Some canonical calculi and logical languages
The concept of hypercalculus

András Máté

24.03.2023

The language of propositional logic

The language of propositional logic

We have an infinite sequence of propositional constants $p_0, p_1, \ldots, p_n, \ldots$ and two logical connectives: $\neg$, $\supset$. How to define the class of wff's as an inductively defined class over a finite alphabet, possibly avoiding the use of natural numbers?

The language of propositional logic

We have an infinite sequence of propositional constants $p_0,\ p_1,\ \ldots,\ p_n,\ \ldots$ and two logical connectives: $\neg$, $\supset$. How to define the class of wff's as an inductively defined class over a finite alphabet, possibly avoiding the use of natural numbers?

Instead of p_n we write $\pi\iota\ldots\iota$. We don't need numbers for indexes, but we need that our propositional constants can be distinguished from each other.

The language of propositional logic

We have an infinite sequence of propositional constants $p_0,\ p_1,\ \ldots,\ p_n,\ \ldots$ and two logical connectives: $\neg,\ \supset$. How to define the class of wff's as an inductively defined class over a finite alphabet, possibly avoiding the use of natural numbers?

Instead of p_n we write $\pi\iota\ldots\iota$. We don't need numbers for indexes, but we need that our propositional constants can be distinguished from each other.

Alphabet: $\mathcal{A}_{PL} = \{(,\ ),\ \pi,\ \iota,\ \neg, \supset\}$. Auxiliary letters: I for *index* and F for *formula*. The calculus $K_{Language(PL)}$:

The language of propositional logic

We have an infinite sequence of propositional constants $p_0, \ p_1, \ \ldots, \ p_n, \ \ldots$ and two logical connectives: $\neg, \ \supset$. How to define the class of wff's as an inductively defined class over a finite alphabet, possibly avoiding the use of natural numbers?

Instead of p_n we write $\pi\iota\ldots\iota$. We don't need numbers for indexes, but we need that our propositional constants can be distinguished from each other.

Alphabet: $\mathcal{A}_{PL} = \{(, \ ), \ \pi, \ \iota, \ \neg, \supset\}$. Auxiliary letters: I for *index* and F for *formula*. The calculus $K_{Language(PL)}$:

1. $I\varnothing$
2. $Ix \to Ix\iota$
3. $Ix \to F\pi x$
4. $Fx \to F\neg x$
5. $Fx \to Fy \to F(x \supset y)$

5*. $Fx \to x$

Comments to the above calculus

Comments to the above calculus

- The numbers in the left column are for the sake of reference only; they don't belong to the calculus.

Comments to the above calculus

- The numbers in the left column are for the sake of reference only; they don't belong to the calculus.
- The sign of the empty word can be omitted from the 1. rule.

Comments to the above calculus

- The numbers in the left column are for the sake of reference only; they don't belong to the calculus.
- The sign of the empty word can be omitted from the 1. rule.
- 5*. is a release rule: it erases an auxiliary letter. We can define the wff's of PL as the $\mathcal{A}_{PL}{}^{\circ} - strings$ derivable in this calculus.

- The numbers in the left column are for the sake of reference only; they don't belong to the calculus.
- The sign of the empty word can be omitted from the 1. rule.
- 5*. is a release rule: it erases an auxiliary letter. We can define the wff's of PL as the $\mathcal{A}_{PL}{}^{\circ} - strings$ derivable in this calculus.
- The language of propositional logic could have been defined without using auxiliary letters (see textbook p. 40). But it is not always possible to eliminate the auxiliary letters and they make our work simpler and more transparent even if they (or some of them) are not necessary.

Propositional logic as calculus (informally)

Propositional logic as calculus (informally)

We want to have a calculus that derives the provable formulas of this language.

Propositional logic as calculus (informally)

We want to have a calculus that derives the provable formulas of this language.
We include the (first five rules of) the previous calculus.

Propositional logic as calculus (informally)

We want to have a calculus that derives the provable formulas of this language.
We include the (first five rules of) the previous calculus.
New auxiliary letter: L with the intended meaning „provable formula”.

Propositional logic as calculus (informally)

We want to have a calculus that derives the provable formulas of this language.
We include the (first five rules of) the previous calculus.
New auxiliary letter: L with the intended meaning „provable formula".
The rules after the definition of the formulas will be just the usual axiom schemes of propositional logic.

Propositional logic as calculus (informally)

We want to have a calculus that derives the provable formulas of this language.
We include the (first five rules of) the previous calculus.
New auxiliary letter: L with the intended meaning „provable formula".
The rules after the definition of the formulas will be just the usual axiom schemes of propositional logic.
The rule of detachment (for '$\supset$') will appear again as a rule of our calculus.

We want to have a calculus that derives the provable formulas of this language.
We include the (first five rules of) the previous calculus.
New auxiliary letter: L with the intended meaning „provable formula".
The rules after the definition of the formulas will be just the usual axiom schemes of propositional logic.
The rule of detachment (for '$\supset$') will appear again as a rule of our calculus.
We need now a release rule for L.

The calculus of propositional logic K_{PL}

The calculus begins with the first five rules of $K_{Language(PL)}$ and continues as follows:

The calculus of propositional logic K_{PL}

The calculus begins with the first five rules of $K_{Language(PL)}$ and continues as follows:

6. $Fu \to Fv \to L(u \supset (v \supset u))$
7. $Fu \to Fv \to Fw \to L((u \supset (v \supset w)) \supset ((u \supset v) \supset (u \supset w)))$
8. $Fu \to Fv \to L((\neg u \supset \neg v) \supset (v \supset u))$
9. $Lu \to L(u \supset v) \to Lv$

9*. $Lx \to x$

The calculus of propositional logic K_{PL}

The calculus begins with the first five rules of $K_{Language(PL)}$ and continues as follows:

6. $Fu \to Fv \to L(u \supset (v \supset u))$
7. $Fu \to Fv \to Fw \to L((u \supset (v \supset w)) \supset ((u \supset v) \supset (u \supset w)))$
8. $Fu \to Fv \to L((\neg u \supset \neg v) \supset (v \supset u))$
9. $Lu \to L(u \supset v) \to Lv$

9*. $Lx \to x$

This calculus defines the class of provable formulas of propositional logic (shortly: the propositional logic) L_{PL}.

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.

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.

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.

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).

1. I The empty word is an index.

The calculus $K_{Language(FOL)}$

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

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

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

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

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

11.	$Ax \rightarrow xoPy \rightarrow Tz \rightarrow xPyz$	Application of predicates
12.	$Px \rightarrow Fx$	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.	$Vx \to Fy \to F\forall xy$	

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.	$Vx \to Fy \to F\forall xy$	
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 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 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 as 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 as 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 as 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 every canonical calculus will derive the code of itself.

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

How to represent an arbitrary calculus **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 into a string of our new calculus by letter.

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 into a string of our new calculus by letter.

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 into a string of our new calculus by letter.

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 into a string of our new calculus by letter.

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 into a string of our new calculus by letter.

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)

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 $\mathbf{h}_1$).