千年の言語？
A Thousand-Year Language?

住井 英二郎
Eijiro Sumii

あんた誰？
Who am I?

- いわゆるプログラミング言語「理論」の研究者（大学准教授）です
  - ML（SML, OCaml）とかλ計算とか教えたり、
  - いわゆる「型理論」とかプログラム等価性とか研究しています
  - PerlとかRubyとか開発していません
    - すみません…
- Academic researcher on programming language "theory"
  - Not developing Perl or Ruby (sorry…)

何でここにいるの？
Why am I here?

- 趣味でMinCamlを作ったため（多分）
  - MinCamlって何？
    ⇒　OCamlサブセットの教育用コンパイラ
  - OCamlって何？
    ⇒　関数型言語MLの方言・拡張

- *Maybe because I developed "MinCaml", an educational compiler for a minimal subset of OCaml*

本題
On the agenda

- 百年後の言語はどうなっていると思いますか？

私の答：わかりません

なんじゃそりゃ

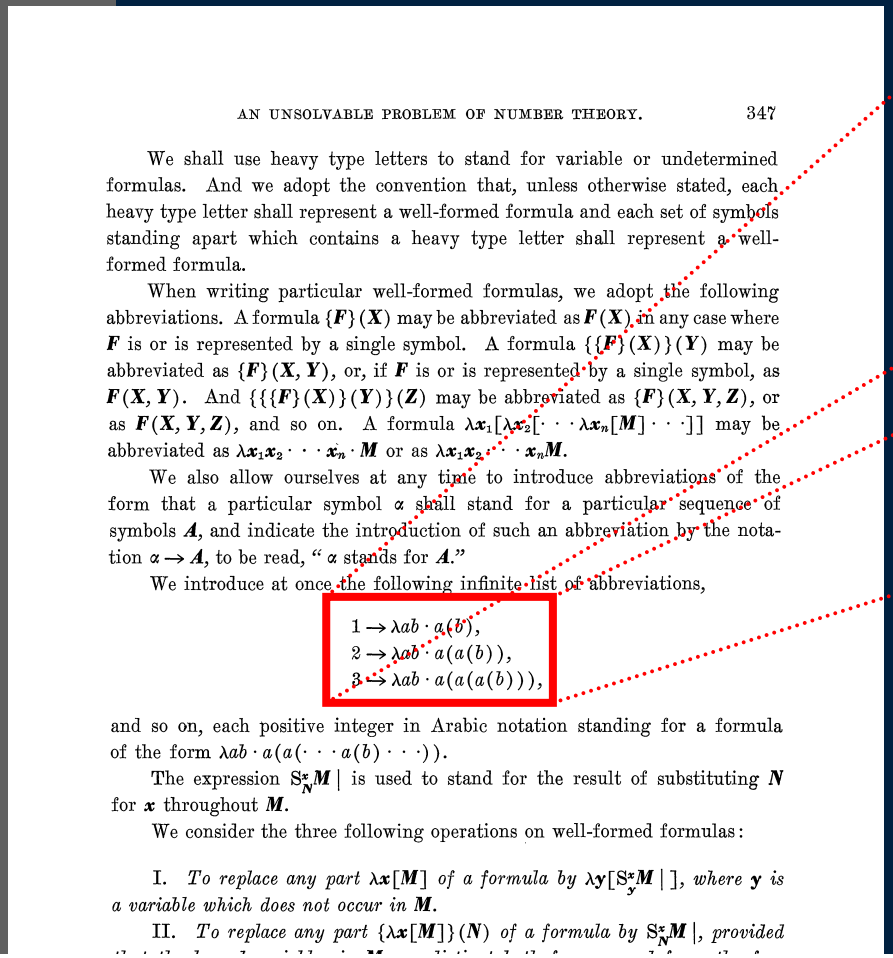
説明します

- What would be a programming language like 100 years later?

My answer: I don't know (I'll elaborate!)

約百年前
Almost 100 years ago

- Alonzo Church, "An Unsolvable Problem of Elementary Number Theory", American Journal of Mathematics, Vol. 58, No. 2 (Apr., 1936), pp. 345-363.



AN UNSOLVABLE PROBLEM OF NUMBER THEORY. 347

We shall use heavy type letters to stand for variable or undetermined formulas. And we adopt the convention that, unless otherwise stated, each heavy type letter shall represent a well-formed formula and each set of symbols standing apart which contains a heavy type letter shall represent a well-formed formula.

When writing particular well-formed formulas, we adopt the following abbreviations. A formula $\{\boldsymbol{F}\}(\boldsymbol{X})$ may be abbreviated as $\boldsymbol{F}(\boldsymbol{X})$, in any case where $\boldsymbol{F}$ is or is represented by a single symbol. A formula $\{\{\boldsymbol{F}\}(\boldsymbol{X})\}(\boldsymbol{Y})$ may be abbreviated as $\{\boldsymbol{F}\}(\boldsymbol{X}, \boldsymbol{Y})$, or, if $\boldsymbol{F}$ is or is represented by a single symbol, as $\boldsymbol{F}(\boldsymbol{X}, \boldsymbol{Y})$. And $\{\{\{\boldsymbol{F}\}(\boldsymbol{X})\}(\boldsymbol{Y})\}(\boldsymbol{Z})$ may be abbreviated as $\{\boldsymbol{F}\}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z})$, or as $\boldsymbol{F}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z})$, and so on. A formula $\lambda \boldsymbol{x}_1[\lambda \boldsymbol{x}_2[\cdots \lambda \boldsymbol{x}_n[\boldsymbol{M}]\cdots]]$ may be abbreviated as $\lambda \boldsymbol{x}_1 \boldsymbol{x}_2 \cdots \boldsymbol{x}_n \cdot \boldsymbol{M}$ or as $\lambda \boldsymbol{x}_1 \boldsymbol{x}_2 \cdots \boldsymbol{x}_n \boldsymbol{M}$.

We also allow ourselves at any time to introduce abbreviations of the form that a particular symbol α shall stand for a particular sequence of symbols $\boldsymbol{A}$, and indicate the introduction of such an abbreviation by the notation $\alpha \rightarrow \boldsymbol{A}$, to be read, "$\alpha$ stands for $\boldsymbol{A}$."

We introduce at once the following infinite list of abbreviations,

$$1 \rightarrow \lambda ab \cdot a(b),$$
$$2 \rightarrow \lambda ab \cdot a(a(b)),$$
$$3 \rightarrow \lambda ab \cdot a(a(a(b))),$$

and so on, each positive integer in Arabic notation standing for a formula of the form $\lambda ab \cdot a(a(\cdots a(b)\cdots))$.

The expression $\mathrm{S}^{\boldsymbol{x}}_{\boldsymbol{N}}\boldsymbol{M}\,|$ is used to stand for the result of substituting $\boldsymbol{N}$ for $\boldsymbol{x}$ throughout $\boldsymbol{M}$.

We consider the three following operations on well-formed formulas:

I. *To replace any part* $\lambda \boldsymbol{x}[\boldsymbol{M}]$ *of a formula by* $\lambda \boldsymbol{y}[\mathrm{S}^{\boldsymbol{x}}_{\boldsymbol{y}}\boldsymbol{M}\,|\,]$, *where* $\boldsymbol{y}$ *is a variable which does not occur in* $\boldsymbol{M}$.

II. *To replace any part* $\{\lambda \boldsymbol{x}[\boldsymbol{M}]\}(\boldsymbol{N})$ *of a formula by* $\mathrm{S}^{\boldsymbol{x}}_{\boldsymbol{N}}\boldsymbol{M}\,|$, *provided*

$$1 \rightarrow \lambda ab \cdot a(b),$$
$$2 \rightarrow \lambda ab \cdot a(a(b)),$$
$$3 \rightarrow \lambda ab \cdot a(a(a(b))),$$

λ計算：Lispや、いわゆる
「クロージャ」の元祖
(Scheme, JavaScript, Ruby, ...)

λ-calculus, the origin of Lisp
and so-called "closures"

ちょっと待て
Wait a sec

それ「プログラミング」言語じゃないでしょ？

- 汎用電子計算機が実現したのは1946年(ENIAC)

⇒「電子」計算機じゃなくてもプログラミングは可能（かつ必要）

- 17世紀から20世紀まで、“Computer” ＝「計算者」（人間の職業）だった

Was it even a “programming” language?

⇒ Yes! “Computers” need not be electronic.

- They were a human profession in the 17th-20th century

数百年前
A few hundred years ago

- 関数：
  17～18世紀（ライプニッツ、オイラー）
- 変数：
  古代ギリシャ、16世紀（ヴィエタ）

- Functions:
  17-18th century (Leibniz, Euler)
- Variables:
  Ancient Greece, 16th century (Vieta)

数千年前
A few thousand years ago

- ユークリッドの互除法：紀元前300年（エジプト）
  - 現在でもRSA暗号の鍵生成等で必須

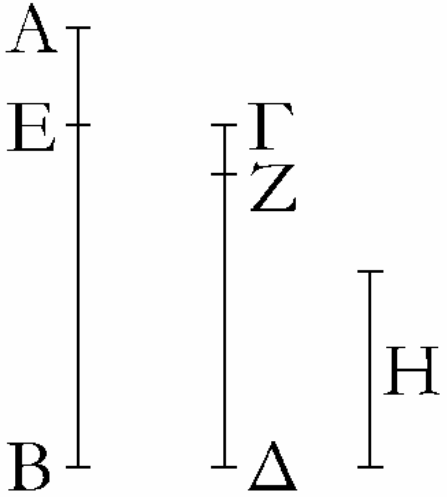
```
> grep euclid openssl-0.9.8h/crypto/bn/bn_gcd.c
static BIGNUM *euclid(BIGNUM *a, BIGNUM *b);
        t=euclid(a,b);
static BIGNUM *euclid(BIGNUM *a, BIGNUM *b)
```

- 二次方程式の一般的解法：紀元前1800年（バビロニア）
  - 当然必要（グラフィックスとかいろいろ）

- Important algorithms recorded before Christ
  - Euclid's algorithm (300 BC)
  - Solution of quadratic equations (1800 BC)

β΄.

Δύο ἀριθμῶν δοθέντων μὴ πρώτων πρὸς ἀλλήλους τὸ μέγιστον αὐτῶν κοινὸν μέτρον εὑρεῖν.



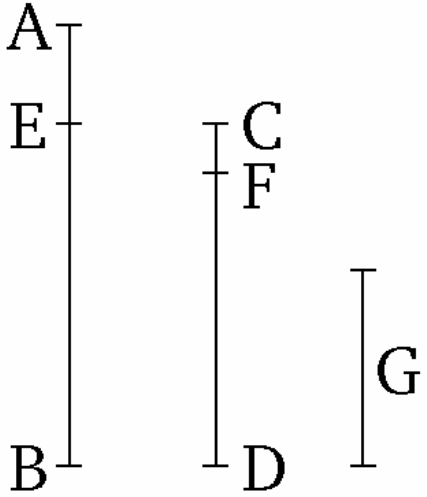
Ἔστωσαν οἱ δοθέντες δύο ἀριθμοὶ μὴ πρῶτοι πρὸς ἀλλήλους οἱ ΑΒ, ΓΔ. δεῖ δὴ τῶν ΑΒ, ΓΔ τὸ μέγιστον κοινὸν μέτρον εὑρεῖν.

Εἰ μὲν οὖν ὁ ΓΔ τὸν ΑΒ μετρεῖ, μετρεῖ δὲ καὶ ἑαυτόν, ὁ ΓΔ ἄρα τῶν ΓΔ, ΑΒ κοινὸν μέτρον ἐστίν. καὶ φανερόν, ὅτι καὶ μέγιστον· οὐδεὶς γὰρ μείζων τοῦ ΓΔ τὸν ΓΔ μετρήσει.

Εἰ δὲ οὐ μετρεῖ ὁ ΓΔ τὸν ΑΒ, τῶν ΑΒ, ΓΔ ἀνθυφαιρουμένου ἀεὶ τοῦ ἐλάσσονος ἀπὸ τοῦ μείζονος λειφθήσεταί τις ἀριθμός, ὃς μετρήσει τὸν πρὸ ἑαυτοῦ. μονὰς μὲν γὰρ οὐ λειφθήσεται· εἰ δὲ μή, ἔσονται οἱ ΑΒ, ΓΔ πρῶτοι πρὸς ἀλλήλους· ὅπερ οὐχ ὑπόκειται. λειφήσεταί τις ἄρα ἀριθμός, ὃς μετρήσει τὸν πρὸ ἑαυτοῦ. καὶ ὁ μὲν ΓΔ τὸν ΒΕ μετρῶν λειπέτω ἑαυτοῦ ἐλάσσονα τὸν ΕΑ, ὁ δὲ ΕΑ τὸν ΔΖ μετρῶν λειπέτω ἑαυτοῦ ἐλάσσονα τὸν ΖΓ, ὁ δὲ ΓΖ τὸν ΑΕ μετρείτω. ἐπεὶ οὖν ὁ ΓΖ τὸν ΑΕ μετρεῖ, ὁ δὲ ΑΕ τὸν ΔΖ μετρεῖ, καὶ

Proposition 2

To find the greatest common measure of two given numbers (which are) not prime to one another.



Let AB and CD be the two given numbers (which are) not prime to one another. So it is required to find the greatest common measure of AB and CD.

In fact, if CD measures AB, CD is thus a common measure of CD and AB, (since CD) also measures itself. And (it is) manifest that (it is) also the greatest (common measure). For nothing greater than CD can measure CD.

But if CD does not measure AB then some number will remain from AB and CD, the lesser being continually subtracted, in turn, from the greater, which will measure the (number) preceding it. For a unit will not be left. But if not, AB and CD will be prime to one another [Prop. 7.1]. The very opposite thing was assumed. Thus, some number will remain which will measure the (number) preceding it. And let CD measuring BE leave EA less than itself, and let EA measuring DF leave FC less

もっと最近
More Recently

- 関数型言語、ガベージコレクション：1950年代(Lisp)
- オブジェクト：1960年代(Simula 67)
- 例外処理、ジェネリックス、型推論：1970年代(ML)
- Important concepts developed in the 50-70's
  - FP, GC, objects, exception handling, generics, type inference, ...

だから何？
So what?

- 「100年後のプログラミング言語」は予想することも、意図して発明することもできない
- しかし、「面白い」と思ったアイディアをきちんと形に残しておけば、100年後や1000年後に開花するかもしれない

- We cannot predict nor invent the (far) future by intention
- However, our idea may bloom 100-1000 years later if we record it well

それでも予想してみると…
That said…

- つい最近まで、「コンピュータ ＝ 人間」「プログラミング言語 ＝ 自然言語＋α」だった
- 将来は、電子計算機を「自然言語＋α」でプログラムできるようになる…かも？？？

- *Until recently, "computers" were human beings and "programs" were written in (almost) human languages*
- *Why not let electronic computers understand those languages?*

それ以外のデタラメ予想
Other random predictions

- 全自動正当性証明器つき言語？
- 大量の目的特化言語(DSL)？
- 非技術者用グラフィカル言語？
- 量子言語？
- Language with fully automatic correctness prover?
- Millions of domain specific languages?
- Graphical languages for your mom and children?
- Quantum languages?

あなたは今まで何をしてきましたか？
What have I done so far?

- 「プログラミング言語理論」の基礎研究
  （≠CやJavaの使い方）
  - λ計算（関数型言語のモデル）とか
    π計算（並行計算のモデル）とか

  Cf.
  - 「コンピュータサイエンスって、WordやExcelの使い方を研究してるんですか」（医者の知人）
  - 「テレビ壊れたから直して」（某先生の御父上）

- Foundational research on programming language theory (λ-calculus, π-calculus, ...)

なぜプログラミング言語理論？
Why PL theory?

- 「計算」の本質に迫る（と思う）から
  - プログラミング言語なんて、文法（見た目）以外は全部チューリング等価じゃないの？
    ⇒　否！「抽象化」が本質的
    （例：アセンブリ言語 v.s. Ruby）
  - 電子計算機でも人間コンピュータでも同じ
- *It is about the essence of computing!*
  - *Doesn't matter whether the computer is electronic or human*

で、お前は何を研究してるの？
My current research topic

- いろいろな抽象化機能がもたらす性質について研究しています
  - 例：ハッシュで実装した辞書と、木で実装した辞書
    - バイト列としては等価にならない
    - オブジェクトとしてカプセル化すれば等価になる
  - 同様に、様々な抽象化機能が様々な性質をもたらす（モジュール、ローカル変数、クロージャ、etc.）
  - そのような性質を証明する手法を研究しています
- A proof method for various properties induced by abstraction mechanisms
  - Objects, modules, local variables, closures, ...

Definition 3.1 (*Bisimulation*). A *bisimulation* is a set X of pairs $(\Delta, \mathcal{R})$ such that:

(1) $\Delta \vdash \mathcal{R}$.

(2) For each

$$(\texttt{fix}\ f(x:\pi):\rho = M,\ \texttt{fix}\ f(x:\pi'):\rho' = M',\ \tau \to \sigma) \in \mathcal{R}$$

and for any $(V, V', \tau) \in (\Delta, \mathcal{R})^\circ$, we have

$$\begin{aligned} &(\texttt{fix}\ f(x:\pi):\rho = M)V \Downarrow \\ \iff\ &(\texttt{fix}\ f(x:\pi'):\rho' = M')V' \Downarrow. \end{aligned}$$

Furthermore, if $(\texttt{fix}\ f(x:\pi):\rho = M)V \Downarrow W$ and $(\texttt{fix}\ f(x:\pi'):\rho' = M')V' \Downarrow W'$, then

$$(\Delta, \mathcal{R} \cup \{(W, W', \sigma)\}) \in X.$$

(3) Let $\Delta = \{(\alpha_1, \sigma_1, \sigma_1'), \ldots, (\alpha_m, \sigma_m, \sigma_m')\}$. For each

$$(\Lambda\alpha.\, M, \Lambda\alpha.\, M', \forall\alpha.\, \tau) \in \mathcal{R}$$

and for any ρ with $FTV(\rho) \subseteq dom(\Delta)$, we have

$$(\Lambda\alpha.\, M)[[\overline{\sigma}/\overline{\alpha}]\rho] \Downarrow \iff (\Lambda\alpha.\, M')[[\overline{\sigma'}/\overline{\alpha}]\rho] \Downarrow.$$

Furthermore, if $(\Lambda\alpha.\, M)[[\overline{\sigma}/\overline{\alpha}]\rho] \Downarrow W$ and $(\Lambda\alpha.\, M')[[\overline{\sigma'}/\overline{\alpha}]\rho] \Downarrow W'$, then

どこがプログラムやねん
It doesn't look like a program!

- 特定のプログラミング言語ではなく、数学的計算モデルとして定式化
  - 一般的本質を抽出するため

- It is a general, mathematical model
  - Rather than a particular, concrete programming language

他には？
What else?

- 文字列のための正規表現型
  - 変数や出力がどういう文字列になるか、プログラム実行前に解析する
- メモリ安全なC言語実装

等々に手を出したことも…（主に学生と）

- *Regular expression types for strings*
- *Memory-safe implementation of C*

etc., with students

MinCamlは？
What about MinCaml?

- あれは研究というより趣味です:-)
  - 関数型言語は難しくも遅くもない、という宣伝目的
    - 対象言語の構文が11行
    - コンパイラは2000行
    - まあまあ自明でないプログラムが動く
      - 千数百行のレイトレーシングなど
    - GCCとほぼ同じぐらいの性能（当時）
- A compiler in 2000 lines, producing as efficient native code as GCC

あなたはこれから何をしますか？
What am I going to do?

- すみません、わかりません
  （はじめに言った通り）
- 百年後・千年後の人も「面白い」と思えるような研究を「記録に残す」
  - ことができれば良いなあ…と思います
- Sorry, I don't know (as stated before)
- However, I'd like to invent and record ideas that are still interesting in the far future