Mathematical language and the human mind and a proposal for a new mathematical framework

20715 In-Hwan Lee (talk on sequences)

https://faraway6834.github.io/unbeauty/privateNote/PersonalPersoanlVeryPersonal/FuckingPersonal

PART1. Reflections on numbers

Math and language

Numbers, rhetoric, etc. are concepts of magnitude. These notions of size are just a way of thinking, as we will see, and the formal dialect of the description of these numbers is math.

But language is not fundamental.

Let's begin.

A dissection of language.

What is a number

Before I talk about numbers, let me say this.

I **have** written about this topic before, treating number as a metaphysical illusion, an abstract object defined mathematically.

Below is that article

Articles We've Covered: Assertions

```
- Premises
 - Implications of the premises
 - 1. why the notion of conservation is unnecessary
in pictograms
 - 2. why an integral part of the abstract notion of
conservation in the mathematics of number should be
written in formal logic, the mathematics of number.
   - Conclusion.
```

Premise

Since we can only perceive the external image through our senses, the internal image (the self, assuming a language of thought L, and operating in L) can only make inductive inferences about the external image.

Implications of the premises

There is no way to deductively show that such an external picture is consistent, and thus no way to deductively rationalize that the external picture is described by formal logic.

1. why the notion of conservation is unnecessary in pictograms

Next: 2. Why the abstract notion of conservation is an integral part of the mathematics of number, and why it should be written in formal logic, the mathematics of number.

Table of Contents

Definition of Unary

```
Notation Unary ≡ [UnarySystem := λF. [Char := λx. "x is charactor"][
String := λx. x ∈ {c | Char(c)}□ [t := |x|] ][ F := λn:ℕ₀.λx:Char. ∀y
(except. y ∈ {S}ⁿ) ][x ← y := z (but. z → y = x) ][x↓□ y := z (but. z ↑□
y = x) ][ℙ₁ := ⏜ ∀String(x), "x is string" F□ ≡ F(n) x▪y ≡
\stackrel{x}{y} " ⌈x⌋ ≡▪x ⟦x⟧₁ ≡ x ⟦x⟧₍□₊₁₎ ≡ x ⟦x⟧ □ ▴ ⌈F□(x)⌋ ≡
F(n⁺)(x) ▾ ⌈F□(x))⌋ ≡ F(n−)(x) F□(x) ⟦ ⌈▴⌋ ⟧₁ ⌈F□(x) ≡ F₍□₊□₎(x) F□(x)
 ⟦ ⌈▾⌋ ⟧₁ ⌈F□(x) ≡ F₍□-□₎(x) F□(x) ⟦ ⌈▴⌋ ⟧₂ ⌈F□(x) ≡ F₍□□₎(x) F□(x) ⟦ ⌈▾⌋ ⟧₂
 ⌈F□(x)⌋ ≡ F(n÷m)(x) F□(x) ⟦ ⌈▴⌋ ⟧₃ ⌈F□(x)⌋ ≡ F(n□)(x) F□(x) ⟦ ⌈▾⌋ ⟧₃ ⌈F□
(x)⌋ ≡ F(□√n)(x) F□(x) ⟦ ⌈▴⌋ ⟧₄ ⌈F□(x)⌋ ≡ F(□n)(x) F□(x) ⟦ ⌈▾⌋ ⟧₄ ⌈F□(x)⌋
≡ F(super-root□(n))(x) F□(x) ⟦ ⌈▴⌋ ⟧₍₂₊□₎ ⌈F□(x)⌋ ≡ F(n ↑□ m)(x) F□(x)
 ⟦ ⌈▾⌋ ⟧₍₂₊□₎ ⌈F□(x)⌋ ≡ F(n ↓□ m)(x) x ▲ y ≡ y ⟦ ⌈▴⌋ ⟧₁ ⌈x⌋ x ▼ y ≡ y
 ⟦ ⌈▾⌋ ⟧₁ ⌈x⌋ x ▶ y ≡ y ⟦ ⌈▴⌋ ⟧₂ ⌈x⌋ x ◀ y ≡ y ⟦ ⌈▾⌋ ⟧₂ ⌈x⌋ F□(x) ↑□ F□(x)
≡ F□(x) ⟦ ⌈▴⌋ ⟧₍₂₊□₎ ⌈F□(x)⌋ F□(x) ↓□ F□(x) ≡ F□(x) ⟦ ⌈▾⌋ ⟧₍₂₊□₎ ⌈F□(x)⌋
F□(x) → F□(x) ≡ F(n → m)(x) F□(x) ← F□(x) ≡ F(n ← m)(x) ⏝]]
```

Usage.

> `(∃UnarySystem(F) s.t. ⊢ ℙ₁)( ⊢ T)`

Since ℙ₁ means "†the string is a tuple of characters and ‡the following syntactic equality holds", when ℙ₁ is true, †the string is defined as a tuple of characters and ‡the syntactic equality is accepted, and when ℙ₁ is false, †the string is not defined as a tuple of characters and ‡the syntactic equality is not accepted.

Thus, `s.t. ⊢ ℙ₁` means that for F, we enforce that ℙ₁ is true if we allow the definition and notation of the string, and false otherwise.

Define the constant "PlasticArrow": in the range $\mathbb{R}^2$ or $\mathbb{R}^3$, define the following

Plastic Arrow.

- Let `□` be the "basis vector" of the vector space $\mathbb{V} = \mathbb{N}_0^1$`.

Note that the number representation in Unary is a scalar in the vector space where `□` is linearly generated.

**Padic Plastic Arrow

- Let `➪` be the "basis vector" of the hyper-real vector space $\mathbb{V} = \mathbb{Q}^1$`.

**Standard Plastic Arrow

- Let `➡️` be the "basis vector" of the vector space $\mathbb{V} = \mathbb{R}^1$`.

Extra: realization?

The trap, of course, is that the "cross-section" of the "plastic-arrow parabola" must exist in order to create such a thing in reality.

However, the "cross section of a plastic arrow parish" can be expressed as the "equation of closed curves" `F(x, y) = 0`. (Hence, there exists a "negative representation F of the equation of the closed curve of the cross section of the plastic arrow parish").

"Realization" lol... Not even close, but hilarious. Lol.

The world is cold.

Consider the visual sensory language `L`.

Definition

```
L is a pictogram language.

  It simply lists pictograms.
```

Counting is defined as counting a pictogram.

Let's define a counting function "arithmetic system".

> `算子システム ≡ [算 := λx. $RF(x)^{-1}$ [R := λf.λx.λy.fyx]] (But. (∃UnarySystem(F) s.t. ⊢ $\mathbb{P}_1$)(⊢ T))`

Then,

Calculate(□) (算子システム)` is a function that inscribes a pictogram to go into the variable □.

> ex) `算("□")("□□□□□") = 5 (算子システム)`

In this case, the pictogram sentence is just a string interpretation, since it is approached from a math language.

As in the grammar translation method, the pictogram is interpreted as a string, not a pictogram,

This is the same as treating the letters of the ancient Mesopotamian civilization's language, written in wedges, as numbers.

Thus, the math gave it meaning, independent of the pictograms.

In fact, besides the pictogram, any string of characters is the definition of the counting function, so counting allows us to make an argument based solely on inductive reasoning that birds can be interpreted as numbers.

```
The concept of conservation is proved inductively, but deductively, assuming itself, requires formal logic, mathematics,
and is therefore "unproven" (unspeakable), and is a universal concept that is acquired after the precursor manipulator
has passed and the concrete manipulator has entered.
```

Consider conservation.

> We have a pictogram called `🍑🍍🍍🍍🍊🍓`.

If that pictogram is

> `🍑🍍🍍🍊🍓`.

to `🍑🍍🍊🍓`.

Therefore, the pictogram does not require conservation.

Therefore, there is no way to rationalize conservation in a pictogram bird image.

Therefore, even if the external bird image is recognized as a pictogram, there is no way to rationalize conservation.

How the universal concept of number is a universal intellectual concept

In terms of our understanding of number ("In mathematical terms, the amount of an object in the external world is interpreted as counting, and counting is an enlargement and rationalization of the result that "counting is fundamental"), number is a linguistically constructed element that we may not understand.

In this sense, dyscalculia can be seen as a disorder that fails to understand an incredibly common linguistic assumption: number.

Charactor/Text(String) is a Symbolization of Language,

Symbol is a shape that is promised to have a specific meaning.

And this is an incredibly universal concept,

In this sense, dyslexia can be seen as a disorder in which a person is unable to understand an incredibly universal human-created concept: the written word.

Next,

```
Finally, in the case of autism, as Temple Grandin says, the language within autism
can be a picture.
```

```
The author is autistic (I'm not defending autism because I'm autistic and I hate
autism), but in autism, some people (like me) tend to interpret language in their own
way.
```

```
The nature of language is not determined by conventional meanings, but inevitably
involves unexpectedness.
```

```
In other words, the essence of everyday language lies in the unexpectedness and
conventions (more like concepts, e.g., a person who frowns is angry) that underlie
human thought.
```

These unexpectednesses are,

> the intelligent surprise of language, which is due to the innate intelligence that humans universally possess.

> There would be a semantic surprise of language, based on our basic human concepts (e.g., that "you reap what you sow" means you get what you put in, which is the core of the surprise).

Thus, there are some individuals with ASD (such as myself in elementary school) who have a profoundly deficient understanding of the fundamental part of human thought, language.

From this perspective, the ability that is deficient in dyscalculia can be seen as a universal concept of number.

This concept of number requires some understanding of number,

A universal intelligence and understanding of numbers is only highly developed in humans, except for dyscalculia, and is not intrinsic.

Normal people have innate intelligence in many areas.

Not having it is called a disability, and frankly, I feel a little disgusted because I don't know such an obvious thing.

Objections to bringing in science

You should not use science as an example to refute inductively proven things, such as the sciences (natural, social, formal, etc.), because in this deductive discussion, it is argued that they are inductive but not deductive.

In this way, as soon as you rationalize a scientific method of inquiry or mathematics on the premise that it is done scientifically, you are committing self-reference, which is forbidden in mathematics (thus, a process such as bootstrapping is a rationalization that commits self-reference, and is therefore far from deductive argumentation (it does not make sense to call it inference to the best explanation)).

Also, science may not need to involve math. Suppose it is proven that math cannot be used in science, then the assumption that science must use math is broken, and a paradigm shift would occur.

The assumption that science must use math is equivalent to the assumption that science must use formal logic.

However, the assumption that science should use formal logic is the assumption that the science can be described in a consistent formal logic, so for this assumption to be true, we need to have rules for using logic (axiomatization, language, notation, etc.) and the assumption that science should investigate consistent objects, so it can only be supported deductively by evidence that axiomatization is available and evidence that the objects being investigated are scientific.

2. why a key part of the abstract concept of conservation in mathematics, called number, must be written in mathematics, which is formal logic: Part 2: Logical treatment In reprinting, algebraic expressions are logically interpreted and computed.

This is covered later in the PPT on slide 63.

In conclusion, the "number as the core of the abstract concept of conservation" in mathematics cannot be established independently without mathematics. (However. I do not treat rhetoric and numbers in language as numbers because they are not mathematical interpretations. In language, numbers can be interpreted in unexpected ways, so they are **not really consistent**; in poetry, one can suddenly become two.)

Conclusion

Thus, even if we accept the bird image as a pictogram, the pictogram bird image does not require the concepts of number, math, or conservation; they are only additional concepts in language, not immaterial entities.

This article is incorrect

-.

In interpreting the pictogram, we can use our underlying knowledge to know the number.

I don't think I've refuted the question of a priori, empirical, linguistic, or instinctive.

Step 1. Antecedents of thoughts

A thought has a content, where the thought is just a straightforward thought, and the content is like the contents of a box, but abstract and indescribable. I never said that the content is a text. It's content, and we can't say what that content is.

We can't say what the next thought is either. We don't know what happens to the next thought.

But we try to think, we don't try to become vegetative in a mental way, unless of course we consciously think, "I'm going to become vegetative in a mental way," then it's a thought.

When we say that human beings are higher, let's discard that idea.

If you have any doubts about the meaning of the words, discard them and listen.

Definitions

To formulate a thought, to propagate a thought, to understand a thought in a way that is natural to me, in a way that is my basic ability, and to add to a thought by something that comes from outside of me. These are called "pioneering"/"propagating"/"utilizing"/"adding", and it is axiomatic that our thoughts have these characteristics.

Addition is the only one here that cannot be done actively. (Asking for something is also passively accepting it).

And propagation is the sharing of thoughts. You don't know what you're getting.

But to use it, you use the next natural thought. The ability to use these natural thoughts is called latent ability and is called talent.

Language and Size

Most of us have the ability to acquire language as a talent, and we have the ability to acquire size as a talent.

These are often talents, but not in special cases.

Language and size are learned, I think.

Language and size have spread.

Sense of size

In fact, some things, like size, come from things without anyone propagating them.

Somehow, it's already there, we think of it as such, and there's a part of us that can't get out of our perspective on it.

There is a "hard mold" that cannot be removed. (Example: When I think of Yun Seok-yeol, I think of martial law... just kidding, this is not a good example lol)

Thought precedes action.

Very often, these concepts are added through utilization.

Especially if we need them to survive.

We have the will to think, and language and size are manifestations of that.

The reason these abilities are not innate, essential roots is that there are exceptions to the rule.

Most of the people I've seen have the ability to learn language or size as a natural talent.

However, these are often things that can or do change the flow of thought,

These "underlying" thoughts are thoughts, not essential human abilities.

Thought precedes action.

Step 2. Size Concepts

The concept of size is something that most people think about.

It stands to reason that not thinking about it is not a basis for discrimination against dyscalculia.

The quantified concept of size is number, and mathematical languages are languages that deal with numbers.

However, mathematical languages are described as formal languages.

Therefore, it is quite natural that some people may not have access to narratives based on size or math that deal with size concepts or math.

Language, number or mathematical descriptions, and the concept of size are only available to us when we are trained to think in them, not when they are natural to us.

Let me summarize in one sentence.

In terms of thinking, not in terms of other things,

Language, numbers, mathematical descriptions, and magnitudes, in terms of what we think about, are thoughts, not instincts unless we are born with a talent for them.

I don't know how much easier it is to explain this.

It's not a talent or anything like that, it's just a thought, an idea, a way to survive (even lions have a size concept).

Above.

Unexpectedness theorem

True Mean Term Definition

- True Mean: Unexpectedness (inner meaning)
  - Shell Mean (껍대기 뜻; 쉘 민) : 겉뜻
  - Exceptless (Exceptless) : Shell Mean = True Mean
  - Hyper-Completeness: Allowing "This sentence is False".
  - Black and White Proposol : ⊢ (¬Hyper-Completeness)
  - Simply Mean Proposol : ⊢ Exceptless
    - Simple languages: languages for which Simply Mean Proposol is antisynthetic

It is natural that a logic whose axioms are black and white propositions can be called black and white logic.

We can also see that the simple language of black and white logic is a formal language.

Supporting explanations for organizing paragraphs

There are some things in our language that should be interpreted as they appear and some that should not,

What should be interpreted as it seems is called simple language (in this case, there is no exception to language interpretation, shut up and mean what it seems).

And there is a language that is not a simple language (in this case, various unexpectednesses are involved in the interpretation of the language, so we cannot simply assume that the meaning is what it seems).

Also, logic can be like the philosophy of Chuang Tzu: "No one has ever lived longer than a newborn baby, so Pang Chao (a legendary god who lived for over 760 years) must have died early",

In black and white logic, it is impossible to be x and not be x at the same time, making the Black and White Proposition true. (Fun Fact: Simple language is a logical corollary of black and white logic)

Axioms

1. words have a True Mean and a Shell Mean

TrueMean Theroem

Even when Hyper-Completeness is true, "Meaning of words ≠ True Mean" is also true as True Mean (after all, "Meaning of words = True Mean" by deduction from a sentence that is true as a premise, which is possible by hyper-completeness), and (Hyper-TrueMean Lemma)

Even when it is not Hyper-Completeness, "Meaning of words = True Mean", so meaning of words = True Meaning (Formal-TrueMean Lemma)

The meaning of a word is the True Mean. (TrueMean Theroem)

A. Formal-TrueMean Lemma

- ¬Hyper-Completeness, ⊭ Meaning ≠ True Mean ⊢ Meaning = True Mean

B. Hyper-TrueMean Lemma

- Hyper-Completeness, Lemma ≠ True Mean ⊢ Lemma = True Mean

C. TrueMean Theroem

- Theroem = True Mean

Proof

1. meaning of words ≠ True Mean [Hyp]

2. Hyper-Completeness [Hyp].

3. "The meaning of a word ≠ True Mean" is also True Mean.

[Paradoxic Lemma]

3. "The meaning of words ≠ True Mean" is also True Mean and true, so the meaning of words = True Mean

4. meaning of words = True Mean

In the following,

Meaning ≠ True Mean, Hyper-Completeness ⊢ Meaning = True Mean ··· (1)

1. semantics ≠ True Mean [Hyp].

2. ¬Hyper-Completeness [Hyp].

3. "The meaning of a word ≠ True Mean" is also True Mean [Paradoxic Lemma].

4. contradiction

In the following, ⊭ Meaning of words ≠ True Mean ⊢ Meaning of words = True Mean

(1), (2) ⊢ Mean = True Mean

Q.E.D.

Commentary

Prove that True Mean is always the meaning of a word,

Suppose there is a statement whose True Mean is not the meaning,

then the True Mean of that statement is the True Mean of the statement that the True Mean does not mean.

This is the Paradoxic Lemma

In the following, we will argue that it is true and false according to Hyper-Completeness.

Each

```
Situation 1. From a Paradoxical Lemma, Hyper-Completeness

Since Paradoxic Lemma can be true, Hyper-TrueMean Lemma is true

Exit situation 1

Situation 2. From a Paradoxic Lemma, Non-Hyper-Completeness

Since the Paradoxic Lemma is a contradiction, its predicate "Meaning ≠ True Mean" is false.

Therefore, the Formal-TrueMean Lemma is true.

Exit Situation 2
```

Below,

by Context 1, Context 2, by Deduction,

the True Mean is always the meaning of the word.

Therefore, TrueMean Theroem is true

Q.E.D.

Exceptless Thorem

Exceptless ⊢ Meaning of Words = Shell Mean

Proof)

1. Exceptless [Hyp].
2. Thorem = True Mean [TrueMean Theroem].
3. Shell Mean = True Mean
4. meaning = Shell Mean [Conclusion]

From the following, it is obviousthat `Exceptless ⊢ Mean = Shell Mean`.

Commentary

As per the TrueMean Theroem proved earlier,

Word Mean = True Mean

Shell Mean = True Mean if, and only if, Shell Mean = True Mean

(in other words, A = B = C, so A = C)

PART 2. Propose a math scheme to leverage underlying function argument-based operations (equation-like; minds are descriptions of operations)

Introduction

I sometimes get the impression that math is arrogant and rushed, so I'd like to make a suggestion.

ALKALIC

Alkalic : Alkalic Linear-algebra + Königsberg Axiom + Lambda Incoding Calculate

Even in model theory...

`M = (ℕ, 0 = ∅, s(x) = x ∪ {x}) Instead`

`by writing the product of sequences as an approximation function,`

`M = Π<ℕ, [0 := ∅], [s := λx. x ∪ {x}]>.`

`in that the same ordinal numbers can be defined,`

`Structure and variable assignment are now inside functions.`

`Structures can now be written to in non-structural logical inferences`

Alkalic Linear-algebra

∀x (each unique)∃!n(x) s.t. n = ObjectID(x) ∈ Scala

- AlkalicVectorSpace = Scala□ [t := |Scala|]
- AlkalicMetrixSpace = AlkalicVectorSpace□ [t := |Scala|]
- SetTheorem ∈ AlkalicMetrixSpace
  - Notation Definition m ∈ n ≡ SetTheorem□□(□)□□(□) [oi := ObjectID]

Lambda Incoding Calculate

Implement a function with lambdas, algebraic functions, and polyvalent functions in an alkalic algebra, for an AlkalicVectorSpace or oidfield = Σ□ ObjectID e□, given a Tensor input.

Variant terms can be~~名~~ ~~vanished~~ as arguments to the theory

explained

Königsberg Axiom

```
⊢ KönigsbergAxiom(x, y, Φ) := (x = y ↔ (Φ ↔ (Φ [x := y])))
```

TIP: It's a substitution.

(There's an error, I'll fix it)

`Alkalic`'s proof of type;
`Alkalic-Proofmood`

```
Rule `using x = y → (Φ ↔ (Φ [x := y]))))
```

```Alkalic-Proofmood

□.1. using x = y → (Φ ↔ (Φ [x := y]))))

□.2. x = y

□.3. Φ

□.4. Φ [x := y]

□.5. Φ ↔ (Φ [x := y])

```

`Rule `using (Φ ↔ (Φ [x := y]))) → x = y ``

`Principle: `x = y ↔ (Φ ↔ (Φ [x := y]))))` from `Φ ↔ (Φ [x := y]))`, showing that it is equivalent to the conclusion (line 5)`

```Alkalic-Proofmood

□.1. using (Φ ↔ (Φ [x := y]))) → x = y

□.2. Φ

□.3. Φ [x := y]

□.4. Φ ↔ (Φ [x := y])

□.5. x = y

```

```
Internally, it only continues to work if the conclusion (line 5) is true, and as the conclusions are listed, the lemmas
are listed until the last line, Theorem, is reached, so that the statement's truth is seen as a lemma.
```

Rule : `Starting Listup Hyperthesis`

Start the hyperthesis listup in advance.

Rule: `Quit Listup Hyperthesis`

No longer receive hyperthesis

Rule : `Starting Another Subproof`

Creates a new stackframe, starting a new subproof

Rule : `Quit Another Subproof`

Finish the subproof, add it to the Lemma List, and pop the stackframe

HAPA - I

```
Rule : `APAristotel-y` (nonHyperVersion type proof only)

 >

 > > Using an external theorem called the HAPA Theorem, we show that
if y = x and y = z, then x = z.

 >

 > > Rule : `APAristotel-z` (nonHyperVersion type proof only)

   > > Using an external theorem called the HAPA Theorem, show that
  if x = z and y = z, then y = x.
```

HAPA Theorem (Hyper Alkalic-Proofmood Theorem)

The basis for Alkalic-Proofmood non-HyperVersion type proofs.

At the same time, the only Alkalic-Proofmood HyperVersion proof of the type proof

When `y = x`, `y = z` are hypothesized,

Through the rule `using x = y → (Φ ↔ (Φ [x := y])))`, it is shown that `y = z ↔ x = z`, i.e,

Grammatically, `y = z ↔ (y = z ↔ (y = z [y := x])))`= `y = z ↔ (y = z ↔ x = z)`, so that

is a surplus scheme for expressing y = z ↔ (y = z ↔ x = z).

```
Alkalic-Proofmood (Power Up - Version)
```

Added a qualifier that must be prefixed to the proof, which can be nested by creating partial proofs, so you can't attach each feature at the same time.

- AristotelProof (default when unspecified): Prove in the way of traditional proofs

- DavidHumeProof : For $\Phi_{\square}$, write a magical induction every t lines, and the column next to it is a space to write if $\Phi_{\square}$ is used in the induction proof. On the last line, without numbering, write the type of inductive proof: ∴ $\Phi_{\square}, \Phi_{\square+1}, ..., \Phi_{\square} \vDash \Phi_{\square+1}$` (strong), `∵ $\Phi_{\square} \vDash \Phi_{\square+1}$` (weak), `∵ $\mathrm{Mod}(\Phi) = \mathbb{N}$` (general mathematical induction).

- EuclidianProof: inductive (different from HegelianProof, it is inductive); at the end of the proof, `∵ $\bot$ ∴ $\nvDash \neg$

Φ ∴ Φ` at the end of the proof (`$\neg\Phi$` is the conclusion).

- HegelianProof : Disproof (used to prove "not a conclusion" because the conclusion is a negative; in the previous version, if a logical error occurs, the program terminates, so AristotelProof, which does not terminate after disproving the error, was needed.)

Also to avoid freezing the verification program,

- PreviewVersion: for this partial or complete proof, add it to the preview list, except for pointing out errors after the program stops, because it is not groundless, and if you make an error referring to this part of the proof, it will spit out the "Reference on Preview" error log separately and error like a normal error.

 - DebugVersion: Debug as soon as an error occurs, debug it, and go through it, **applies to the entire program**.

 - ConjureVersion: speculative, inserting `�` wherever to avoid stopping; this part-whole proof is treated as a hyperthesis.

 - NormalVersion (default when unspecified): traditional

Finally, to facilitate the computation of polynomials,

partial proof form called `Polynomial Simplify` and factorize the following, or solve for `P(x) = 0` (in the latter case, specify `using P(x) = 0 Algorithm` in advance) Accept the proof without error.

A. Use the `LinearSimplify` command to simplify first-order expressions with multiple unknowns, based on the LinearSimplify Theorem

B. The `Substracting [y := xⁿ]` command allows you to work with a pre-substitution as if it were a first-order expression, for a statement `Φ` with x^n replaced by y, according to the true statement `y = xⁿ` in the `Substracting Variable` field, until you get Φ [y := xⁿ]. (Substituting Variable Fields; `Substracting Variable Field Proofs`)

C. Use the `Solution (a, b, c, d, e)` command to factorize (approximate formulas) / expand (Bietz's theorem) quadratic expressions.

D. Apply `[x := t + b/na]` via the `TschirnhausTheoremSubsituate (n, a, b, x)` command, again relying on the Substracting command for smoothness of proof (although it's actually a simple sentence (`[x := t + b/na]`) that could be defined syntactically without needing to).

E. The `synthetic division` qualifier in partial proof grammars, using the assembly method (since the ellipsis is used for multiple rows of a fixed column in a logical induction, it has the disadvantage that when used here, the rows are polynomials and the order of the computation is columns, so it has to be reversed).

F. `Alright synthetic division' qualifier, use the normal assembly method and compile it as a synthetic division in the preprocessing step.

G. For distributive laws, put everything inside `distribute[ target ]` and distribute according to the Generalized Theorem of Distributive Laws for this proof system.

H. `AlgebraicFormula`: specifies that it was computed using a previously proved multiplication formula.

I. Gaussian Eimination or ERO & Substitute : Gaussian elimination or addition/substitution

J. System of Quadratic Equations by Quadratic Form : Solving quadratic equations by quadratic form

K. System of Quadratic Equations by Cubic Form: Solve trigonometric equations in cubic form

L. Règle de Cramer : Solve with Cramer's formula

M. `Extraneous Root is (□)`: Specifying an irrational root

N. `PolynomialFractionize`: Fractionize polynomials

O. `SolvePolynomialFraction`: Solve the corresponding value

P. `Fractions Solution is (□)`: specify the solution

Syntax for automata in formal proofs

If the line `□.` is a partial proof, represent it as `□.line.`.

However, if there is only whitespace and `-`, `-`, `-`, `-` between the line and the line, the line is considered readable and commented out.

Also, if a line is preceded by a dotted `|`, followed by a specific column of the string, and the end of the line is connected to a comment in the `-` type described above, it will not be error-checked separately.

Otherwise, it is a column separator and is not considered a comment.

Finally, the `[NOTE : ]` format is considered a comment.

If it is located inside a Markdown document, it will only read the parts that are codenamed HAlkalic-Proofmood (Hyper Version), Alkalic-Proofmood (Normal Version), and PowerAP (Power Up Version). It also renders the partial proof code in the markdown as a partial proof.

Finally, the htmlized and organized rendered view will need to have LaTeX notation added.

(Note that the html view has not yet been type-checked, so it is compiled, not executed. To run it, you need to run it in an executor, which will parse the document, interpret it with the supplied [labare](https://faraway6834.github.io/unbeauty/privateNote/Proof/labare)-[unbare](https://faraway6834.github.io/unbeauty/privateNote/Proof/unbare) code (labare-unbare is not an interpreter language, but a formal interpreter, user-friendly interpreter, and low-level compilation language.), and is reviewed; in hindsight, it is better called a proof verifier than an executor, since it does not execute any programming language, but only checks for correct use of the inference rules, and provides a review (with errors, logs, and status).

Logical treatment When reproduced, algebraic expressions are logically interpreted and computed.

Otherwise, no logical symbols can be derived from the logical flow of interpretation.

The computation of an expression is done by assigning its values, e.g.,$\bar{X}$(, f($\bar{X}$)), and to reject this method is to reject logic. (No one has ever lived longer than the firstborn baby, so even Pang Chao (a legendary god who lived over 760 years) died prematurely)

Deriving Logical Corollaries in the Flow of Logical Analysis of Algebraic Expressions

Step.1. Show that a model set as a set satisfying an equation is a solution set

First, define a polynomial function P as follows.

> `P :≜ λA. λx. Π□ x - A□`

And define an equationalized logical function Φ such that

> `Φ :≜ λf. (f(x) = 0)`

And finally, define a polynomial equation λf.

> `p :≜ φ - P`

Then,

> `Mod(f(A)) = {x | x ⊨ (Π□ x - A□ = 0)} = {x̄ | Π□ x̄ - A□ = 0} = {A□ | ∀i}`

which is obvious.

`Step.2. Derive the Logical Corollary`

For polynomial equations $p(A)$, $p(B)$,

0. $\mathrm{Mod}(p(A)) \subseteq \mathrm{Mod}(p(B))$

1. $\{A_i \mid \forall i\} \subseteq \{B_i \mid \forall i\}$

2. $\forall i\ A_i = B_i$《Note: that's a leap, it only works if you sort the array》.

3. $\exists C\ P(B) = P(A)P(C)$

4. $p(a)|p(b)$

By,

For any polynomials f, g, the polynomial equation $\Phi(f) \vDash \Phi(g)$

if this side of

$f \mid g$

Show that ¬x = T - x is interpreted as (Warning: not a proof)

A. proof of `x ≠ T ⊢ T ± x ≠ (1 ± 1)T`

0. `x ≠ T` (non-recursive premise)

1. `T ± x ≠ T ± T` (binomial by function `(T ±)`)

2. `T ± x ≠ T ± T = (1 ± 1)T` (computed as an extension of 1)

3. `T ± x ≠ (1 ± 1)T` (summarizing the expression in 2)

… ■

Then

B. proof of `⊭ (1 + 1)T = 0 ∨ (1 + 1)T = T`

0. First, let's divide it into part A bỳ⊭ (1 + 1)T = 0` andpart B bt `⊭ (1 + 1)T = T`.

1.A. (1 + 1)T = 0 (integral formula reprinted)

2.A. (1 + 1)T = 2T = 0 (extension of 1.A.)

3.A. T = 0 ⊨ 2.A. ⊨ (1 + 1)T = 0 (deduction)

4.A. ⊭ T = 0 ⊨ 2.A. ⊨ (1 + 1)T = 0 (deduction)

5.A. ⊭ (1 + 1)T = 0 (induction)⋯ ⊥

6.A. ∴ ⊭ (1 + 1)T = 0 (deduction)⋯ ■

1.B. (1 + 1)T = T (inductive formula reprinted)

2.B. (1 + 1)T = 2T = T (computed as an extension of 1.B.)

3.B. T = 0 ⊨ 2.A. ⊨ (1 + 1)T = T (deduction)

4.b. ⊭ t = 0 ⊨ 2.a. ⊨ (1 + 1)t = t (deduction)

5.B. ⊭ (1 + 1)T = T (deduction)⋯ ⊥

6.B. ∴ ⊭ (1 + 1)T = T (deduction)⋯ ■

And so on

```
C.1. A, B ⊨ (x ≠ T ⊢ T - x ≠ (1 - 1)T) (follows from A, B)

C.2. A, B ⊨ (x ≠ T ⊢ T - x ≠ (1 - 1)T = 0T = 0) (a continuation of C.1.)

C.3. A, B ⊨ (x ≠ T ⊢ T - x ≠ 0) (summarizing the expression from C.2.)

C.4. A, B ⊨ (T - x = 0 ⊨ x = T ⊨ x) (Deductive Reasoning: Treatment from
C.3.) 《Note: Interpret the meaning of "x = T" in terms of "the truth value x
is true."

C.5. A, B ⊨ (T - x = 0 ⊨ x) (summarizing the expression from C.4.)  《Note:
Interpret the meaning of "x = T" in the proof of C.4. in terms of "the truth
value x is true."

C.6. Content of C.5. ⊢ ¬x = T - x (final conclusion)  《Note: Interpret the
meaning of "x = T" in terms of "the truth value x is true."

  Q.E.D.
```

$x \wedge y \wedge xy$

Show that $x \wedge y$ is interpreted as xy.

The solution of the equation $(T - x)(T - y) = 0$ for T is

$x = T \vee y = T$.

Therefore, $x = T \vee y = T \models T - (T - x)(T - y) = T$,

$x \vee y = T - (T - x)(T - y)$, which is interpreted as $x \vee y = T - (T - x)(T - y)$.

Then, by De Morgan's Law, $\neg(\neg x \vee \neg y) = x \wedge y$, we have

$T - T + (T - T + x)(T - T + y)$

$= xy$.

$\cdots$ Done.

Meaning of an equation: as a predicate in predicate logic (functional logic), tentatively using a specialized quantifier, potentially using a predicate quantifier

The impossibility of the equation P(x) = 0 means that

∄P(x) = 0, while

∀P(x) ≠ 0 and ··· ①.

If the equation P(x) = 0 is not infeasible, then

∃P(x) = 0. ··· ②.

The equation P(x) = 0 is negative,

Since it is a negative equation,

∀P(x) = 0. ··· ③ ···

In ①, for the inequality P(x) = 0,

P(x) ≠ 0 is negative, and ··· ④

For a nonsingular equation P(x) = 0,

P(x) ≠ 0 is indefinite ··· ⑤.

Then, according to ③, let us define the following,

`Φ :≜ λf. (∃f(x) = 0)`

P :≜ λf. (∃f(x) ≠ 0)

Then we know that

Since

By ④, if Φ(f) is false, then P(f) is negative.

By ⑤, if Φ(f) is negative, then P(f) is false.

If Φ(f) is true, then there exists an x satisfying $f(x) = 0$.

If P(f) is true, then there exists an x such that $f(x) = 0$ is unsatisfiable.

If we want to create a negative inequality

$\neg\Phi(f) = P(f)$, $\Phi(x) = \neg P(f)$,

negate the nonsingular equation Φ(f), or

negate the infeasible equation P(f).

For the predicate P

$\mathrm{Mod}(P) = \varnothing$ if $\nexists P(x)$ then $\nvDash P$

if and only if

$\mathrm{Mod}(\neg P) = U$ iff $\forall \neg P(x)$ iff $\vDash \neg P$

Thus, the equation is basically a predicate of characterization, which can be used

PART 3. Rationale that the equation is easy

SIOT[Siot], Standard IO Table (Mean : Standard Input and Output Truthtable)

## Instruction structure.

논리 연산을 [FAN, Function AbbreviatedNaming](https://faraway6834.github.io/unbeauty/privateNote/functional_petterns_commandstyle_naming#%EA%B8%B0%EB%8A%A5%20%EC%B6%95%EC%95%BD%EB%AA%85%EB%AA%85%EB%AA%85\(%EC%B6%95%EC%95%BD%EB%AA%85%20%EB%AA%85%EB%AA%85\)%20\(Function%20AbbreviatedNaming\)%20%EC%B2%B4%EA%B3%84%20%EC%84%A4%EB%AA%85)으로 정의하여,

Spring as a tool to use as a feature.

### SIOT's interpretation.

Then, show that SIOT is a calculation table that represents all of its operations (examples of calculation tables are the addition table, multiplication table, square root table, and common logarithm table).

### Interpretation of logical operations.

The logical operations you have learned and the texts with examples,

When utilizing them, show the equivalence of abbreviated name operators simply written in FAN by raising SIOT.

Then you can just use them because they are equivalent.

### Understanding the name of logical operations.

It is tempting to criticize dichotomous logic as extreme,

but that's the way math is.

Slow.

### Interpretation of SIOT.

Then, show that SIOT is a calculation table representing all of its operations (examples of calculation tables are the addition table, multiplication table, square root table, and common logarithm table).

### Interpretation of logical operations.

The logical operations you have learned and the texts with examples,

When utilizing them, show the equivalence of abbreviated name operators simply written in FAN by raising SIOT.

Then you can just use them because they are equivalent.

### Understanding the name of logical operations.

It is tempting to criticize dichotomous logic as extreme,

but let's call it math.

## Appearance.

We call the truth-assigning cells Input, put them on the left side of the table, and put the rest of the operations on the right.

Then, at the top of the truth table, append only two large cells, Input and Output.

Left and right sides of the ### operator (operands, truth values assigned to)

The left and right sides are the targets of the operation, called Left Side and Right Side in English.

We call them LS and RS for short (acronyms)

Yap

### True and false

True and false are specialized pieces of information,

In order for a computer to represent them, it needs to store the information (tip: you can convince it that "you can actually write a program to do it")

In a computer, true = yes = 1, false = no = 0

### Naming conventions for operators

L□R■ means that L is □ and R is ■.

It is a proposition (discernible), has a truth value, has an operational value as its meaning, and is expressible.

When 0000 is a regular expression: AZVC (AlwaysZeroValueCommand)

When 1100 is a regular expression: GRSV(GetRSValue)

When 1010 is a regular expression: GLSV(GetLSValue)

When the complement of a regular expression is understood as negative: use the non-prefix, abbreviated `N`.

Operators other than #### are logically derived.

You don't need to understand that part. You only need to write the result, and you can use it later if you learn it.

CMD Algebra: 1. CMD Dimensions

M Operations

- Each function in Chapter M, which is an uncurled D-operation.

M composition

- A composition transformed in D-space.

- Determined in Chapter M by tensor product binding in D-space.

M Notation

- A tensor product bound on the M -notation definition, with square brackets for each M -basis, and a tensor product bound on the M -notation definition.

M-correspondence

- Always corresponding to a field D such that the field M always exists with respect to the field D.

C-space and its organization

- Slots.

- M and D (CMD, as you may have noticed).

- The existence of a C-correspondence that properly fulfills the M-correspondence and the D-correspondence.

- There exists a C-configuration in which M and D are modified by the C-definition, and the C-response modifies the C-internal space (which is actually what we're talking about).

C notation

- An internal outsourcing machine to be invoked in a C slot by evaluating over the higher dimensions that make up the C composition when each definition is added. (Composition)

CMD

2. CMD Symbolization

- To define the indexing of a sequence □ as a graph on a coordinate plane partially excluding the line segment to the right of the top line of □, and to define the remaining part to be written, so that the symbols drawn in the form of a graph for symbols are written as symbols for notation.

- With CMD: just write it in Polish notation without parentheses.

(notation)

3. with CMD.

In a minimalistic way

- A. lower the dimensionality of the part you are defining

- B.1. Include the least semantically optimal inclusion.

and so on, removing the dimensions to be organized at the top for CMD evaluation in favor of semantic organization. (Algorithm)

Function Abbreviated Naming (Function AbbreviatedNaming) Organization

Declaring a functional operator in CMD algebra,

It is a usage system that allows you to learn the concept of naming operators by abbreviated names, from Function, which means function, to the expression of a pattern for a blank that acts as a function.

## Function shorthand naming

Write as an equation after `#`, not as a binding notation.

Function Abbreviated Naming (Function AbbreviatedNaming) Scheme Description

Creates a function abbreviated naming scheme that defines a function abbreviated naming operator as a function abbreviated name for a feature pattern with numbered blanks, where the specific input of the function abbreviated naming operator, corresponding to the numbered blanks, is interpreted by simply unabbreviating the abbreviated expression and replacing it with its meaning.

Thus, semantic equality is established.

## Notation

Since you can't put a number on top of a blank space,

a string of characters such as `🄰🄱🄲🄳🄴🄵🄶🄷🄸🄹🄺🄻🄽🄼🄾🄿🅀🅁🅂🅃🅄🅅🅆🅇🅈🅉  ` replaces the number space.

### Example: Trinity function abbreviated name operator

(Note: markdown specifies the source type as FAN, there may be some overlap)

Definition

```FAN

# Trinity = 🄰(🄱 - 🄲) + 🄲

```

Use (value becomes 12 here)

```FAN

Trinity 1 12 21

```FAN

```
LAFTF 1.1Logic-Algebra Formular TransForm 1.1 (old version incomplete)

def)

    ⎧ To : {x|Set(x))} × {x|Set(x))} → {z|z={f:x → y | p(x, y)}}
p: ⎪
    ⎨.
    ⎩ To : x, y ↦ {f | f : x → y}

def notation) A To B ≡ To(A, B)

  def written) A To B ≡ A 2 B

----

def) (∃! int ∈ Bool2{0, 1} ∃! int-¹)(int-¹(x) ≡ (x = 1) ≡ (x ≠ 0))

  def) bool ≡ int-¹
```

Symbolic induction

(bool - f)(x, y) :≡ bool(x) ∧ bool(y)

    (x̄, f(x̄)) = (x̄, ×(x̄)) (but <×, Y¡□□>)

    ∵ ⌠ f(0, x) = 0 = 0 × x

⎜ f(1, x) = x = 1 × x

⎨ ⎨

⌊ \[x, y\]□ = 1 = \[x, y\]ₓ (h=f, limits of symbols...)

    ∵ f(n, x) = f(x, n) (but n ∈ Y¡□□)

∴ f(x, y) = xy

It simply means that they are equal...,

I'm going to take a wild guess that this is self-evident if math has any validity.

¬.

(bool - f)(x) :≡ ¬ bool(x)

f(0) = 1, f(1) = 0

∴ f(x) = (0-1)/(1-0) x + 1 = 1 - x, so when we plot a point on the graph, if we change the number from f(0) to f(1) at the point we want to see, we get a decreasing number of 1s.

(No, that's not the right word, it's a summary, but it's a summary nonetheless)

Then, according to Conjuntive Normal Form

⎧ $b = \sum A_{\square} 2^{n-1}$ (n=1~4, but. $\bar{x}$ = ((0, 0) $\wedge$ $f(\bar{x}) = (A_1, A_2, A_3, A_4)$)

⎪ (0, 1)

⎪ (1, 0),

⎨ (1, 1))

Hexf(b) with ⎩ Hexf(b) = f is composable with xy and 1-x

where $x \oplus y = (x \vee y) \wedge (\neg (x \wedge y)) = (x \wedge \neg y) \vee (y \wedge \neg x)$,

$\mathrm{int}(x \oplus y) = \mathrm{int}(x \vee y) - \mathrm{int}(x \wedge y)$

$\because$ $f_1(x, y) = xy$

$f_2(x, y) = x + y - xy$

$f_3(x, y) = \mathrm{int}(\mathrm{bool}(x) \oplus \mathrm{bool}(y))$

$\mathrm{Hexf}^{-1}(f_3) = \mathrm{Hexf}^{-1}(f_2) - \mathrm{Hexf}^{-1}(f_1)$

Also,

$\mathrm{int}(x \oplus y) = \mathrm{int}(x \wedge \neg y) + \mathrm{int}(\neg x \wedge y) - \mathrm{int}(\neg x \wedge y)\, \mathrm{int}(x \wedge \neg y)$

$\mathrm{int}(\neg x \wedge y)\, \mathrm{int}(x \wedge \neg y) = \mathrm{int}((\neg x \wedge y) \wedge (x \wedge \neg y)) = \mathrm{int}(\bot) = 0$

Shift.

`This was supposed to be a space for me to say thank you for giving me the spiritual strength to come up with an idea in France...`

`■■If■this■is■not■the■life■I■want■to■burn■to■death■how■I■have■to■try■to■get■the■LAFTF1.I went to wikipedia to get the xor symbol, but I found it somewhere else, and then I went to the English wikipedia to get it, and then I went to the English wikipedia to get it, and then I went to the English wikipedia to get it, and then I went to the English wikipedia to get it, and then I went to the English wikipedia to get it, and then I went to the English wikipedia to get it, and then I went to the English wikipedia to get it. 식이라고 ■ㄴ좋은 이론이 선행되었는데 이거 모르고 위의 내용 ■■ 혼자 하나하나씩 유도한 내가 ■■이지 ■■이 ■■이 ■나 ■생 ■반 ■해네 ㅋㅋㅋ ■친 뭐 앞으로 서술할 비트연산도 마찬가지겠지 뭐 하 ■발 인생.`

`shl□(x) = shl□(2□-¹ + x) (β = B (Bits) Same, limits of symbols, overlap for reference, and ■be corrected from a to β once more. Ouch)`

`(shl□)□□ (Λ ≡ lm, same limits, would be written like this, ⁽\*⁾)`

`(shl□)□ (since there is no subscript a... ■Bee ⁽\*⁾)`

Principle; Swear less, I'm offended even if I censor myself, I'm not good enough

Originally (=in the original handwriting), it should be written, "Such a beautiful fact," but I still feel the same way.

$f(x) = x - \lfloor x \rfloor$

$f(x) = f(1 - x)$

f□□ = 1 (Note: $^{(}$*$^{)}$)

$f_a = 1$ (note: $^{(}$*$^{)}$)

$g(x) = 2f(x)$

g□□ = f□□, $g_a = 2f_a$, f□□ = f_a,

g□□ = k, $g_a = 2k$ ($k = f_x$)

Then

$(shl_\beta)_{□□} = k$, $(shl_\beta)_a = 2k$ $(k=2^{B-1})$

$shl_\beta(x) = k\ g(x/k)$

We can also derive the graph, function, and just-now versions of the same thing, but for simplicity, shl = $\lfloor x/2 \rfloor$. $x \div 2 = y \ldots r$, $x \equiv y \pmod{2}$, and $\lfloor x/2 \rfloor = (x-r)/2$. (a downward shifting ■■ (dip ■■ (dip ■))

Taking advantage of the information loss (?) of some bits in the shifting, we can write

$idx_\beta(x) = shr^B - shl_\beta^n(x)$

-.

Summarize later (EZ)

-.

def) bitwise $f_\beta (\Sigma A_{1,i}\ 2^{i-1}, \ldots(\times n)\ldots, \Sigma A_{□,i}\ 2^{i-1}) \equiv \Sigma f(A_{1,i}, \ldots(\times n)\ldots, A_{□,i})\ 2^{i-1}$ (i = 1 ~ B)

-.

nagation = bitwise-no, $nagation_\beta(\Sigma A_i\ 2^{i-1} =(= x)) = \Sigma (1-A_i)\ 2^{i-1} = \Sigma\ 2^{i-1} - \Sigma A_i\ 2^{i-1} = (2^B-1) - (\Sigma A_i\ 2^{i-1} =(= x))$, $nagation_\beta(x) = (2^B-1) - x$ (i = 1 ~ B)

-.

p(∘, *) := (a ∘ (b \* c) = a ∘ b \* a ∘ c where (∘, \*))

p(×, +) is true, but p(+, ×) is not, so that for a polynomial f, we usually get $f(\Sigma A_{1,i}\ 2^{i-1}, \ldots(\times n)\ldots, \Sigma A_{□,i}\ 2^{i-1})\ 2^{i-1})$ for a polynomial f. Non-negative logical operations are not interchangeable.

Therefore, unlike the negation operation, which is interchangeable by negating for subtraction, it is difficult for me to prove the other hexa(n) by applying the bitwise commutativity law

ㅋㅋ

Kingchiman says there is a bitwise indexing seat.

bitwise $f_{\beta}(x, y) = \Sigma f(idx_{\beta}(B - i, x), idx_{\beta}(B - i, y))2^{i-1}$ (i = 1 - B)

All in one second with math lol

Note: $f(x) = x - \lfloor x \rfloor$ is a type of sawtooth wave, $actan(tan(x/\pi))$ is also a type of sawtooth wave, and Gaussian function is just a ■a self-explanatory arithmetic operation that even a grade-schooler can understand.

-.

def) $(\exists!\, boolf \in \{x \mid Set(x)\}2\{f:U\to\{0,1\}\}\ \exists!\, boolf^{-1})(boolf^{-1}(p) \equiv \{x \mid p(x)\})$

def) $setize \equiv boolf^{-1}$

Bullshit; bullshit; meaninglessness is a machine or calculator truth table formula so 5

A proposition p that works on a Turing machine that writes Laftf 1.1 (this) is a set of propositions that are

can be a proposition about arithmetic, logic, or bitwise operations, and setize(p) is a set, which in turn can be a set $S = \{a_1, ..., a_n\}$ at some time $x = a_1 \vee ... \vee x = a_n$, so that every set S is reducible to p as a boolf(S), so that sets are restricted to the collection of all sets in the sense of the word.

In effect, a proposition = (= equation), so we are done, just write ■a.

That's all for LAFTF 1.1.

instagram @leenuxmathno7e

[github.com/FarAway6834](https://www.google.com/url?q=http://github.com/FarAway6834&sa=D&source=editors&ust=1737545183319986&usg=AOvVaw0udQDDfIxTu4hUrjAnk67S)

/c0dk1ddy

\It's a modular-2 ring that uses ANF (arithmetic normal form), Zhegalkin's formula, and bitwise operations applied to it as functions of truth-assigned operators rather than modular-2 rings, so it's not internally organized as modular-2, but I think LAFTF is a good abstraction to use on a computer.

I don't know if it's true that some titanium alloy on YouTube can solve math problems by thinking of them on its own, but I hope it is.

part 4. programming fw

unbeauty; something similar is already done with information (using sequences, inductive definitions: divisors, etc.), so **this is the sequence part (the core argument of the presentation**)

unbeauty esolang (0^0=1, x->0 x^x->1)

mathmatics.... beautiful 4 me....

`sementically-recursive-defined-recurrence-formular based` programming language

actually, I didn't finished this work, because of my high school exam.

~~[redacted]ing Korea~~

example : (relies on unbeauty + selection structure formula)

~~~
noptlib.unbe

```unbeauty

sqrt = cacher[1](λx. x^0.5)

abs = cacher[1](λx. this.sqrt(x^2))

partial_beq0c = cacher[1](λx, n, p. ((0^this.abs(n)) + n×(x + ((-1)^p)×n) ÷ (0^this.abs(n) + n^2)) × f(x, n - 1, p))

partial_beq0 = cacer(λb, x, p. this.partial_beq0(x, 2^b - p, p))

beq0 = cacher[1](λb, x. this.partial_beq0(b, x, 0) × this.partial_beq0(b, x, 1)

beq = cacher[1](λb, x, y. this.beq0(b, this.abs(x - y)))

dose_it_positive = cacher[1](λb, x. this.beq0(b, this.abs(b, x - this.abs(x))))

__cmp__ = cacher[1](λb, x. this.beq(b, x) + (-1)^(this.dose_it_positive(b, x)+1))

__le__ = cacher[1](λb, x, y. this.dose_it_positive(b, x - y))

conditionalidx = cacher[1](λp,x,y.p×(x-y)+y)

conditional = cacher[1](λp, x, y. p×x + (1-p)×y)

_4bit_eqer_ = cacher[1](λx,y.this.beq(4,x,y))

_4bit_idxer = cacher[1](λi,a0,a1,a2,a3,a4,a5,a6,a8,a9,aA,aB,aC,aD,aE,aF.this._4bit_eqer_(i,0)×a0+this._4bit_eqer_(i,1)×a1+this._4bit_eqer_(i,2)×a2+this._4bit_eqer_(i,3)×a3+this._4bit_eqer_(i,4)×a4+this._4bit_eqer_(i,5)×a5+this._4bit_eqer_(i,6)×a6+this._4bit_eqer_(i,7)×a7+this._4bit_eqer_(i,8)×a8+this._4bit_eqer_(i,9)
×a9+this._4bit_eqer_(i,10)×aA+this._4bit_eqer_(i,11)×aB+this._4bit_eqer_(i,12)×aC+this._4bit_eqer_(i,13)×aD+this._4bit_eqer_(i,14)×aE+this._4bit_eqer_(i,15)×aF)

Truncation
~~~

Splitting and concatenating

```
not = cacher[1](λx.1-x)

and = cacher[1](λx,y.x×y)

sor = cacher[1](λs,x,y.x+y-(1+s)*this.and(x, y))

or = cacher[1](λx,y.this.sor(0,x,y))

xor = cacher[1](λx,y.this.sor(1,x,y))

nand = cacher[1](λx,y.this.not(this.and(x,y)))

nor = cacher[1](λx,y.this.not(this.or(x,y)))

nxor = cacher[1](λx,y.this.not(this.xor(x,y)))

sub = cacher[1](λx,y.this.and(x,this.not(y)))

rsub = cacher[1](λx,y.this.sub(y,x))

rimp = cacher[1](λx,y.this.not(this.rsub(x,y)))

imp = cacher[1](λx,y.this.not(this.sub(x,y)))

a = cacher[1](λx,y.x)

b = cacher[1](λx,y.y)

nota = cacher[1](λx,y.this.not(x))

notb = cacher[1](λx,y.this.not(y))

logicalerr = cacher[1](λx,y.0)

alwaysstruth = cacher[1](λx,y.1) no nor
```

Ha,,,

```
lpu = cacher[1](λcod,x,y.this._4bit_idxer(cod,this.logicalerr(x,y), this.and(x,y), this.sub(x,y), this.b(x,y), this.rsub(x,y), this.a(x,y), this.xor(x,y), this.or(x,y), this.nor(x,y), this.nxor(x,y), this.nota(x,y), this.rimp(x,y), this.notb(x,y), this.imp(x,y), this.nand(x,y), this.alwaystruth(x,y)))

__gt__ = cacher[1](λb, x, y. this.not(this.__le__(b, x, y)))

__lt__ = cacher[1](λb, x, y. this.__gt__(b, y, x))

__ge__ = cacher[1](λb, x, y. this.__le__(b, y, x))

__ne__ = cacher[1](λb, x, y. this.not(this.beq(b, x, y)))

bits2bool = cacher[1](λb, x. this.not(this.beq0(b,,x)))

shr = cacher[1](λx, n.x÷(2^n))

shl = cacher[1](λb, x, n.(2^b)×x-(2^b)×(x÷(2^(b-n))))

idx = cacher[1](λb, x,i.this.shr(this.shl(b, x, i), b))

_bpucc_ = cacher[1](λb, cod,i,x,y.this.lpu(cod, this.idx)(b, x, i), this.idx)(b, y, i)) × (2^i))

_bpuc_ = cacher[1](λb, cod,i,x,y.this._bpucc_(b, i,x,y) + this.bits2bool(i)×this._bpuc_(b,i-1,x,y))

bpu = cacher[1](λb,cod,x,y.this._bpuc_(b,cod,b-1,x,y))
```

ex2.ubt - extend not base cls, ex1 cls.

```
:noptlib

fibo = cacher[1](λb, x.this.conditional(this.beq0(b, x), 0, this.conditionalidx(this.beq(b, x, 1), 1, this.fibo(b, x - 1) + this.fibo(b, x - 2))))
```

Note

## [coding plan 24.02.24 ~ ...](./plan)

### compile time lazy evil optimizing

`[○×]this.__NAME__(□)` -> `([○×]this.__NAME__)(□)`

 > also as `conditional` too. (when it is arguemnt, not return)

## FUOIR Unbeauty Optimize IR

example.unbe

```unbeauty
temp = cacher[0](λx. 2 × x)

main = cacher[0](λx. this.temp(x) + 1)
```

Uh-oh

example.auty (jsonic)

```fuior
[
    0,
    "x",
    "this.temp(x) + 1",
    {
        "temp" : [0, "x", "2 × x"],
    }
]
```

### plan change

not using macro, will use PCRE

#### symopt

just add optimizer at ir,

that sympy optimizer.

`this.□(○)` -> `base64(hash(□(○)))` to symbolize (like-lexing)

```
#LinearUnbeauty Plans (I'm an optimizer who can make anything, so don't worry about my coding skills)
```

```
# LinearUnbeauty scheme
```

```
[Unbeauty](https://faraway6834.github.io/unbeauty) with linear algebra support
```

```
  ... ends
```

LABARE language plan

An extension of [Linear Beauty](https://faraway6834.github.io/unbeauty/privateNote/Proof/LinearUnbeauty), with support for lambda arithmetic.

Specific implementation will be deferred,

It seems to favor productivity over performance, but in practice I'll probably go for performance.

This is the language for type proofs.

There is a kind of operation called Subrootine Realizationship that handles matrix inputs/returns as uncircularized inputs/outputs, but it only mimics it and doesn't actually do anything.

It is only used in Proofmood mode

## Name [derived from](https://genius.com/Edna-st-vincent-millay-euclid-alone-has-looked-on-beauty-bare-annotated)

Hmm...

The unbare language plan

A language for proof of type.

It is intended to spit out [labare](https://faraway6834.github.io/unbeauty/privateNote/Proof/labare) code, not to be executed.

The structure of the output laber code :

- virtual pararel scadular
- calculatable part
    - pararel scadular
    - haltible part
    - outof haltible part
- uncalculable part

In other words, the goal is to create mathematical Subrootine-Relationship code.

PART 5. So What

The difficulty level of "Alkali Mathmatics with easy explanation by SIOT, CMD, FAN and LAFTF1.1" is considered easy

The reason why I think Alkalic is easy is simple.

Let's assume that Alkalic is an equation ■ or someone who is good at solving it. If you teach it to a student who has completed the Olympiad of Integer Theory, after preaching him what a tautology is, he will thank you for changing logic to algebra.

But the truth is, the characteristic of people who have done the Number Theory Olympiad is that they can solve equations and functions ■ easily, which means that they can solve equation problems that look like f(x) = g(x) or h(x) = const. By giving an equation problem that looks like this, we can express the equation as a statement about the computation of the operation (the two computations are equal, or some computation is constant), which shows that the equation is essentially a statement about a function (operation), so we can explain the empty arguments in this FAN at a middle school level that can be explained well by an elementary school student because it teaches them to simply do the computation.

There you have it, easy.

Goal

- To someday write a book called "Alkalic Geogebra Science and Formal Explanation of IT", in which math, science, formal science, natural science, and social science are all described in alkaline terms, and then the math is explained in formal text again.
- To break down the rationalization of arrogant self-explanation of concepts like size, and to make the concepts more humble and say, "I think this is how it is."

The end goal

- To make linguistic ideas comprehensible to people who are not linguistically gifted or have poor interpretation skills.
- Use methods that can be learned by humans with less basic thinking talent rather than the traditional methods that take talent for granted.
- Avoid unfounded disdain; disdain for the untalented; and unfounded and irrational exclusion of the bi-literate.

PART 6. Plausibility

Structure and clarity

PART1. Reflections on numbers

Part 2. Proposing a math system that utilizes underlying function argument-based operations (equation-like; minds are descriptions of operations)

part 3. the rationale that equations are easy

part 4. programming fw

part 5. so what

The attitudes are practically all summarized in part 5.

Number 3 is Sam, a rebuttal to the silly question of whether number 2 is more difficult than traditional math.

He reflects on numbers, breaking down the authority of language and numbers.

Then, he proposes a more acceptable and understandable math, a humble, reformulated math that makes sense to the learner, that makes sense to the discipline, that makes sense to the instrumental, linguistic math. (Kantor is not humble.)

And finally, I propose a mathematical programming language and a way to learn it, and conclude by saying that UnBeauty, which is actually called Edgar, is also a tool language, and Edgar is even block coding.

PART 7. Why You're Not Leftist feat. Righty Lefty

The Right Aspirations Left manifesto is a bit political, so filter it out, and there's an open call for moderate right-leaning submissions.

Right-Longing Leftism

Introduction.

I think it is somewhat extreme for the right to resort to somewhat leftist means when they cannot achieve rightist goals by rightist means.

However, even if extreme, the end goal of the right is a rightist goal.

Therefore, achieving rightist goals is what rightist behavior should be about.

## The main points

1. What is the right-wing goal? The right-wing goal is a conservative steady state.

We want a conservatively normalized society because that is our ideal.

However, the conservative steady state will inevitably be broken at some point because the new image is absurd.

When the inevitable de-normalizing conservative cataclysm comes, we, as conservatives, will have to stop it.

This is the state of the Great Catastrophe.

However, if, in order to prevent the Great Catastrophe in the first place, we aim for peace at all times, and, as a practical defense, try to prevent the Great Catastrophe and the breakdown of peace, I see it as an ideal society.

In this ideal, we need to **normalize peacetime**.

**Peacetime normalization is the answer.

-Conservatives

2. to achieve peacetime normalization, conservative and reform conservatives must coexist. Normalization should not be moderate, or normalization should not be only reforms, and not coexistence creates a breakdown of peace. But what is wrong with reforming in a way that is moderate and achieves peacetime normalization, so that reform becomes a reason to do it even if it risks breaking peacetime normalization?

When the purpose is to repeat reforms or extreme right-wing actions, it takes away from the conservative color of peace. How left-wing is that, yes, that's too much, when the goal is to repeat reforms or far-right behavior, it's not conservative, it's left-wing, it's catastrophic, and we have to prevent catastrophe!!!

3. The insidious far left must always be watched. They seek the ideal steady state, the leftist rationale. Their killing and actions in pursuit of the ideal steady state, their right-wing means for pragmatic practice, and their aggressive movements must be watched closely, and their actions must be prevented/responded to at all times to prevent the cataclysm they are trying to create. The way to counter is to build a dirty plague, like a vaccine after identifying it. The first step is to read the book. The next step is to read their ideas, first believe as a lamb, then be a lion, then rebel, then be a child, then find a compromise. Fourth, examine what you read to see if it suits your conservative purposes, and fifth, refute them, anticipate them, prevent them, and preserve the peace. That's the basic idea that allows anyone to predict the insidious leftists.

If that doesn't work, you have to find another way, but I don't know if they're that insidious yet, so I'll give them the benefit of the doubt. (For example, studying and dealing with communist books to build and destroy their dirty ideas from scratch. This is the role of the conservative intellectual elite, the human intellect.)

Peace is best, no matter the means.

"Therefore".

Consider the left wing, where the ideal new world is a conservative steady state,

This is the Right-Desire Left.

It's dangerous;;

The right-craving leftist color scheme is a leftist whose goal is to defend the conservative norm, to hold the conservative norm and its radical variant, the reformless peacetime normalcy, as the ideal new world, to hold the reformless peacetime normalcy, as the ideal new world, and to act out of a longing for the right "right."

The Alternative Right is not enough!!!

We must call out for the right longing!!!!!

Let's filter what my crazy far-right friend says.
He hates politics.

Steady-state orientation

This is what this training scheme strives for

It's steady-state oriented and value-neutral, so it's not about liberals or conservatives.