order language, with an infinite sequence of predicates and name functors for any arity.

A language of first-order logic (informal description)

Primitive symbols: variables, predicates of any arity, name functors of any arity

This language will be the *maximal* first-order language, with an infinite sequence of predicates and name functors for any arity.

Initial letters for three sorts of primitive symbols: $\mathfrak{x}$ for variables, π for predicates and φ for name functors.

A language of first-order logic (informal description)

Primitive symbols: variables, predicates of any arity, name functors of any arity

This language will be the *maximal* first-order language, with an infinite sequence of predicates and name functors for any arity.

Initial letters for three sorts of primitive symbols: $\mathfrak{x}$ for variables, π for predicates and φ for name functors.
Primitive predicates of the same arity resp. primitive name functors of the same arity resp. variables will be distinguished from each other by indexes.

A language of first-order logic (informal description)

Primitive symbols: variables, predicates of any arity, name functors of any arity

This language will be the *maximal* first-order language, with an infinite sequence of predicates and name functors for any arity.

Initial letters for three sorts of primitive symbols: $\mathfrak{x}$ for variables, π for predicates and φ for name functors.
Primitive predicates of the same arity resp. primitive name functors of the same arity resp. variables will be distinguished from each other by indexes.

Strings of o-s (omicrons) will mark the arity of a predicate resp. name functor, and they will also count the empty places of the predicate resp. the name functor.

A language of first-order logic (informal description)

Primitive symbols: variables, predicates of any arity, name functors of any arity

This language will be the *maximal* first-order language, with an infinite sequence of predicates and name functors for any arity.

Initial letters for three sorts of primitive symbols: $\mathfrak{x}$ for variables, π for predicates and φ for name functors.
Primitive predicates of the same arity resp. primitive name functors of the same arity resp. variables will be distinguished from each other by indexes.

Strings of o-s (omicrons) will mark the arity of a predicate resp. name functor, and they will also count the empty places of the predicate resp. the name functor.
The (primitive) logical constants of first-order logic are the usual ones. The alphabet of our first-order language:

$$\mathcal{A}_{Language(FOL)} = \{(,\ ),\ \iota,\ o,\ \mathfrak{x},\ \varphi,\ \pi,\ =,\ \neg,\ \supset,\ \forall\}$$

A language of first-order logic (informally, continued)

A language of first-order logic (informally, continued)

A language of first-order logic (informally, continued)

We apply name functors and predicates always for one argument (individual term) only, i.e. we fill in the argument places one by one(currying).

A language of first-order logic (informally, continued)

We apply name functors and predicates always for one argument (individual term) only, i.e. we fill in the argument places one by one(currying).

Auxiliary letters (with intended meanings in brackets): I (index), A (arity), V (variable), N (name functor), P (predicate), T (term), F (formula). We use calculus variables as needed (not to be changed with object-language variables).

The calculus $K_{Language(FOL)}$

1. I The empty word is an index.

1. I The empty word is an index.
2. $Ix \to Ix\iota$

The calculus $K_{Language(FOL)}$

1. I — The empty word is an index.
2. $Ix \to Ix\iota$
3. A — The empty word is an arity.

The calculus $K_{Language(FOL)}$

1. I — The empty word is an index.
2. $Ix \to Ix\iota$
3. A — The empty word is an arity.
4. $Ax \to Axo$

The calculus $K_{Language(FOL)}$

1. I	The empty word is an index.
2. $Ix \to Ix\iota$	
3. A	The empty word is an arity.
4. $Ax \to Axo$	
5. $Ix \to V\mathfrak{x}x$	

The calculus $K_{Language(FOL)}$

1. I — The empty word is an index.
2. $Ix \to Ix\iota$
3. A — The empty word is an arity.
4. $Ax \to Axo$
5. $Ix \to V\mathfrak{x}x$
6. $Ax \to Iy \to xN\varphi xy$ — n-ary name functors

The calculus $K_{Language(FOL)}$

1. I — The empty word is an index.
2. $Ix \to Ix\iota$
3. A — The empty word is an arity.
4. $Ax \to Axo$
5. $Ix \to V\mathfrak{x}x$
6. $Ax \to Iy \to xN\varphi xy$ — n-ary name functors
7. $Ax \to Iy \to xP\pi xy$ — n-ary predicates

The calculus $K_{Language(FOL)}$

1. I — The empty word is an index.
2. $Ix \to Ix\iota$
3. A — The empty word is an arity.
4. $Ax \to Axo$
5. $Ix \to V\mathfrak{x}x$
6. $Ax \to Iy \to xN\varphi xy$ — n-ary name functors
7. $Ax \to Iy \to xP\pi xy$ — n-ary predicates
8. $Vx \to Tx$ — The variables are terms.

The calculus $K_{Language(FOL)}$

1. I — The empty word is an index.
2. $Ix \to Ix\iota$
3. A — The empty word is an arity.
4. $Ax \to Axo$
5. $Ix \to V\mathfrak{x}x$
6. $Ax \to Iy \to xN\varphi xy$ — n-ary name functors
7. $Ax \to Iy \to xP\pi xy$ — n-ary predicates
8. $Vx \to Tx$ — The variables are terms.
9. $Nx \to Tx$ — Zero-argument name functors are terms.

The calculus $K_{Language(FOL)}$

1.	I	The empty word is an index.
2.	$Ix \to Ix\iota$	
3.	A	The empty word is an arity.
4.	$Ax \to Axo$	
5.	$Ix \to V\mathfrak{x}x$	
6.	$Ax \to Iy \to xN\varphi xy$	n-ary name functors
7.	$Ax \to Iy \to xP\pi xy$	n-ary predicates
8.	$Vx \to Tx$	The variables are terms.
9.	$Nx \to Tx$	Zero-argument name functors are terms.
10.	$Ax \to xoNy \to Tz \to xNyz$	Application of name functors with at least one argument

The calculus $K_{Language(FOL)}$ (continuation)

11. $Ax \to xoPy \to Tz \to xPyz$ Application of predicates

The calculus $K_{Language(FOL)}$ (continuation)

11.	$Ax \to xoPy \to Tz \to xPyz$	Application of predicates
12.	$Px \to Fx$	Zero-arity predicates are formulas.

The calculus $K_{Language(FOL)}$ (continuation)

11.	$Ax \to xoPy \to Tz \to xPyz$	Application of predicates
12.	$Px \to Fx$	Zero-arity predicates are formulas.
13.	$Tx \to Ty \to F(x = y)$	

The calculus $K_{Language(FOL)}$ (continuation)

11.	$Ax \to xoPy \to Tz \to xPyz$	Application of predicates
12.	$Px \to Fx$	Zero-arity predicates are formulas.
13.	$Tx \to Ty \to F(x = y)$	
14.	$Fx \to F\neg x$	

The calculus $K_{Language(FOL)}$ (continuation)

11.	$Ax \to xoPy \to Tz \to xPyz$	Application of predicates
12.	$Px \to Fx$	Zero-arity predicates are formulas.
13.	$Tx \to Ty \to F(x = y)$	
14.	$Fx \to F\neg x$	
15.	$Fx \to Fy \to F(x \supset y)$	

The calculus $K_{Language(FOL)}$ (continuation)

11.	$Ax \to xoPy \to Tz \to xPyz$	Application of predicates
12.	$Px \to Fx$	Zero-arity predicates are formulas.
13.	$Tx \to Ty \to F(x = y)$	
14.	$Fx \to F\neg x$	
15.	$Fx \to Fy \to F(x \supset y)$	
16.	$Vx \to Fy \to F\forall xy$	

11.	$Ax \to xoPy \to Tz \to xPyz$	Application of predicates
12.	$Px \to Fx$	Zero-arity predicates are formulas.
13.	$Tx \to Ty \to F(x = y)$	
14.	$Fx \to F\neg x$	
15.	$Fx \to Fy \to F(x \supset y)$	
16.	$Vx \to Fy \to F\forall xy$	
16^*.	$Fx \to x$	Release rule

Closing remark and a homework

Closing remark and a homework

The $\mathcal{A}_{Language(FOL)}$-strings derivable in this calculus are just the wff's of our $Language(FOL)$. By changing the release rule and/or leaving off some rules we could define other syntactical categories (terms, atomic formulas, etc.) of the language.

Closing remark and a homework

The $\mathcal{A}_{Language(FOL)}$-strings derivable in this calculus are just the wff's of our $Language(FOL)$. By changing the release rule and/or leaving off some rules we could define other syntactical categories (terms, atomic formulas, etc.) of the language.

Homework: How to change $K_{Language(FOL)}$ to define the terms resp. atomic formulas of our language?

Hypercalculi and their use

Hypercalculi are canonical calculi that we use to define classes of canonical calculi (in some encoded form) and other general concepts connected with canonical calculi.

Hypercalculi and their use

Hypercalculi are canonical calculi that we use to define classes of canonical calculi (in some encoded form) and other general concepts connected with canonical calculi.

$\mathcal{C}$-*rule* over an alphabet $\mathcal{C}$;
derivability in a canonical calculus:
both were defined inductively.

Hypercalculi are canonical calculi that we use to define classes of canonical calculi (in some encoded form) and other general concepts connected with canonical calculi.

$\mathcal{C}$-*rule* over an alphabet $\mathcal{C}$;
derivability in a canonical calculus:
both were defined inductively.

Canonical calculi are finite sequences of rules (special strings). To represent them as strings we need a *sequencing character* distinct from the letters.

Hypercalculi and their use

Hypercalculi are canonical calculi that we use to define classes of canonical calculi (in some encoded form) and other general concepts connected with canonical calculi.

$\mathcal{C}$-*rule* over an alphabet $\mathcal{C}$;
derivability in a canonical calculus:
both were defined inductively.

Canonical calculi are finite sequences of rules (special strings). To represent them as strings we need a *sequencing character* distinct from the letters.

Hypercalculi are canonical calculi that we use to define classes of canonical calculi (in some encoded form) and other general concepts connected with canonical calculi.

$\mathcal{C}$-*rule* over an alphabet $\mathcal{C}$;
derivability in a canonical calculus:
both were defined inductively.

Canonical calculi are finite sequences of rules (special strings). To represent them as strings we need a *sequencing character* distinct from the letters.
An informal remark: Hypercalculi are canonical calculi just as any other calculus. *We* read the strings they produce as rules, derivability relations or calculi. The calculus deriving the code of any canonical calculus also derives the code of itself.

How to represent an arbitrary calculus **C**?

How to represent an arbitrary calculus $\mathbf{C}$?

We want to construct a calculus $\mathbf{H}_1$ that derives strings representing any canonical calculus.

How to represent an arbitrary calculus **C**?

We want to construct a calculus $\mathbf{H}_1$ that derives strings representing any canonical calculus.

Be **C** an arbitrary canonical calculus. First, translate it by letter into a string of our new calculus.

How to represent an arbitrary calculus **C**?

We want to construct a calculus $\mathbf{H}_1$ that derives strings representing any canonical calculus.

Be **C** an arbitrary canonical calculus. First, translate it by letter into a string of our new calculus.

Letters of the alphabet of **C** will be represented as $\{\alpha,\ \beta\}$-strings beginning with α and followed by β-s.

How to represent an arbitrary calculus **C**?

We want to construct a calculus $\mathbf{H}_1$ that derives strings representing any canonical calculus.

Be **C** an arbitrary canonical calculus. First, translate it by letter into a string of our new calculus.

Letters of the alphabet of **C** will be represented as $\{\alpha, \beta\}$-strings beginning with α and followed by β-s.

The **C**-variables will be translated similarly, but the beginning character will be ξ instead of α.

How to represent an arbitrary calculus **C**?

We want to construct a calculus $\mathbf{H}_1$ that derives strings representing any canonical calculus.

Be **C** an arbitrary canonical calculus. First, translate it by letter into a string of our new calculus.

Letters of the alphabet of **C** will be represented as $\{\alpha,\ \beta\}$-strings beginning with α and followed by β-s.

The **C**-variables will be translated similarly, but the beginning character will be ξ instead of α.

Translation of the arrow: $\gg$. Sequencing character: $*$.

How to represent an arbitrary calculus **C**?

We want to construct a calculus $\mathbf{H}_1$ that derives strings representing any canonical calculus.

Be **C** an arbitrary canonical calculus. First, translate it by letter into a string of our new calculus.

Letters of the alphabet of **C** will be represented as $\{\alpha,\ \beta\}$-strings beginning with α and followed by β-s.

The **C**-variables will be translated similarly, but the beginning character will be ξ instead of α.

Translation of the arrow: $\gg$. Sequencing character: $*$.

So the the strings that represent calculi will consist of the characters of the following alphabet:

$$\mathcal{A}_{cc} = \{\alpha,\ \beta,\ \xi,\ \gg,\ *\}$$

The alphabet of $\mathbf{H}_1$

The alphabet of $\mathbf{H}_1$

The alphabet will contain $\mathcal{A}_{cc}$ as a subset. Above that, we'll need the following auxiliary characters (intended meaning in brackets):

The alphabet of $\mathbf{H}_1$

The alphabet will contain $\mathcal{A}_{cc}$ as a subset. Above that, we'll need the following auxiliary characters (intended meaning in brackets):

- I (index)
- L (Translation of a letter of **C**)
- V (Translation of a **C**-variable)
- W (Translation of a word, i.e. variable-free string)
- T (Translation of a term, i.e. string of letters and variables)
- R (Translation of a **C**-rule)
- K (Translation of an arbitrary calculus **C**)

The calculus $\mathbf{H}_1$ (beginning)

1. I
2. $Ix \to Ix\beta$
3. $Ix \to L\alpha x$
4. $Ix \to V\xi x$
5. W
6. $Wx \to Ly \to Wxy$
7. T
8. $Tx \to Ly \to Txy$
9. $Tx \to Vy \to Txy$

The calculus $\mathbf{H}_1$ (continuation)

The calculus $\mathbf{H}_1$ (continuation)

10. $Tx \to Rx$
11. $Tx \to Ry \to Rx \gg y$
12. $Rx \to Kx$
13. $Kx \to Ry \to Kx * y$

13^* $Kx \to x$

The calculus $\mathbf{H}_1$ (continuation)

$$\begin{array}{ll} 10. & Tx \to Rx \\ 11. & Tx \to Ry \to Rx \gg y \\ 12. & Rx \to Kx \\ 13. & Kx \to Ry \to Kx * y \\ 13^* & Kx \to x \end{array}$$

This calculus derives the translation of any calculus over any alphabet (including its own translation $\mathbf{h}_1$).

Next goal: formalize derivability

Next goal: formalize derivability

Construct a (hyper)calculus $\mathbf{H}_2$ such that if the calculus $\mathbf{C}$ derives the string c then $\mathbf{H}_2$ derives a string that is the translation of $\mathbf{C} \mapsto c$.

Next goal: formalize derivability

Construct a (hyper)calculus $\mathbf{H}_2$ such that if the calculus $\mathbf{C}$ derives the string c then $\mathbf{H}_2$ derives a string that is the translation of $\mathbf{C} \mapsto c$.

We extend $\mathbf{H}_1$ (dropping the release rule 13*) to the calculus $\mathbf{H}_2$.

Next goal: formalize derivability

Construct a (hyper)calculus $\mathbf{H}_2$ such that if the calculus $\mathbf{C}$ derives the string c then $\mathbf{H}_2$ derives a string that is the translation of $\mathbf{C} \mapsto c$.

We extend $\mathbf{H}_1$ (dropping the release rule 13*) to the calculus $\mathbf{H}_2$.

Two new auxiliary letters: D for derivable and S for substitution. In more details:

Next goal: formalize derivability

Construct a (hyper)calculus $\mathbf{H}_2$ such that if the calculus $\mathbf{C}$ derives the string c then $\mathbf{H}_2$ derives a string that is the translation of $\mathbf{C} \mapsto c$.

We extend $\mathbf{H}_1$ (dropping the release rule 13*) to the calculus $\mathbf{H}_2$.

Two new auxiliary letters: D for derivable and S for substitution. In more details:

- xDy: the calculus x derives the string y

Next goal: formalize derivability

Construct a (hyper)calculus $\mathbf{H}_2$ such that if the calculus $\mathbf{C}$ derives the string c then $\mathbf{H}_2$ derives a string that is the translation of $\mathbf{C} \mapsto c$.

We extend $\mathbf{H}_1$ (dropping the release rule 13^*) to the calculus $\mathbf{H}_2$.

Two new auxiliary letters: D for derivable and S for substitution. In more details:

- xDy: the calculus x derives the string y
- $vSuSySx$: if we substitute the word y for the variable x, we get the string v from the string u. Remember that words are variable-free strings.

Next goal: formalize derivability

Construct a (hyper)calculus $\mathbf{H}_2$ such that if the calculus $\mathbf{C}$ derives the string c then $\mathbf{H}_2$ derives a string that is the translation of $\mathbf{C} \mapsto c$.

We extend $\mathbf{H}_1$ (dropping the release rule 13^*) to the calculus $\mathbf{H}_2$.

Two new auxiliary letters: D for derivable and S for substitution. In more details:

- xDy: the calculus x derives the string y
- $vSuSySx$: if we substitute the word y for the variable x, we get the string v from the string u. Remember that words are variable-free strings.

In the above description of the intended meaning, I have dropped the phrase 'translation of'. But never forget that we speak here not about the letters, variables, etc. of our hypercalculus, but about the strings translating the letters etc. of the original calculus.

Substitution in $\mathbf{H}_2$

Substitution needs an inductive definition, too:

Substitution in $\mathbf{H}_2$

Substitution needs an inductive definition, too:

14. $Lu \to uSuSySx$
15. $\gg S \gg SySx$
16. $Vx \to Iz \to x\beta zSx\beta zSySx$
17. $Vx \to Iz \to xSxSySx\beta z$
18. $Vx \to Wy \to ySxSySx$
19. $vSuSySx \to wSzSySx \to vwSuzSySx$

Substitution in $\mathbf{H}_2$

Substitution needs an inductive definition, too:

14. $Lu \rightarrow uSuSySx$
15. $\gg S \gg SySx$
16. $Vx \rightarrow Iz \rightarrow x\beta zSx\beta zSySx$
17. $Vx \rightarrow Iz \rightarrow xSxSySx\beta z$
18. $Vx \rightarrow Wy \rightarrow ySxSySx$
19. $vSuSySx \rightarrow wSzSySx \rightarrow vwSuzSySx$

Base: The substitution of the variable x by the word y makes y from x (rule 18.) and leaves any other character – letters (14.), the arrow (15.), other variables (16.-17) – unchanged.

Substitution in $\mathbf{H}_2$

Substitution needs an inductive definition, too:

14. $Lu \to uSuSySx$
15. $\gg S \gg SySx$
16. $Vx \to Iz \to x\beta zSx\beta zSySx$
17. $Vx \to Iz \to xSxSySx\beta z$
18. $Vx \to Wy \to ySxSySx$
19. $vSuSySx \to wSzSySx \to vwSuzSySx$

Base: The substitution of the variable x by the word y makes y from x (rule 18.) and leaves any other character – letters (14.), the arrow (15.), other variables (16.-17) – unchanged.
Inductive rule (19.): If the substitution makes v from u and w from z, then from their concatenation uz it makes the concatenation of the results vw.

Derivability in $\mathbf{H}_2$

Derivability in $\mathbf{H}_2$

Base: every calculus derives its rules. (In details: an one-rule calculus derives the rule, and longer calculi derive their last, first and middle rules.) Inductive rules are substitution and detachment.

Derivability in $\mathbf{H}_2$

Base: every calculus derives its rules. (In details: an one-rule calculus derives the rule, and longer calculi derive their last, first and middle rules.) Inductive rules are substitution and detachment.

20. $Rx \to xDx$
21. $Rx \to Ky \to y * xDx$
22. $Rx \to Ky \to x * yDx$
23. $Rx \to Ky \to Kz \to y * x * zDx$
24. $zDu \to vSuSySx \to zDv$
25. $xDy \to xDy \gg z \to xDz$

Derivability in $\mathbf{H}_2$

Base: every calculus derives its rules. (In details: an one-rule calculus derives the rule, and longer calculi derive their last, first and middle rules.) Inductive rules are substitution and detachment.

20. $Rx \to xDx$
21. $Rx \to Ky \to y * xDx$
22. $Rx \to Ky \to x * yDx$
23. $Rx \to Ky \to Kz \to y * x * zDx$
24. $zDu \to vSuSySx \to zDv$
25. $xDy \to xDy \gg z \to xDz$

The calculus $\mathbf{H}_2$ consisting of the rules 1-25 derives Ka, Wb and aDb iff a is the translation of some calculus $\mathbf{C}$, b is the translation of a word c of the alphabet of $\mathbf{C}$ and $\mathbf{C}$ derives c. We can't give suitable release rules here.

The calculus $\mathbf{H}_3$

$\mathbf{H}_2$ (over an alphabet $\mathcal{A}_{cc}$ plus 9 auxiliary letters) derives strings with the intended meanings „a is a calculus", „b is a string of the alphabet of a", „a derives b". (a and b are *translations, codes* of a calculus resp. word in $\mathcal{A}_{cc}$.)

The calculus $\mathbf{H}_3$

$\mathbf{H}_2$ (over an alphabet $\mathcal{A}_{cc}$ plus 9 auxiliary letters) derives strings with the intended meanings „a is a calculus", „b is a string of the alphabet of a", „a derives b". (a and b are *translations, codes* of a calculus resp. word in $\mathcal{A}_{cc}$.)

The calculus $\mathbf{H}_3$ is an extension of $\mathbf{H}_2$. It renders numerals to every $\mathcal{A}_{cc}$-string. (This is in effect a Gödel numbering.)
Numerals: strings consisting of α-s only.

The calculus $\mathbf{H}_3$

$\mathbf{H}_2$ (over an alphabet $\mathcal{A}_{cc}$ plus 9 auxiliary letters) derives strings with the intended meanings „a is a calculus", „b is a string of the alphabet of a", „a derives b". (a and b are *translations, codes* of a calculus resp. word in $\mathcal{A}_{cc}$.)

The calculus $\mathbf{H}_3$ is an extension of $\mathbf{H}_2$. It renders numerals to every $\mathcal{A}_{cc}$-string. (This is in effect a Gödel numbering.)
Numerals: strings consisting of α-s only.

First step: introduce a lexicographic ordering of $\mathcal{A}_{cc}$-strings.
New auxiliary letter: F for the relation 'follows'.
I. e., xFy should mean that the string y follows x in the lexicographic ordering.
Base: α follows the empty word.
Inductive rules define the follower of a string according to its last letter.

Lexicographic ordering

26. $F\alpha$
27. $x\alpha Fx\beta$
28. $x\beta Fx\xi$
29. $x\xi Fx \gg$
30. $x \gg Fx*$
31. $xFy \to x * Fy\alpha$

Lexicographic ordering

26. $F\alpha$
27. $x\alpha F x\beta$
28. $x\beta F x\xi$
29. $x\xi F x \gg$
30. $x \gg F x*$
31. $xFy \rightarrow x * Fy\alpha$

From the language radix axioms it follows that:
Every $\mathcal{A}_{cc}$-string has one and only one follower;
Except of the empty string, each string is the follower of one and only one string.
The empty string is not a follower of anything.
I. e., strings with the empty string as 0 and this follower-relation as the successor-function fulfil axioms of primitive Peano arithmetics without mathematical induction.

Gödel numbering of $\mathcal{A}_{cc}$-strings

Now we can add the (Gödel-)numbering to our calculus on the trivial way.
G is a a new auxiliary letter, intended meaning of xGy: y is the ordinal number of x in the lexicographic ordering.

Gödel numbering of $\mathcal{A}_{cc}$-strings

Now we can add the (Gödel-)numbering to our calculus on the trivial way.
G is a a new auxiliary letter, intended meaning of xGy: y is the ordinal number of x in the lexicographic ordering.
Basis: the ordinal number of the empty string is the empty string itself.
Inductive rule: to get the number of the follower of a string x we need to add an α to the number of x.

Gödel numbering of $\mathcal{A}_{cc}$-strings

Now we can add the (Gödel-)numbering to our calculus on the trivial way.
G is a a new auxiliary letter, intended meaning of xGy: y is the ordinal number of x in the lexicographic ordering.
Basis: the ordinal number of the empty string is the empty string itself.
Inductive rule: to get the number of the follower of a string x we need to add an α to the number of x.

32. G
33. $xFy \to xGz \to yGz\alpha$

Gödel numbering of $\mathcal{A}_{cc}$-strings

Now we can add the (Gödel-)numbering to our calculus on the trivial way.
G is a a new auxiliary letter, intended meaning of xGy: y is the ordinal number of x in the lexicographic ordering.
Basis: the ordinal number of the empty string is the empty string itself.
Inductive rule: to get the number of the follower of a string x we need to add an α to the number of x.

32. G

33. $xFy \to xGz \to yGz\alpha$

Our hypercalculus $\mathbf{H}_3$ now consists of the rules 1-33. and it suffices to prove at least one important incompleteness result.

Autonomous numerals

Autonomous numerals

Be **C** an arbitrary calculus.

The translation of **C** into our language is some $\mathcal{A}_{cc}$-word a.

$\mathbf{H}_3$ derives Ka.

There is a numeral c s.t. $\mathbf{H}_3$ derives aGc, i. e. the Gödel number of **C** is c.

Autonomous numerals

Be **C** an arbitrary calculus.
The translation of **C** into our language is some $\mathcal{A}_{cc}$-word a.
$\mathbf{H}_3$ derives Ka.
There is a numeral c s.t. $\mathbf{H}_3$ derives aGc, i. e. the Gödel number of **C** is c.

Does **C** derive a string whose translation is c?
Be **C** a calculus with this property (deriving its own Gödel number).
Then $\mathbf{H}_3$ derives aDc, too.
Let us call such c-s autonomous numbers.

Autonomous numerals

Be **C** an arbitrary calculus.
The translation of **C** into our language is some $\mathcal{A}_{cc}$-word a.
$\mathbf{H}_3$ derives Ka.
There is a numeral c s.t. $\mathbf{H}_3$ derives aGc, i. e. the Gödel number of **C** is c.

Does **C** derive a string whose translation is c?
Be **C** a calculus with this property (deriving its own Gödel number).
Then $\mathbf{H}_3$ derives aDc, too.
Let us call such c-s autonomous numbers.
Let us extend $\mathbf{H}_3$ to define autonomous numbers.
New auxiliary letter: A with the intended meaning „autonomous”. New rule:

Be **C** an arbitrary calculus.
The translation of **C** into our language is some $\mathcal{A}_{cc}$-word a.
$\mathbf{H}_3$ derives Ka.
There is a numeral c s.t. $\mathbf{H}_3$ derives aGc, i. e. the Gödel number of **C** is c.

Does **C** derive a string whose translation is c?
Be **C** a calculus with this property (deriving its own Gödel number).
Then $\mathbf{H}_3$ derives aDc, too.
Let us call such c-s autonomous numbers.
Let us extend $\mathbf{H}_3$ to define autonomous numbers.
New auxiliary letter: A with the intended meaning „autonomous". New rule:

$$34. \quad xDy \to xGy \to Ay$$

Our Gödel-like theorem

Our Gödel-like theorem

The numbers are the strings of the one-letter alphabet $\mathcal{A}_0 = \{\alpha\}$, so their class is $\mathcal{A}_0^\circ$ and it can be defined inductively. The class of autonomous numerals, in class theoretic notation:

$$\boldsymbol{Aut} = \{x : x \in \mathcal{A}_0^\circ \wedge \mathbf{H}_3 \mapsto Ax\}$$

By adding a release rule deleting A to $\mathbf{H}_3$, we gain a definition of $\boldsymbol{Aut}$ by a canonical calculus.

Our Gödel-like theorem

The numbers are the strings of the one-letter alphabet $\mathcal{A}_0 = \{\alpha\}$, so their class is $\mathcal{A}_0^\circ$ and it can be defined inductively. The class of autonomous numerals, in class theoretic notation:

$$\boldsymbol{Aut} = \{x : x \in \mathcal{A}_0^\circ \wedge \mathbf{H}_3 \mapsto Ax\}$$

By adding a release rule deleting A to $\mathbf{H}_3$, we gain a definition of $\boldsymbol{Aut}$ by a canonical calculus.

We prove that the string class $\mathcal{A}_0^\circ - \boldsymbol{Aut}$ (the class of non-autonomous numerals) cannot be defined inductively.

The numbers are the strings of the one-letter alphabet $\mathcal{A}_0 = \{\alpha\}$, so their class is $\mathcal{A}_0^\circ$ and it can be defined inductively. The class of autonomous numerals, in class theoretic notation:

$$\boldsymbol{Aut} = \{x : x \in \mathcal{A}_0^\circ \wedge \mathbf{H}_3 \mapsto Ax\}$$

By adding a release rule deleting A to $\mathbf{H}_3$, we gain a definition of $\boldsymbol{Aut}$ by a canonical calculus.

We prove that the string class $\mathcal{A}_0^\circ - \boldsymbol{Aut}$ (the class of non-autonomous numerals) cannot be defined inductively.

Theorem: There is no canonical calculus $\mathbf{C}$ over some $B \supseteq \mathcal{A}_{cc}$ s.t. for any string x,

$$\mathbf{C} \mapsto x \Leftrightarrow x \in \mathcal{A}_0^\circ - \boldsymbol{Aut}$$

Proof of the theorem

Proof of the theorem

Let us assume toward contradiction that we have a calculus $\mathbf{C}$ with the Gödel number g s.t for every non-autonomous numeral c, $\mathbf{C} \mapsto c$, and there is no autonomous numeral d for that $\mathbf{C} \mapsto d$.

Proof of the theorem

Let us assume toward contradiction that we have a calculus $\mathbf{C}$ with the Gödel number g s.t for every non-autonomous numeral c, $\mathbf{C} \mapsto c$, and there is no autonomous numeral d for that $\mathbf{C} \mapsto d$.

Suppose that $\mathbf{C} \mapsto g$. In this case, $\mathbf{C}$ is an autonomous calculus, g is an autonomous number, therefore $\mathbf{C}$ does not derive g. Contradiction.

Proof of the theorem

Let us assume toward contradiction that we have a calculus $\mathbf{C}$ with the Gödel number g s.t for every non-autonomous numeral c, $\mathbf{C} \mapsto c$, and there is no autonomous numeral d for that $\mathbf{C} \mapsto d$.

Suppose that $\mathbf{C} \mapsto g$. In this case, $\mathbf{C}$ is an autonomous calculus, g is an autonomous number, therefore $\mathbf{C}$ does not derive g. Contradiction.

Suppose that $\mathbf{C}$ does not derive g. In this case, $\mathbf{C}$ is not an autonomous calculus, g is a non-autonomous number, therefore $\mathbf{C} \mapsto g$. Contradiction again, q.e.d.

Proof of the theorem

Let us assume toward contradiction that we have a calculus $\mathbf{C}$ with the Gödel number g s.t for every non-autonomous numeral c, $\mathbf{C} \mapsto c$, and there is no autonomous numeral d for that $\mathbf{C} \mapsto d$.

Suppose that $\mathbf{C} \mapsto g$. In this case, $\mathbf{C}$ is an autonomous calculus, g is an autonomous number, therefore $\mathbf{C}$ does not derive g. Contradiction.

Suppose that $\mathbf{C}$ does not derive g. In this case, $\mathbf{C}$ is not an autonomous calculus, g is a non-autonomous number, therefore $\mathbf{C} \mapsto g$. Contradiction again, q.e.d.

This theorem is Gödel-like because it shows that no inductive definition can be given for the notion „non-autonomous calculus" just like Gödel's first incompleteness theorem shows that no inductive definition can be given for the notion „arithmetical truth". And this proof uses an analogue of the Liar Paradox, too.