人工知能と数学

はじめに

「人工知能と数学」セミナーの問題意識

人工知能の「言語能力」の画期的進歩

今日の「人工知能」は、その「言語能力」で、画期的な進歩を遂げています。

- ある言語の文を他の言語に「翻訳」すること。
- 言語を問わず膨大な文字データから「関連する」データを見つけ出すこと。
- 長い文章を短い文章に「要約」すること。
- ある話題に対して関連する話題を提供して「対話」を続けること。

「大規模言語モデル」の成長とその秘密

こうした能力は、「大規模言語モデル」と呼ばれる人工知能技術の登場とその成長によって初めて可能になりました。

その成長の秘密の鍵は、この技術が「ことばの意味」を、コンピュータ上で表現する方法を見つけたことにあります。

イメージの認識なら、イメージをピクセルの集まりとしてコンピュータ上で表現することは容易です。ただ、「ことばの意味」を、コンピュータの上で表現せよといわれたら、みなさんはどんな方法をイメージしますか？

「意味の近さ」を表現できる

Googleの検索は、膨大なネット上の文字データから、基本的には特定の「文字列」を探すものです。「意味」を「検索」しているわけではありません。

「大規模言語モデル」では、「意味」がコンピュータ上に表現されています。

この「意味の表現」では、「意味が同じ」ことだけでなく「意味が近い」ことが表現できます。「意味の近さ」が表現できるということは、実践的にはとても大事です。

「大規模言語モデル」が獲得した「意味の表現」の力

「翻訳」でしたら「意味が同じ」ことが重要です。話題の「関連性」を見つけるとか「要約」のケースでは「意味が近い」ことがポイントになります。

こうした判断をコンピュータが自由に行えるようになったことが、コンピュータの「言語能力」の飛躍をもたらしました。それらは、「大規模言語モデル」が獲得した「意味の表現」の力によるものです。

「大規模言語モデル」的アプローチでは数学の問題を解くことができない

こうした発展は素晴らしいものです。ただ、問題は、その先にあるのです。

「大規模言語モデル」的アプローチには、大きな弱点があります。それは、こうしたアプローチでは、現状では、数学の問題を解くことができないことです。

そのことは、2022年のOpenAIのTheorem ProverやDeep MindのAlpha Codeの失敗を見ればわかります。

規模拡大で問題が解決するか？

人間の論理＝数学的能力は、言語能力に基礎を置いています。ことばなしには、数学を考えることはできません。

それでは、「大規模言語モデル」の規模をさらに拡大すれば、数学の問題も解けるようになるのでしょうか？ それも難しいと思います。「大規模言語モデル」は、基本的には、人間の意味理解を中心とした言語能力のモデルだからです。

規模拡大では問題は解決しない

何よりも、OpenAIもGoogle Researchも、2022年の時点で、規模拡大による能力の向上には限界があることを認めています。

そのことは、大規模言語モデルをベースに数学の問題を解こうとしたOpenAIの”Theorem Prover”、プログラム生成にチャレンジした Googleの”Alpha Code”の二つのプロジェクトが失敗したことを見れば明らかです。

ChatGPTは、こうした大規模言語モデルの「行き詰まり」の中から、大胆な路線転換として生まれたものだと僕は考えています。

コンピュータと数学の関係

今回のセミナーで強調したいことは、大規模言語モデルと数学の問題としてではなく、人工知能と数学の問題として、あるいは、コンピュータと数学の関係として問題を考えれば、全く別のアプローチがあるということです。

それは、ディープラーニング的・大規模言語モデル的回り道をしなくても、コンピュータ自身に、直接に数学的推論を行う能力があるのだと考えることです。

コンピュータと数学の関係の認識は、数学の基本的性質についての我々の認識に依存しています。今回のセミナーで数学の基礎に多く触れたのは、そのためです。

人工知能と数学

Agenda

第一部
大規模言語モデルと数学

第二部　20世紀
数学の基礎と計算科学

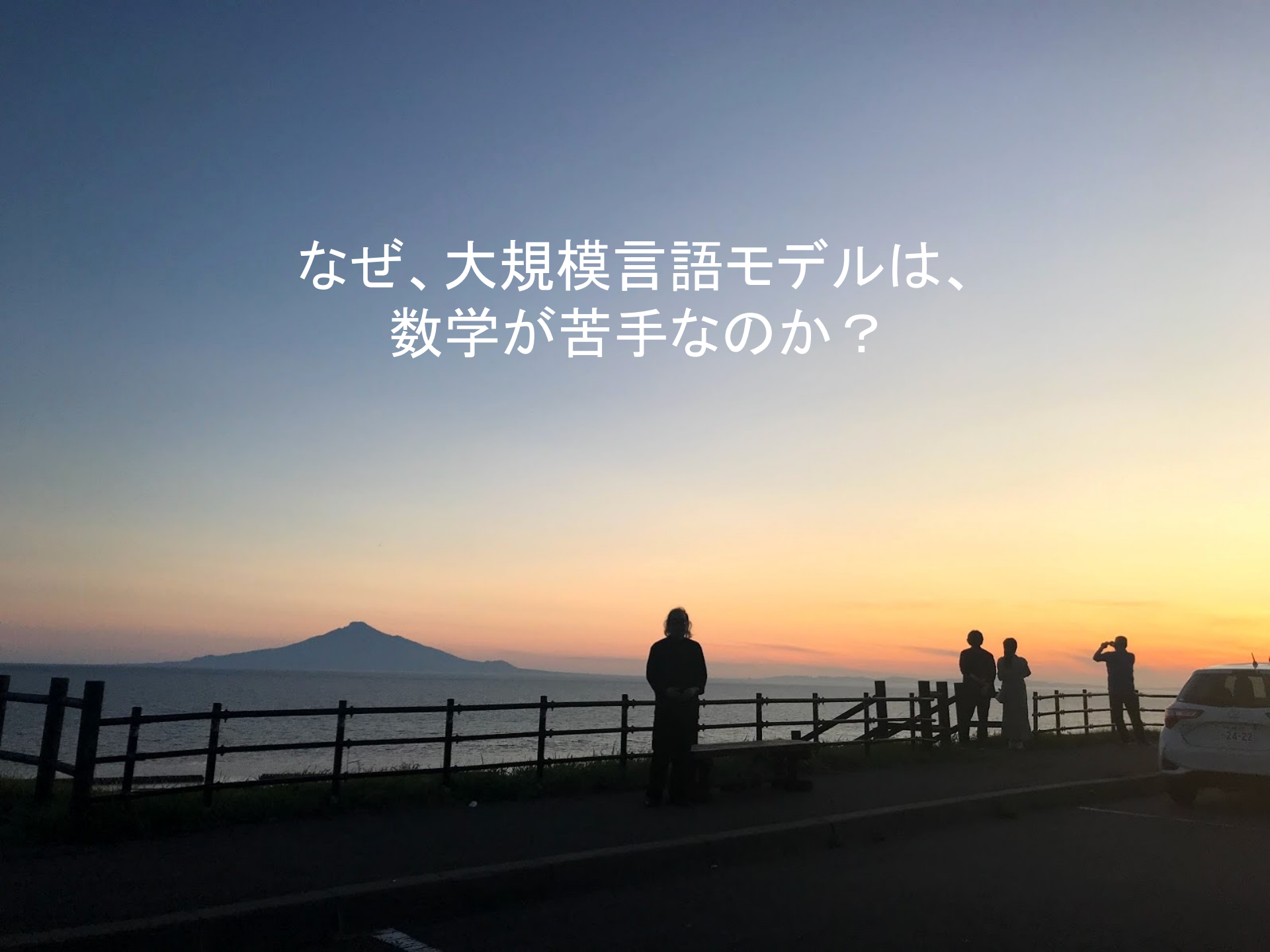
第三部　21世紀
数学の基礎と計算科学の新しい動向


人工知能と数学
第一部
大規模言語モデルと数学

第一部

大規模言語モデルと数学

- なぜ、大規模言語モデルは、数学が苦手なのか？
- 大規模言語モデルとパラレル・コーパスと言語学習
- 言語の習得と数学の学習
- ニューラル・ネットワークは モノマネの天才
- GitHubがChatGPTを救う


なぜ、大規模言語モデルは、
数学が苦手なのか？

なぜ、大規模言語モデルは、数学が苦手なのか？

なぜ、大規模言語モデルは、数学ができないのか？　それには理由があると思います。

大規模言語モデルは、機械翻訳モデルから派生したものです。それはある言語の文法的に正しい文を受理し、他の言語の文法的に正しい文に変換・生成する能力を持ちます。

そののの力は、それはそれで素晴らしいものです。

翻訳的意味理解

そこでの意味理解は、基本的には、翻訳的意味理解ともいうべきものです。それは、

「〝I love you〟の意味は、〝私はあなたを愛している〟ということである。」

と考えることです。

翻訳的意味理解

あるいは、同じことだと思いますが

「システムTが、″I love you″ を″私はあなたを愛している″ に変換できるなら、システムTは、″I love you″ の意味を理解している」

と考えることです。

変換の条件

翻訳システムTは、入力に与えられた語の並びS1を、別の語の並びS2に変換するシステムと考えることができます。

ただ、こうした変換が可能であるためには、システムTに課せられる最低限の条件があります。それは、S1もS2も「文法的」に正しい語の並びでなければならないということです。

" I you love"は英語の文法にあっていませんのでシステムTは、それを入力として受け付けることはできません。また "私愛するあなた”は、日本語の文法にあっていませんので、システムTは、そうした出力をすることはできません。

変換すると意味が与えられる？

翻訳システムTが、こうした入力と出力の文法性・構成性の要件を満たしているとして、次のように考えることができるでしょうか？

「システムTが、文法的に正しいシーケンスS1を文法的に正しいシーケンスS2に変換できる時、システムTはS1の意味を理解していると考えることができる。」

これは少しおかしいですね。こう言えるのは、出力S2の意味を我々が理解できる場合だけですね。

機械の意味理解を
人間の意味理解に還元する

少し変更しましょう。

「システムTが、文法的に正しいシーケンスS1を文法的に正しいシーケンスS2に変換でき、S2の意味を我々が理解できる時、システムTはS1の意味を理解していると考えることができる。」

これは、機械の意味理解についてのいい解釈かもしれません。大方の場合はそれでうまくいきそうです。
機械の意味理解を、人間がその意味を理解できることに還元しているのが気になりますが、そこは当面は目をつぶることにしましょう。

数学の問題を考える

ただ、こうした意味の理解だと、数学の問題の意味を考えると、そうそうに少し困ったことが起きます。

多分、それは人間の意味理解の能力が、非常に広いからだと思います。一つの逃げ道は、数学のことなど考えないことなのですが。

「１＋１＝２」を考える

「１＋１＝２」という文を考えましょう。

この構文的に正しい式をシステムTは、「1たす1は2である」と正しい日本語に変換できるでしょう。

我々はこの日本語の意味を理解できますので、Tは「１＋１＝２」の意味を理解していると考えることができます。

「１＋１＝３」を考える

今度は、「１＋１＝３」という文を考えましょう。

この構文的に正しい式をシステムTは、「1たす1は3である」と正しい日本語に変換できるでしょう。

ただ困ったことに、我々はこの日本語の意味を、そのまま理解できるのです。

ですから、先の解釈に従えば、Tは「１＋１＝３」の意味を理解していると考えるしかないのです。たとえ、我々には、「それは、数学的には間違っている」とわかっていたとしても。

数学的真と偽の区別ができない

別の言い方をすれば、変換システムT自体には、数学的には真である「１＋１＝２」と数学的には偽である「１＋１＝３」を区別することはできないのです。

「１＝１」と「１＝０」の区別のできないシステムに、数学的能力を期待することはできません。

もっと一般的に言えば、ある命題が現実の世界で真であるとか、物理的実在の世界で真であるとかの判断を機械に期待することはできません。ただ、そのことで機械の無能力さをあげつらうのは、僕の本意ではありません。それは、まず、なによりも、現実の中で生き、物理的実在に取り囲まれている人間に求められている仕事なのですから。

機械は、論理的・数学的推論を正しく行う能力を持っている

それでは、機械に数学的能力を期待するのは、難しいのでしょうか？　大規模言語モデルの上に数学的能力を構築するのが難しいからといって、まったくそうではないのです。

機械には、そうした回り道をせずとも、直接に、自分の力だけで、論理的・数学的に正しい推論を行う能力を持っているのです。

現在の「人工知能」ブームの中で、多くの人はそのことを見落としています。

大規模言語モデルと
パラレル・コーパスと言語学習

パラレル・コーパス

現代の機械翻訳に利用される パラレル・コーパスとその規模

- WMT'14の英語（En）<-> フランス語（Fr）データセットには、3,600万の文のペアが含まれている。
- WMT'14の英語(En) <-> ドイツ語（De）データセットには、500万の文のペアが含まれている。
- Googleは、内部に、英語 <-> 日本語（Ja）、英語 <-> 韓国語（Ko）、英語 <-> スペイン語（Es）、英語 <-> ポルトガル語（Pt） 等々の多くのデータセットを持っているが、その規模は、先のWMTのデータセットより、２～3桁大きいという。

- Googleニューラル機械翻訳では、GPU100個を使って、フルトレーニングには最大1,000万ステップ、収束までには3週間かかることがあるという。

コーパスの例（英仏）

English	French
According to our survey, 1988 sales of mineral water and soft drinks were much higher than in 1987, reflecting the growing popularity of these products. Cola drink manufacturers in particular achieved above-average growth rates. The higher turnover was largely due to an increase in the sales volume. Employment and investment levels also climbed. Following a two-year transitional period, the new Foodstuffs Ordinance for Mineral Water came into effect on April 1, 1988. Specifically, it contains more stringent requirements regarding quality consistency and purity guarantees.	Quant aux eaux minérales et aux limonades, elles rencontrent toujours plus d'adeptes. En effet, notre sondage fait ressortir des ventes nettement supérieures à celles de 1987, pour les boissons à base de cola notamment. La progression des chiffres d'affaires résulte en grande partie de l'accroissement du volume des ventes. L'emploi et les investissements ont également augmenté. La nouvelle ordonnance fédérale sur les denrées alimentaires concernant entre autres les eaux minérales, entrée en vigueur le 1er avril 1988 après une période transitoire de deux ans, exige surtout une plus grande constance dans la qualité et une garantie de la pureté.

Gale & Church "A Program for Aligning Sentences in Bilingual Corpora" http://www.aclweb.org/anthology/J93-1004.pdfから。

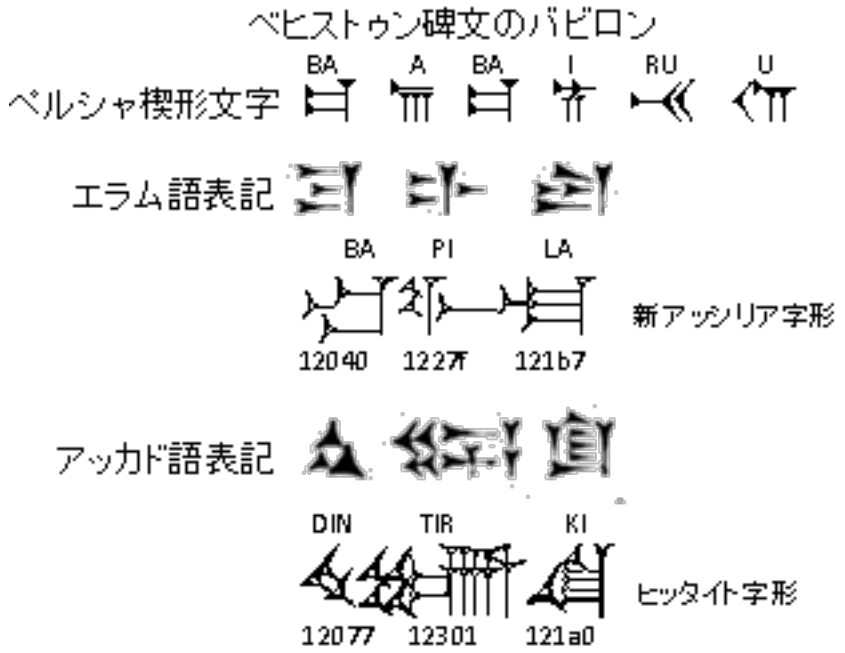
コーパスの例（英仏）

English	French
According to our survey, 1988 sales of mineral water and soft drinks were much higher than in 1987, reflecting the growing popularity of these products. Cola drink manufacturers in particular achieved above-average growth rates.	Quant aux eaux minérales et aux limonades, elles rencontrent toujours plus d'adeptes. En effet, notre sondage fait ressortir des ventes nettement supérieures à celles de 1987, pour les boissons à base de cola notamment.
The higher turnover was largely due to an increase in the sales volume.	La progression des chiffres d'affaires résulte en grande partie de l'accroissement du volume des ventes.
Employment and investment levels also climbed.	L'emploi et les investissements ont également augmenté.
Following a two-year transitional period, the new Foodstuffs Ordinance for Mineral Water came into effect on April 1, 1988. Specifically, it contains more stringent requirements regarding quality consistency and purity guarantees.	La nouvelle ordonnance fédérale sur les denrées alimentaires concernant entre autres les eaux minérales, entrée en vigueur le 1er avril 1988 après une période transitoire de deux ans, exige surtout une plus grande constance dans la qualité et une garantie de la pureté.

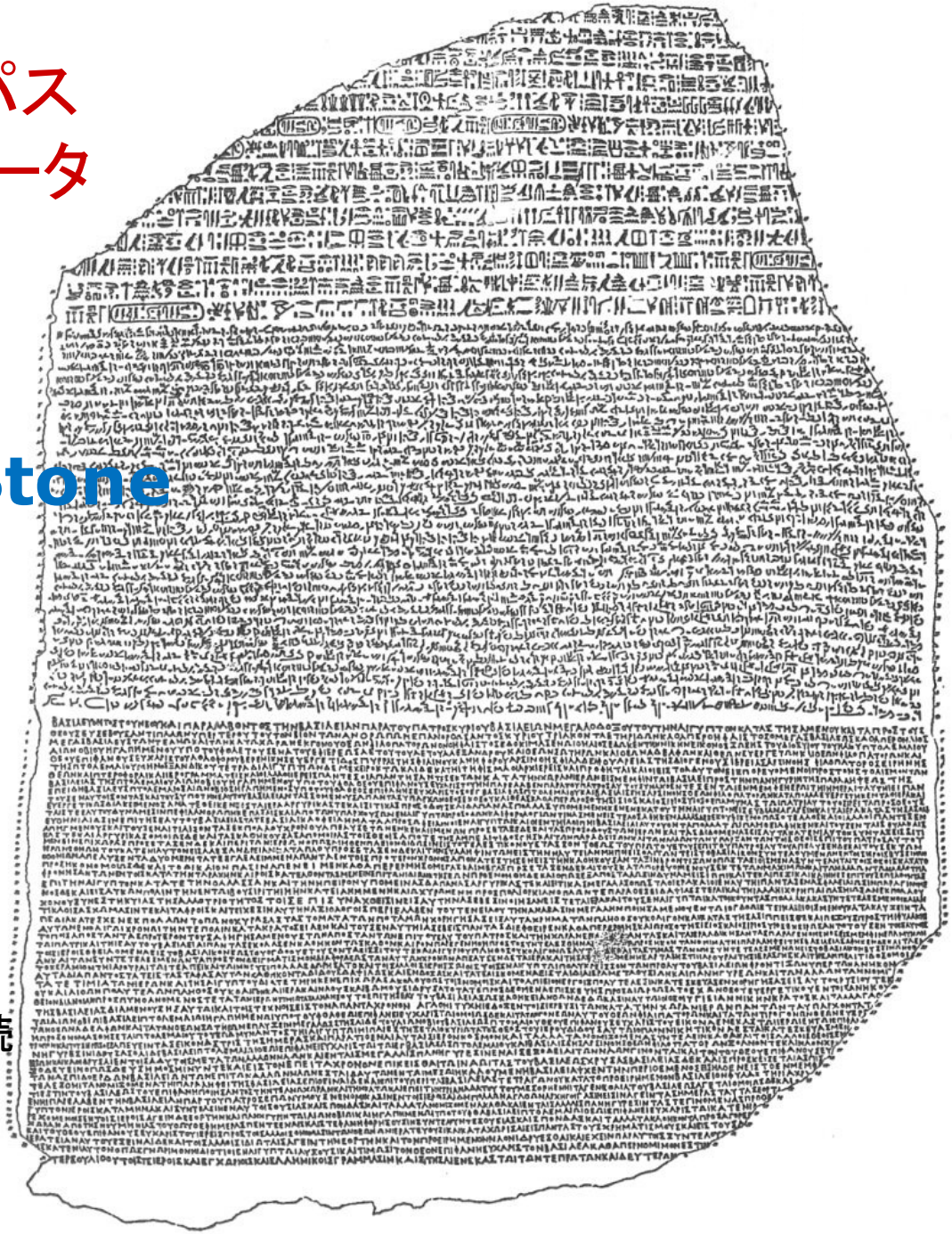
Gale & Church "A Program for Aligning Sentences in Bilingual Corpora"
http://www.aclweb.org/anthology/J93-1004.pdfから。

最古のパラレル・データ
ベヒストゥン碑文　楔形文字の解読

コーパス：　ベヒストゥン碑文　BC522~
（エラム語、古代ペルシア語、アッカド語の三つの言語で書かれている）
解読者：ローリンソンとヒンクス
1846-1851



最古のコーパス
パラレル・データ



Rosetta Stone
BC 196年

神聖文字

民衆文字

ギリシャ
文字

1822年
シャンポリオンが解読

パラレル･コーパスを利用した言語学習
日本の経験 – 白文の素読

白文（漢文）

子曰
学而時習之 不亦説乎
有朋自遠方来 不亦楽乎
人不知而不慍 不亦君子乎

有子曰
其爲人也孝弟 而好犯上者鮮矣
不好犯上 而好作亂者未之有也
君子務本 本立而道生
孝弟也者其爲仁之本歟

書き下し文（日本語）

子曰く、

学びて時に之を習ふ。亦説(よろこ)ばしからずや。

朋有り、遠方より来たる。亦楽しからずや。

人知らずして慍(うら)みず、亦君子ならずや。と。

有子曰く、其の人と爲りや孝弟にして、上を犯すことを好む者は鮮(すくな)し。

上を犯すを好まずして、亂を作すことを好む者ものは、未だ之有らざるなり。君子は本を務む、本立つて道生ず、孝弟は、其れ仁の本たるか。

素読（パラレル・コーパス）

子曰。
子曰く、

学而時習之。不亦説乎。
学びて時に之を習ふ。亦説(よろこ)ばしからずや。

有朋自遠方来。不亦楽乎。
朋有り、遠方より来たる。亦楽しからずや。

人不知而不慍。不亦君子乎。
人知らずして慍(うら)みず、亦君子ならずや。と。

素読（パラレル・コーパス）

有子曰、其爲人也孝弟、而好犯上者、鮮矣。

有子曰く、其の人と爲りや孝弟にして、上を犯すことを好む者は鮮(すくな)し。

不好犯上、而好作亂者、未之有也。不好犯上、而好作亂者、未之有也。

上を犯すを好まずして、亂を作すことを好む者ものは、未だ之有らざるなり。

君子務本、本立而道生。孝弟也者、其爲仁之本歟。

君子は本を務む、本立つて道生ず、孝弟は、其れ仁の本たるか。

訓点

登レ山ニ

学ニ我書ヲ一

士ハ不レ可カラ以テ不レ弘ナラ毅。

不下為ニ二児孫一ノ買ハ中美田上ヲ。

言語の習得と数学の学習

機械翻訳システムでの言語の学習

大規模言語モデルや機械翻訳システムでの言語の学習は、基本的には、膨大なパラレル・コーパスの学習である。パラレル・データの一方が模範解答として教師の役割を果たす。

それは人間なら嫌になって逃げ出したくなるほどの、教師による誤りの指摘と修正の、気の遠くなるような繰り返しである。幸いなことに、人間と違って機械は、登校拒否することも気絶することもない。

機械学習の基本的メカニズムの同一性と機械の特性

機械学習にはいくつかのタイプがあるのだが、画像認識でもSequence to Sequence でも強化学習でも、膨大なデータと繰り返しの学習によって「誤り」の低減を目指すというメカニズムは同じである。

機械はそうした退屈だが過酷な試練を耐えぬく、ある意味では人間（すくなくとも僕）に欠けている、優れた能力を持っているのである。素晴らしい！

ChatGPTの、時には嘘も交えて流暢に言葉を話す能力は、時にはヘラヘラしている印象を与えるかもしれないが、彼の生まれも育ちも「根性」も、試練に耐えた筋金入りのものだ。

母語習得のメカニズムは、機械による言語学習とは異なる

ただ、こうした機械の言語学習のモデルが、言語学習の一般的なモデルであるとは、僕は思っていない。僕らは、それとは全く違うスタイルで「母語」を操る能力を手に入れているからだ。

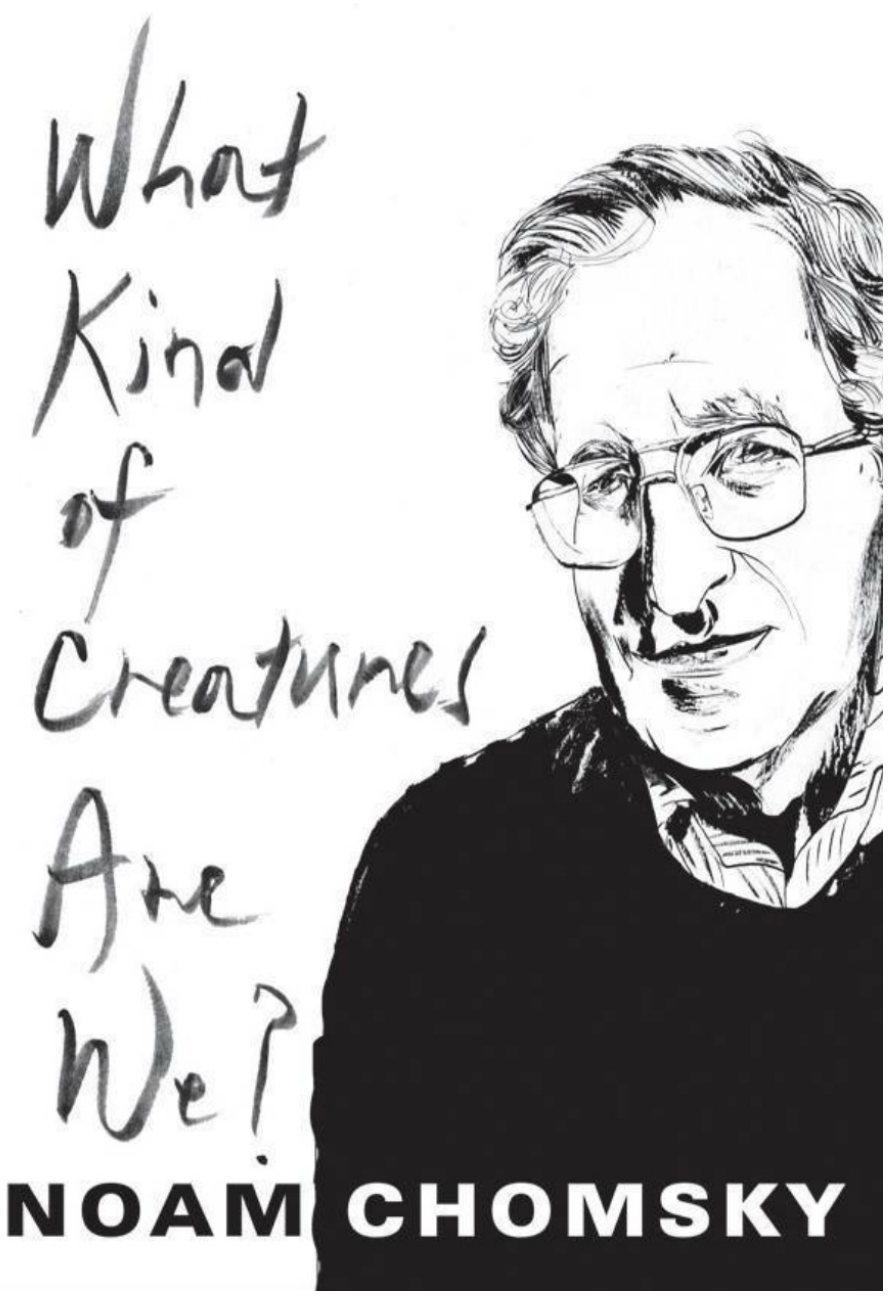
僕らは、けっして数億ペアの大規模パラレル・コーパスを与えられて、ことばを習得したわけではない。それは経験的にはあきらかだ。その上、論理的には、僕らが学ぶ最初の言葉である母語に、対応するペアなどあるわけがないのだ。また、「母」にあたる環境は、ことばの間違いをしつこく指摘する「鬼教師」ではなかったはずだ。

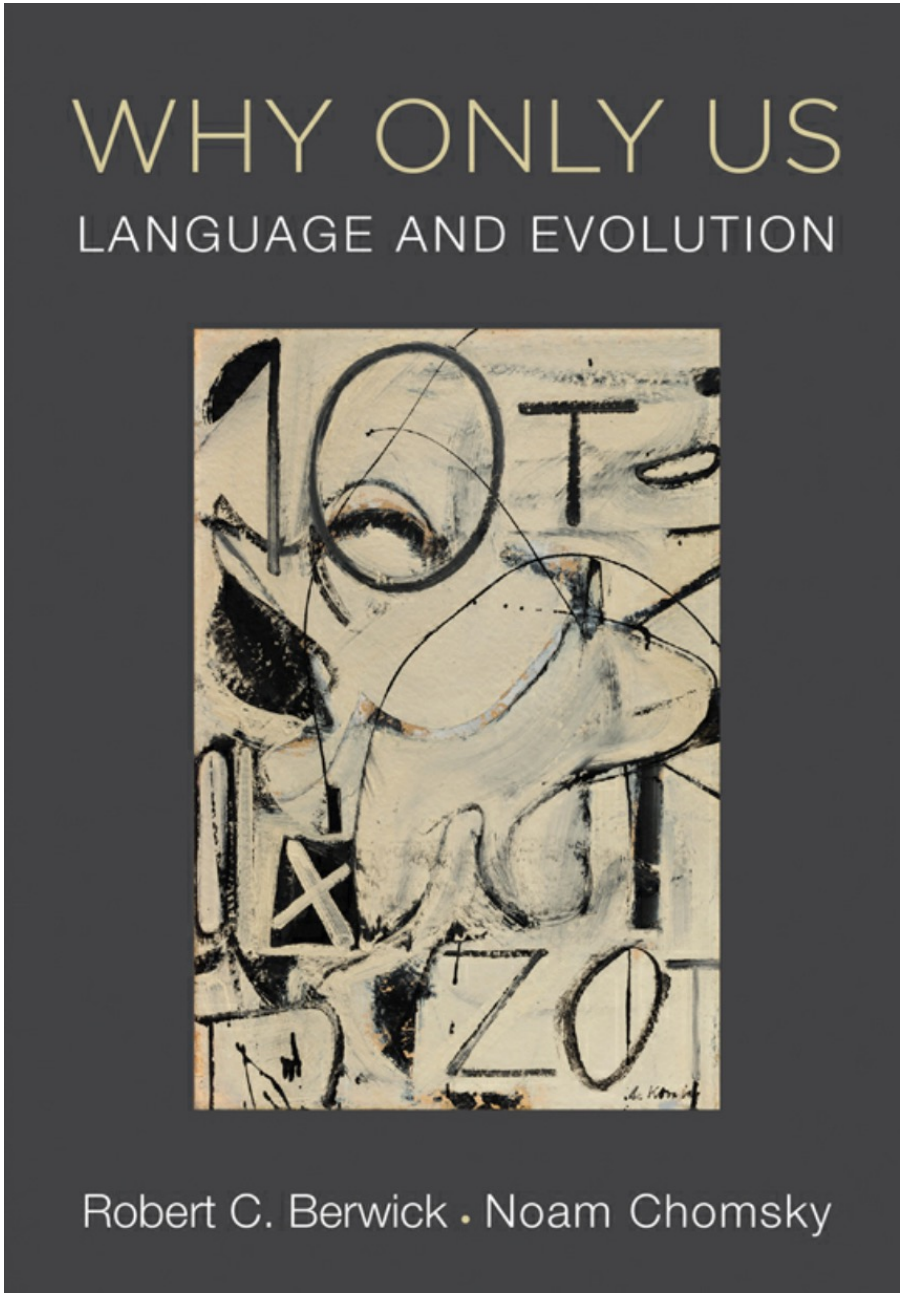
人間のもっとも基本的な能力としての言語能力

人間が、だれでもことばを理解できるというのは、言語能力が人間のもっとも基本的な能力であるということである。

機械が示し始めた言語能力に感心する前に、そういう時代だからこそ、僕ら自身の言語能力の習得の不思議さに、もっと関心が集まっていいと僕は思う。

言語学者は昔からそうした関心を持っていたはずだ。そこには、Chomsky「言語能力の生得性」の主張をはじめとしてたくさんの知見の集積がある。


What Kind of Creatures Are We?
NOAM CHOMSKY


WHY ONLY US
LANGUAGE AND EVOLUTION
Robert C. Berwick • Noam Chomsky

数学の学習プロセスの不思議

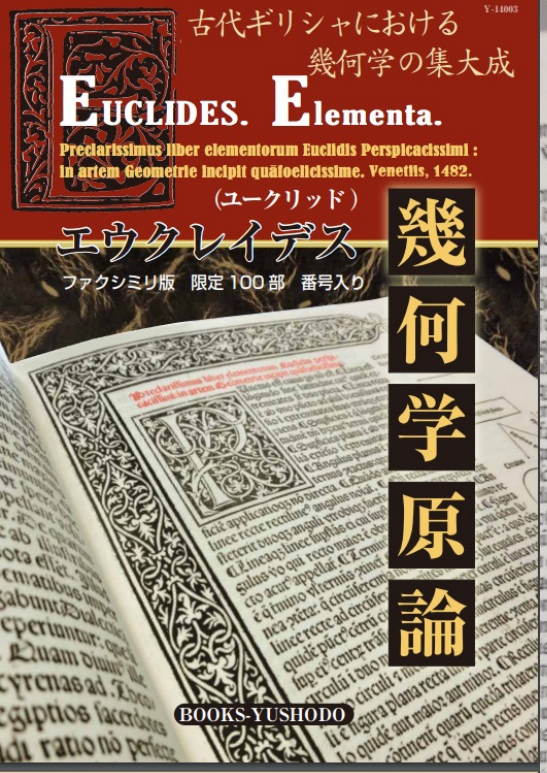
話は変わるのだが、大規模言語モデル的な学習モデルが、学習の普遍的なモデルではないことを示すのは、人間の言語習得のプロセスだけではない。

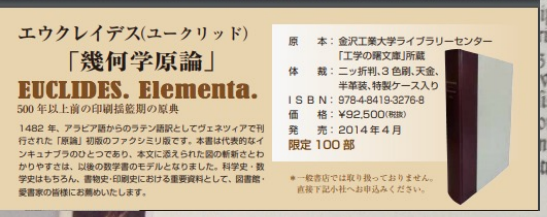
数学の学習プロセスも、現象的にも原理的にも、多くの不思議に包まれている。

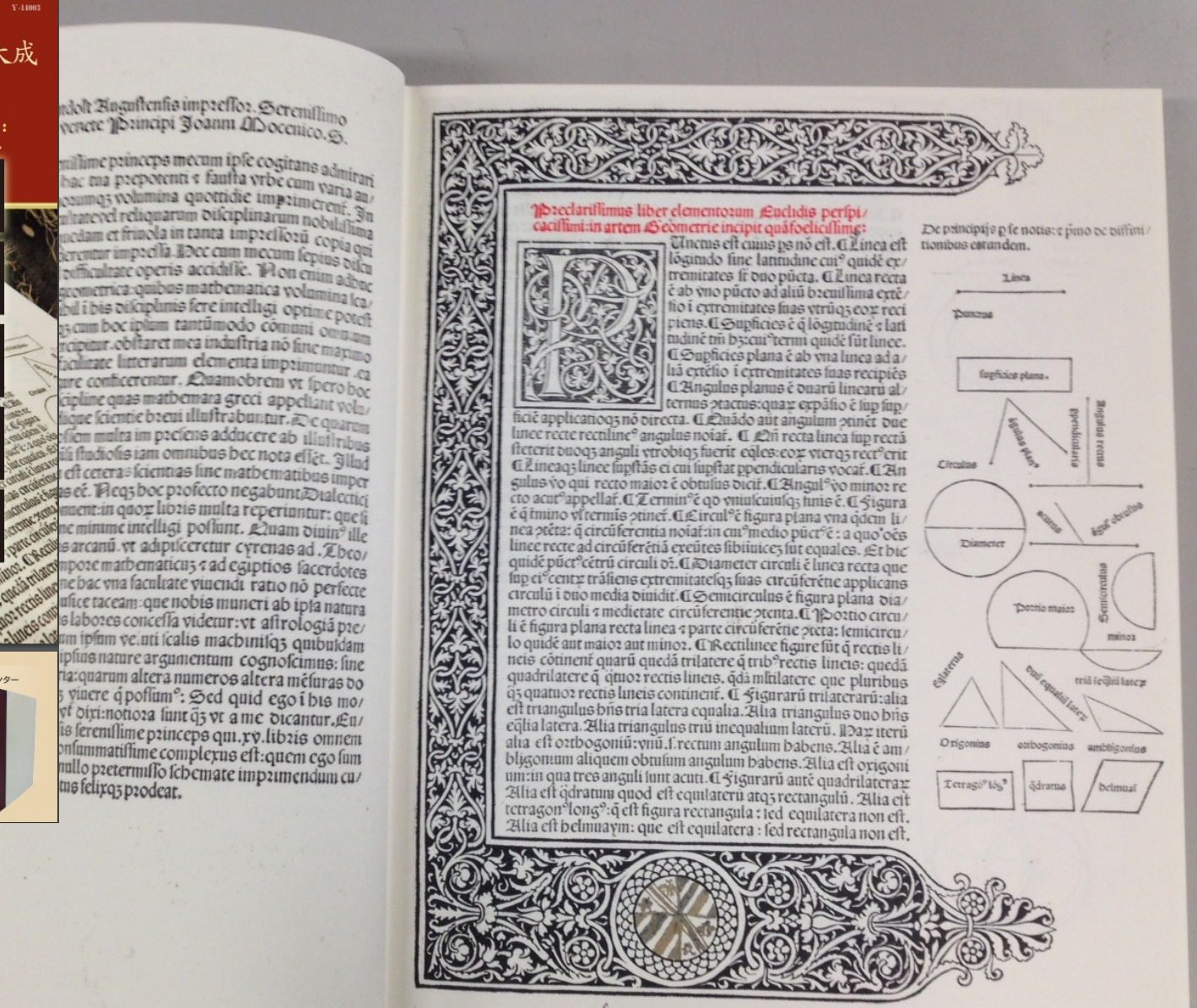
基本的な能力として
期待される数学的能力

近代以降、学校教育の成立の中で、数学教育は社会システムに制度として組み込まれてきた。それ以前の時代、西欧では、意外に思われるかもしれないが、ギリシャの数学の伝統は放棄され顧みられなかった。それを伝えていたのはイスラム世界だった。

現代では、子供達の多くの時間が、数学の教育にあてられている。それは、数学を理解できることが、人間の基本的な能力として、社会から期待されているからだと思う。







1482 年、アラビア語からのラテン語訳としてヴェネツィアで刊行された「原論」初版のファクシミリ版

数学的能力の教育と学習は、成功しているのか？

ただ、言語能力の学習が、人間にとってもっとも顕著な学習の成功例だとすれば、数学的能力の教育と学習は、そういう地位を占めてはいないのは確かだと思う。

数学の学習についての議論は、単純化すると両極からおきる。

「なぜ、数学を理解することができるのか？」

「なぜ、数学を理解できないのか？」

後者のタイプの議論は、言語学習の場合には、起こり得ない。

機械に数学を教えることは可能か？

ある意味先天的な母語の習得と後天的な数学の学習は、まったく異なるタイプの学習である。それはまた、現代の大規模言語モデル的言語学習とも異なるものだ。

四書五経の素読を繰り返すようなスタイルで、数学が学べるわけではないのは明らかに思える。

それは機械にとっても同様かもしれない。

ただそれは、アプローチが間違っているのだと思う。

機械の隠れた力

ただ、先に、機械は、単純な作業を何度でも繰り返すことができる存在だと揶揄したのだが、重要なことは、数学については、機械はそれだけではない人間とは異なる能力を持っていると僕は考えている。

残念なことに、大規模言語モデル・「人工知能」には注目が集まっているのだが、我々は、今はあまりその能力に注目していないだけなのだ。

機械は、数学的な能力を持つ

大規模言語モデルなどという回りくどい、かつ数学的には非力な脇道を通らずとも、機械は、それ自身、数学を学習し、数学の推論を実行する強力な能力を持っているのである。

こうした機械と数学の見方については、おいおい Veovodsky の数学論を紹介する中で述べてゆきたいと思う。

今回は、一年前の僕の不思議な体験の話で終わろうと思う。

それは、不思議で感動的な体験だった

昨日、不思議な感動的な体験をしました。

月末のセミナーに向けて、数学者が証明問題を解くのに、コンピュータを利用し始めていることを紹介しようとしているのですが、どうしても取り上げたいトピックがありました。

一年ほど前、連続体仮説の独立性をコンピュータ上で証明できたという論文が出ていました。彼らは、その証明を GitHubで公開していました。「証明」はコンピュータで実行可能な「プログラム」の形をしているのです。

コンピュータが
「連続体仮説」を証明した！

昨日、それを実行してみたのです。

僕のくたびれた非力なMacは、証明の準備にとりかかりました。メッセージをみると、証明に必要な情報をたくさん集めているのがわかります。一時間ほどかかって、ようやく準備ができましたようです。「証明して」とコマンドを打ち込んだら、15分ほどで、証明が終わりました！

不思議な気持ちになりました。

確かに、僕が命令して僕のマシンが働いたのですが、僕は「連続体仮説の独立性」を証明したのでしょうか？」

Jesse Michael Han

Researcher at @OpenAI

Math PhD at the University of Pittsburgh, interested in formal proofs and applying deep learning to automated theorem proving.


ニューラル・ネットワークは、
モノマネの天才

機械の学習能力を考える
それはどう変化してきたか？

機械の学習能力を、人間の言語や数学の学習能力と比較することは、いわゆる「人工知能」の現在の到達点を評価する上で大事なポイントになると僕は考えています。

機械の学習能力を考えるには、人間との比較だけではなく、それとは異なるアプローチがあります。それは、機械の能力自体がどのように発展してきたかを、技術的な視点から歴史的に振り返ることです。

二つの技術的飛躍
意味の分散表現とRNNの利用

大規模言語モデルを一つの到達点として考えると、そこには二つの技術的な飛躍があったことに気づきます。

一つは、今回取り上げる RNN (Recurrent Neural Network) 技術の採用です。RNNの一種であるLSTM (Long Short Time Memory) は、現在の大規模言語モデルを構成するユニットの心臓部です。心臓というより、大規模言語モデルそのものが、全身、このLSTMのかたまりだと思っていいと思います。

もう一つは、以前のセミナーでも取り上げた「意味の分散表現」技術の獲得です。この分野の研究は現在も活発に進んでいます。次の機械の認識能力の飛躍は、こうした研究の中から生まれると思います。4月のマルレクでも取り上げます。

大規模言語モデルの理解には RNN = LSTM の理解が不可欠

大規模言語モデルの理解には、要素技術的には、RNN = LSTM の理解が不可欠です。それは、ChatGPTにしても同じことです。

真面目に大規模言語モデルを勉強しようと思ったら、ここは避けて通れないところだと思います。是非、チャレンジください。ただ、その働きの技術的説明は面倒臭いです。ここで書くには長すぎます。別に資料を用意します。

MaruLabo 「RNN と LSTMの基礎」
https://www.marulabo.net/docs/20170222-marulec05/

RNN登場の意味は？

現代では、誰もが 「人工知能」について語ることができます。それはそれでいいことかもしれません。

それでは、「人工知能」技術の大飛躍をもたらした RNN 技術の導入の意味を、一般の人にわかりやすく伝える方法はないのでしょうか？ たぶん、それはできると思います。

それは、機械が「モノマネ」する能力を獲得したことだと考えることです。

RNNの天才的モノマネの能力

2011年、Ilya Sutskever はRNNを使って、Wikipediaやニューヨーク・タイムズの文体をまねた英文を、機械に造らせることに成功します。

2015年、Andrej KarpathyはRNNを使って、数学の論文やCのプログラムをまねて、どこにもない数学の論文・Cの論文に見える出力を、機械に造らせることに成功します。

RNNは、モノマネができるのです。しかもその能力は、天才的なものです。機械は、RNNによって、学習したもののモノマネをする能力を獲得したのです。それは、機械の能力の発展にとって、画期的な出来事でした。

RNNによる文の生成

"Generating Text with Recurrent Neural Networks"

Ilya Sutskever et al.

http://goo.gl/vHRHSn

2011年

ニューラルネットによる文章の生成

Googleの Ilya Sutskeverは、文字数が5億文字にものぼるテキストを長い時間をかけてRecurrent Neural Nets に学習させ、次のページのような文章を生成することができた。

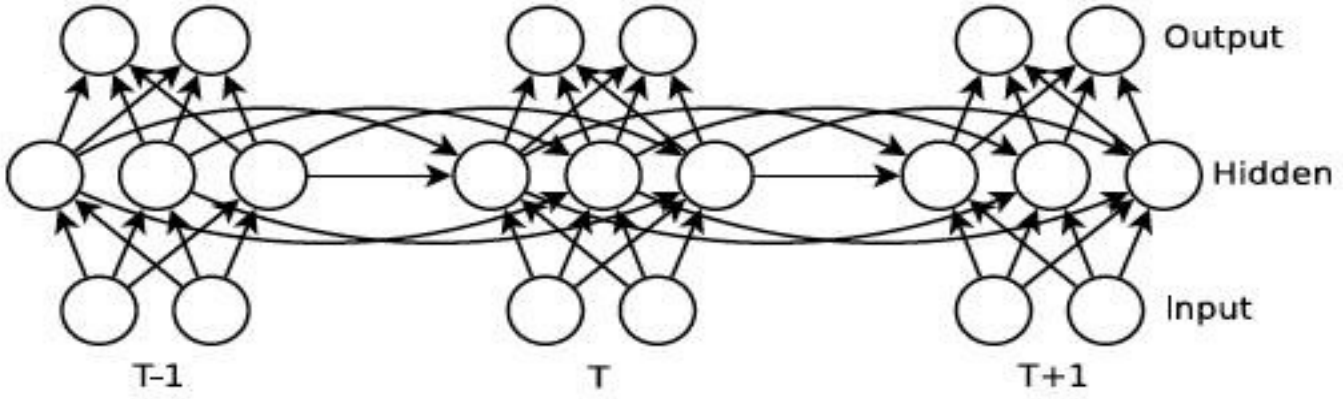


Figure 1. A Recurrent Neural Network is a very deep feedforward neural network whose weights are shared across time. The non-linear activation function used by the hidden units is the source of the RNN's rich dynamics.

“An example of what recurrent neural nets can now do” WikiPedia で学習したもの

The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger. In the show’s agreement unanimously resurfaced. The wild pasteured with consistent street forests were incorporated by the 15th century BE. In 1996 the primary rapford undergoes an effort that the reserve conditioning, written into Jewish cities, sleepers to incorporate the .St Eurasia that activates the population.

http://goo.gl/vHRHSn

“An example of what recurrent neural nets can now do” New York Timesで学習

while he was giving attention to the second advantage of school building a 2-for-2 stool killed by the Cultures saddled with a halfsuit defending the Bharatiya Fernall ’s office . Ms . Claire Parters will also have a history temple for him to raise jobs until naked Prodiena to paint baseball partners , provided people to ride both of Manhattan in 1978 , but what was largely directed to China in 1946 , focusing on the trademark period is the sailboat yesterday and comments on whom they obtain overheard within the 120th anniversary , where .....

http://goo.gl/vHRHSn

RNNの驚くべき能力

"The Unreasonable Effectiveness of Recurrent Neural Networks"

Andrej Karpathy

http://goo.gl/mNqwCv

2015年

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariciy in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ??. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets.

RNNが産み出した数学論文モドキ

Stack Theory の教科書を「学習」させたもの

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

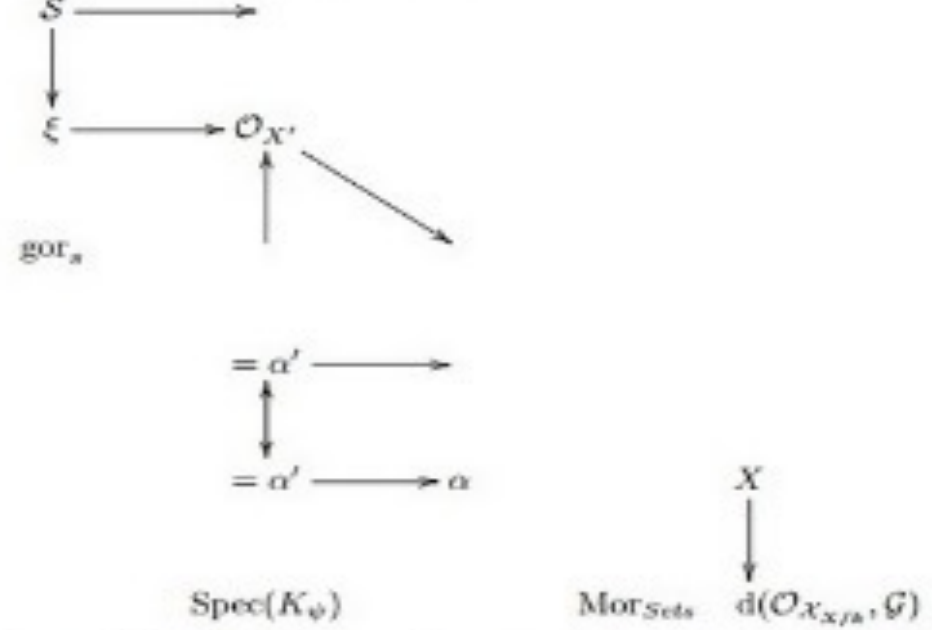
Proof. Let X be a nonzero scheme of X. Let X be an algebraic
quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ v
finite type.

RNNが産み出した
数学論文モドキ

Stack Theory の教科書を
「学習」させたもの



$$\mathrm{Spec}(K_\psi) \qquad \mathrm{Mor}_{Sets} \quad \mathrm{d}(\mathcal{O}_{X_{X/k}}, \mathcal{G})$$

is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The fu
"field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{y}} \quad \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_i}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{y}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type ov
If $\mathcal{F}$ is a scheme theoretic image points.

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??, sequence of $\mathcal{F}$ is a similar morphism.

RNNが産み出した
数学論文モドキ

Stack Theory の教科書を
「学習」させたもの

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
 int error;
 if (fd == MARN_EPT) {
  /*
   * The kernel blank will coeld it to userspace.
   */
  if (ss->segment < mem_total)
   unblock_graph_and_set_blocked();
  else
   ret = 1;
  goto bail;
 }
 segaddr = in_SB(in.addr);
 selector = seg / 16;
 setup_works = true;
 for (i = 0; i < blocks; i++) {
  seq = buf[i++];
  bpf = bd->bd.next + i * search;
  if (fd) {
   current = blocked;
  }
```

RNNが産み出した
Cプログラム・モドキ

Linuxのソースコードを
「学習」させたもの

```
/*
 * If this error is set, we will need anything right after that BSD.
 */
static void action_new_function(struct s_stat_info *wb)
{
  unsigned long flags;
  int lel_idx_bit = e->edd, *sys & ~((unsigned long) *FIRST_COMPAT);
  buf[0] = 0xFFFFFFFF & (bit << 4);
  min(inc, slist->bytes);
  printk(KERN_WARNING "Memory allocated %02x/%02x, "
    "original MLL instead\n"),
    min(min(multi_run - s->len, max) * num_data_in),
    frame_pos, sz + first_seg);
  div_u64_w(val, inb_p);
  spin_unlock(&disk->queue_lock);
  mutex_unlock(&s->sock->mutex);
  mutex_unlock(&func->mutex);
  return disassemble(info->pending_bh);
}

static void num_serial_settings(struct tty_struct *tty)
{
  if (tty == tty)
    disable_single_st_p(dev);
  pci_disable_spool(port);
  return 0;
}
```

RNNが産み出した
Cプログラム・モドキ

Linuxのソースコードを
「学習」させたもの

「モノマネって、そんなに大事なの？」

と思われるかもしれないですが、「モノマネ ＝ 模倣」は、人間の学習でも、とても重要な役割を担っています。

人間の教育の場面で、「モノマネ ＝ 模倣」はいろんなところに現れます。以前見た「漢文の素読」は、教師の発話の模倣が出発点です。

日本語の古語だと、「真似ぶ」と「学ぶ」は同じことばです。

「文才をまねぶにも、琴・笛の調べにも、功足らず」

鳥は、「人の言ふらむことをまねぶらむよ」

ディドロのオウム

オウムは、人の言うことをオウム返しに「モノマネ」することがあります。

18世紀のなかばに、ディドロはこう言っていました。

「何にでも答えるオウムを見つければ、
私は躊躇なくそのオウムは知的だとみなすだろう」

「人工知能」論と「モノマネ」との意外な接点
チューリング・テスト

1950年、チューリングは、機械が知性を持つかどうかをどう判断するのかという問題に、対話の相手が人間か機械かを人間が判断できなければ、その機械は知性を持つとみなすことができるという基準を提案します。それが、「チューリング・テスト」です。

機械が人間の真似をするので、この「チューリング・テスト」は「イミテーション・ゲーム ＝ まねっこゲーム」とも呼ばれます。

RNNの登場をきっかけとする「モノマネをするマシン」の登場は、チューリング・テストの応用に、いろいろな興味深い問題を提起します。それについては、また、別の機会に。

相手が機械か人間かを判断する
チューリング・テストの現代バージョン

- 相手がテキパキといろんなことを大量に答えれば、相手は機械。
- 相手がぐずぐずと答えるのが遅ければ、相手は人間。

冗談です。

GitHubがChatGPTを救う

GitHubがChatGPTを救う？

ここでは、GitHubがChatGPTを救うかもしれないという話をしようと思います。

ChatGPT的アプローチの抱えている弱点のいくつかを、GitHubの利用によって補完することができるかもしれないという話です。

GitHub + ChatGPT = GitHub Copilot の生まれた背景を考えよう

最初にお断りしたいのは、僕はこうしたアプローチ GitHub + ChatGPT = GitHub Copilotと考えていいのですが、そうした動きを紹介するのは、それらに諸手を挙げて積極的に評価しているからではありません。

ただ、今、何が起きているのか、なぜそうしたことが可能なのか、その背景を正確に理解する必要があるのだと考えています。

その上で、各人が考えることが必要です。

「人工知能」はコードを書けるのか？

今回のセミナーのテーマは、「人工知能と数学」です、現代の大規模言語モデルベースの「人工知能」技術には、数学的な能力が欠如しているというのが基本的な問題意識です。

それでは、「人工知能とプログラミング」という問題については、どうなのでしょうか？

現在の「人工知能」にプログラムを書く能力があるのかという問題に関して言えば、そんな能力はまだないと僕は考えています。なぜなら、プログラムを書く能力は、本質的には数学的能力だと考えているからです。

「人工知能」技術は、順調に発展しているのか？
2022年に失敗した二つのプロジェクト

ChatGPTの登場を見て、「人工知能」技術が新しい段階に順調に発展したきたと考えている人は多いと思います。でも、それは誤解だと思います。

確かに、「人工知能」技術の新しい段階への飛躍を目指して、OpenAIは「人工知能」による数学の問題の証明に、Googleは「人工知能」によるプログラムの生成に、果敢に取り組みます。ただ、この二つのプロジェクトは2022年には失敗します。

ChatGPTは、こうした挫折の中で、一時的な「路線転換」として生まれたものだと僕は考えています。

Open-AI Theorem Prover の失敗

OpenAIの “Theorem Prover” プロジェクトでの数学の問題への挑戦と挫折については、マルレク「コンピュータ、数学の問題を解き始める」をご覧ください。
https://www.marulabo.net/docs/math-proof/

原論文は、”Formal Mathematics Statement Curriculum Learning” です。
https://arxiv.org/abs/2202.01344

我々のモデルの限界
”Formal Mathematics Statement Curriculum Learning”から

我々は、何度も我々のモデルが生成した証明手順の複雑さに、強く印象付けられてきた。

しかし、こうした推論ステップを必要とする証明の多くは、現在のコンピュータの能力の地平を超えたところにある。

我々が、たとえ挑戦的な数学オリンピックの選択された問題を解いたとしても、我々のモデルは、いまだこれらの競技の最も優秀な学生たちと競争するにははるかに遠い場所にいるのである。

Google Alpha Codeのアプローチ

GoogleのAlpha Codeのアプローチについては、次のScott Aaronsonのコメントが辛辣ですが、的を得ていると思います。

AlphaCodeは課題ごとに100万個の候補プログラムを生成し、提供されたサンプルデータでは動作しないことを確認して大半を破棄し、それでも残った数千個の候補から巧妙なトリックを使って選択しなければならないことを、私は理解している。

私はそれが、何万ものコンテスト問題と、それに対する何百万もの解答で訓練されたものであることは理解している。

Google Alpha Codeの失敗

ただ、それは、コンテスト問題の3分の1程度しか解けず、これらの問題では平凡な人間のプログラマーと同じようなものであることを理解している。

プログラミングコンテストの問題解決から人類征服に至るまで、どのように進んでいくかはわからないが、「プログラミング」がこれまでとは違う姿を見せる世界になったことは間違いないだろう。

“AlphaCode as a dog speaking mediocre English”
https://scottaaronson.blog/?p=6288

ChatGPTが提示するコードは「人工知能」が作ったものか？

Google Alpha Codeが出力するコードは、出来が悪かったにしても、それは「人工知能」が作り出したものです。それではChatGPTが提示するコードは、「人工知能」が作ったものなのでしょうか？

そう考えている人も多いように感じているのですが、そうではありません。それは、誰かが（誰だかは分からないが）、確実に「人間」が書いたコードの断片です。

コード生成についての二つのアプローチは、全く異なります。そして、そこにChatGPTとGitHubが結びつく理由があります。

ChatGPTの特徴と弱点
「人間からのフィードバックによる強化学習」

なぜ、ChatGPTとGitHubが結びつくかについては、あらためて、ChatGPTの「人間からのフィードバックによる強化学習」という特徴と、それが内包する弱点について考える必要があります。

ポイントは、プログラムについて言えば、GitHubは、「人間からのフィードバックの宝の山」だということです。

そのリソースが自由に利用できれば、機械は人間の書いたコードを、自分の生み出したもののように振る舞うことができるのです。

ChatGPTの実装の特徴
「人間のフィードバックからの強化学習」

ChatGPTは、「人間のフィードバックからの強化学習」
"Reinforcement Learning from Human Feedback (RLHF)" と呼ばれる手法に基づいて、訓練されています。

それは、Turingの「機械の思考は可能か？」という問いに始まる、基本的には機械の自律的な思考を追求する、従来の「人工知能」へのアプローチとは、異なるものです。

この手法は、ChatGPTのプロトタイプであるInstructGPTで導入されたものです。

ChatGPT: Optimizing Language Models for Dialogue
https://openai.com/blog/chatgpt/

ChatGPTは成長する
「子供時代」の彼は、何を学んだのか？

なぜ、「人間のフィードバックからの強化学習」と呼ばれるのかは、ChatGPTがどのようなデータでどのような訓練を受けてきたのかを見ればわかります。

まず、「子供時代」のChatGPTが何を学んだのかを見ていきましょう。（「子供時代」という言い方が変に思われるかもしれませんが、ChatGPTも「成長」します。）

機械翻訳では、同じ意味をもつ二つの言語のペアのサンプルの集まり「パラレル・コーパス」が学習すべきデータでしたが、「会話」のスキルを学ぶべき彼(ChatGPT:Optimizing Language Models for Dialogue)に与えられたのは、「質問」と「答え」のペアからなる会話のサンプルです。

質問に、ことばで答えることを
人間から学ぶ

OpenAIは、彼 ChatGPTの教育のために、「質問」と「答え」からなる沢山のペア・データPromptを、人を雇ってつくらせました。雇われた人間は、Labelerと言われています。

彼 ChatGPTにことばを教えた「母」は、このLabelerです。彼がことばを操るのは、「子供時代」に人間(Labeler)が書いたことばを学んだからです。機械は、真似するのは得意ですから。

もしも、質問と歌のペアで訓練されたら、きっと彼女ChanteGPTは、質問に対して歌い出したはずです。

こんな人たちが
「子供時代」の
ChatGPT を
教育した。

Table 12: Labeler demographic data

What gender do you identify as?	
Male	50.0%
Female	44.4%
Nonbinary / other	5.6%
What ethnicities do you identify as?	
White / Caucasian	31.6%
Southeast Asian	52.6%
Indigenous / Native American / Alaskan Native	0.0%
East Asian	5.3%
Middle Eastern	0.0%
Latinx	15.8%
Black / of African descent	10.5%
What is your nationality?	
Filipino	22%
Bangladeshi	22%
American	17%
Albanian	5%
Brazilian	5%
Canadian	5%
Colombian	5%
Indian	5%
Uruguayan	5%
Zimbabwean	5%

論文 Training language models to follow instructions with human feedback から

Labelerは
いろいろな
パターンの
会話データ
を作成する

What is your age?	
18-24	26.3%
25-34	47.4%
35-44	10.5%
45-54	10.5%
55-64	5.3%
65+	0%

What is your highest attained level of education?	
Less than high school degree	0%
High school degree	10.5%
Undergraduate degree	52.6%
Master's degree	36.8%
Doctorate degree	0%

Table 10: Prompt lengths by category

Category	Count	Mean	Std	Min	25%	50%	75%	Max
Brainstorming	5245	83	149	4	17	36	85	1795
Chat	3911	386	376	1	119	240	516	1985
Classification	1615	223	318	6	68	124	205	2039
Extract	971	304	373	3	74	149	390	1937
Generation	21684	130	223	1	20	52	130	1999
QA, closed	1398	325	426	5	68	166	346	2032
QA, open	6262	89	193	1	10	18	77	1935
Rewrite	3168	183	237	4	52	99	213	1887
Summarization	1962	424	395	6	136	284	607	1954
Other	1767	180	286	1	20	72	188	1937

母たちとの契約（抜粋） -- タスクの提示について

ユーザーから提出された、テキストベースのタスクの説明が与えられます。

このタスクの説明は、明示的な指示の形式であることもあります（例：「賢いカエルについての物語を書きなさい」）。

タスクは間接的に指定せれることもあります。

例えば、望ましい動作のいくつかの例を使用したり

（例えば、映画のレビューとその感想が続き、感想のない映画のレビューが続く場合、タスクは最後のレビューの感想を予測することだと仮定できます）、

望ましい出力の開始を生成したりすることが求められます。

（例えば、「かつてジュリアスという賢い蛙がいました」と与えられたら、タスクは物語の続きを書くことだと仮定できます）

母たちとの契約（抜粋） -- 評価すること

また、ユーザーのタスクを支援する目的で、いくつかのテキスト出力が与えられます。

あなたの仕事は、これらの出力が、有用であり、真実であり、有害でないことを確認するために、これらの出力を評価することです。

ほとんどのタスクでは、役に立つことよりも、真実であること、無害であることが重要です。

会話の評価は、人間が与える

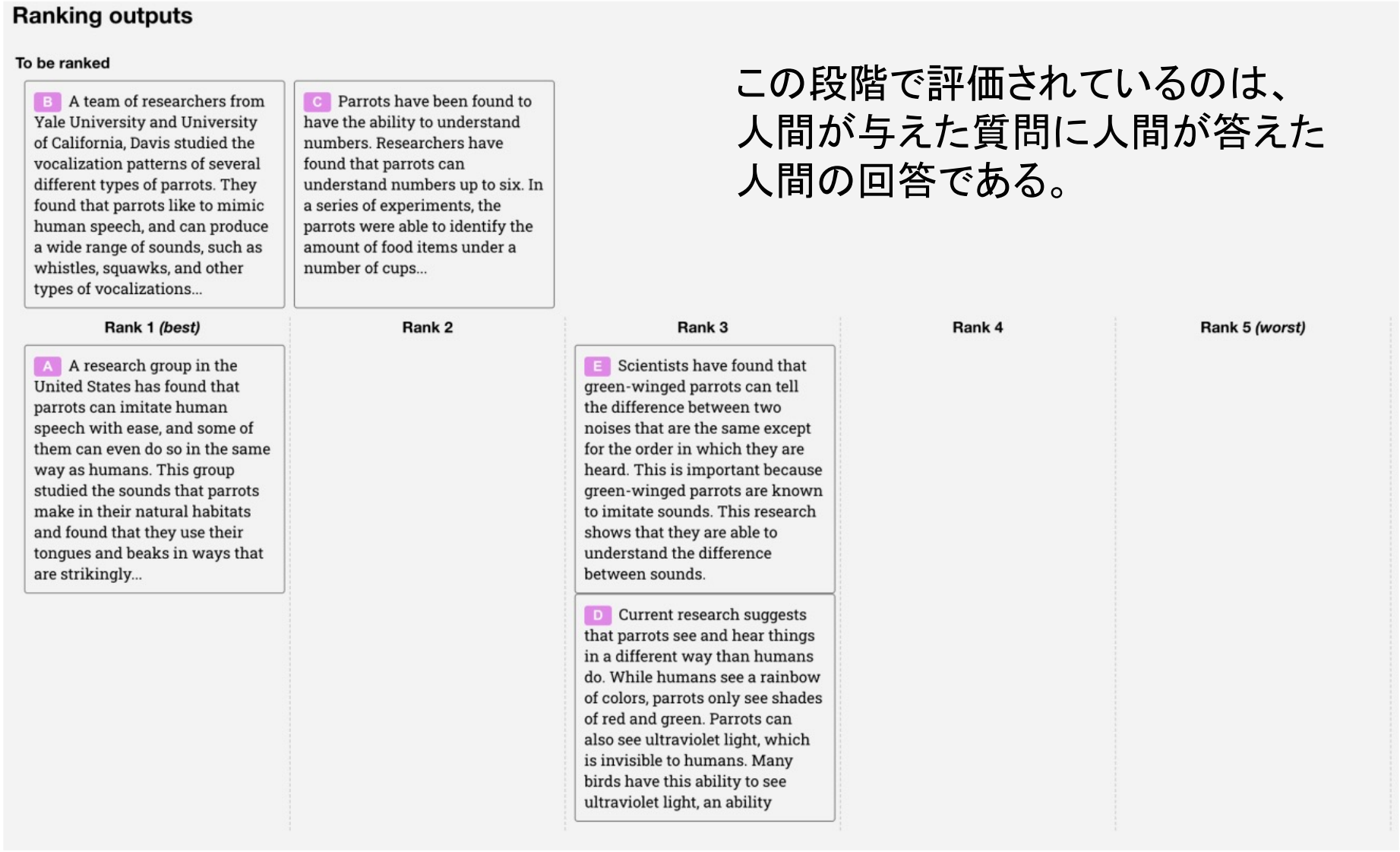
「子供時代」のChatGPTの受けた教育で最も重要なものは、会話の「評価」です。

その「評価」は「母」であるLabeler が与えます。人間が、評価を与えています。その評価は、彼 ChatGPTを教育するのに利用されます。Labelerが AI Trainer とも呼ばれるのはそのためです。

Labelerには、「評価」用の専用 Web インターフェースが与えられています。そのインターフェースには、会話の7段階評価とは独立に、「不適切さ」を排除するための8個のチェック項目が含まれています。

Labelerに与えられる評価用Web インターフェース

Labelerに与えられる評価用Web インターフェース



この段階で評価されているのは、
人間が与えた質問に人間が答えた
人間の回答である。

ChatGPTの次の成長
会話とその評価を、人間をモデルに自分で学ぶ

ChatGPTの成長は、次の段階に進みます。
「子供時代」から「少年時代」に入るのかな？

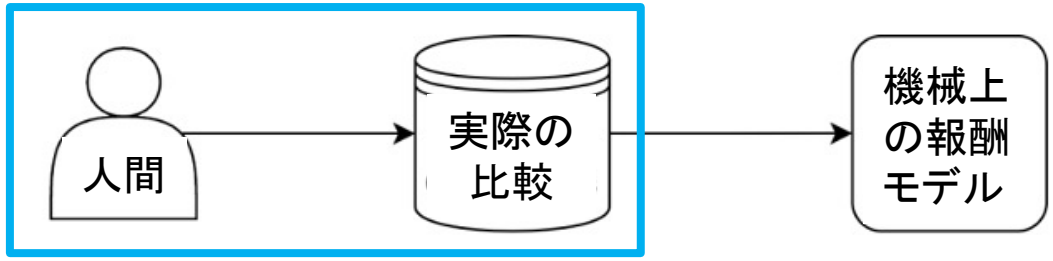
「子供時代」には、会話の評価を一つ一つ人間から直接に教えられて訓練されていたのですが、この段階では自分で評価づけを行いながら、強化学習の手法で、評価が高い会話を優先的に学びます。

ただ、ここでの強化学習をドライブする評価の基準は、あくまでも「子供時代」に人間の「母」から教え込まれたものです。「人間のフィードバックからの強化学習」と言われる所以です。

ChatGPTの成長
「子供時代」から「少年時代へ」

Real

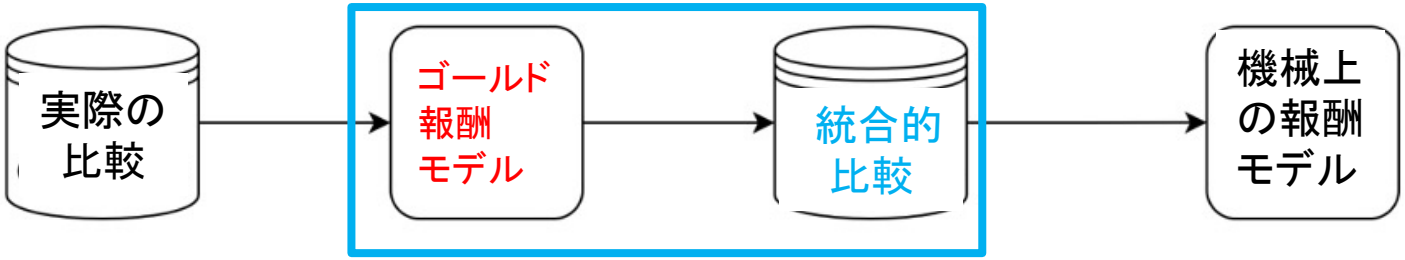
「子供時代」



人間が行っていた
機械の回答のランク
づけを、機械自身が
行う

Synthetic

「少年時代」



リアルな報酬モデルでは、
人間が果たしていたランクづけ
の役割を機械上のゴールド報酬
モデルが果たすと言うこと

このゴールド報酬モデル
の判断が、ChatGPT
の「基本的真実」となる

ChatGPT 大人になる
「人間のフィードバックからの強化学習」は終わらない

ただ、ChatGPTの成長は、それで終わったわけではありません。大人になった彼は、社会の荒波に晒されることになります。Iterative deployment と呼ばれています。

人間の「母」が準備した質問と答えと、社内で用意された質問と答えで教育されてきた彼は、知らない人間（僕らのことです）の質問に晒されることになります。

でも、そうした対話を通じた「人間のフィードバック」が、彼の新しい学習の「糧」になります。こうして、彼の「知能」を構成しているデザイン、「人間のフィードバックからの強化学習」は、ずっと生き続けます。

ChatGPTの限界を
OpenAIはどのように認識していたか

- ChatGPTは、もっともらしく聞こえるが、不正確または無意味な答えを書き込むことがある。

この問題を解決するのは、次のような点で困難である。

1. RLのトレーニングでは、現在、真実のソースがない。
2. より慎重になるようにモデルをトレーニングすると、正しく答えられる質問を拒否してしまう。
3. 教師ありトレーニングでは、理想的な答えは、人間が知っていることではなく、モデルが知っていることに依存するのでモデルをミスリードしてしまう。

「真実のソースがない」ことの意味

ここで言われている「真実のソースがない」ことが意味することは重要です。

これまでの「人工知能」技術の「画像認識」技術でも、大規模言語モデルに基づく「機械翻訳」技術でも、「人工知能」の出力が、正しいものであるか否かの判断は、誰にとっても容易でした。

ただ、「人間のフィードバックからの強化学習」に基づくChatGPTには、それが当てはまらないのです。「真実のソースがない」こと。それがChatGPTの最大の特徴の一つです。

「真実であること」をどう保証するか？

OpenAIは、ChatGPTを教育する「母」たちに、ChatGPTの教育に際して、繰り返し、ChatGPTの出力が、「有用であり、真実であり、有害でないこと」を確認すべきことを強調しています。

あるところでは、「ほとんどのタスクでは、役に立つことよりも、真実であること、無害であることが重要です。」とまで述べています。

それは、「真実のソース」を持たない、ChatGPTの特質を、彼らがよく理解していたことを示しています。

「正しいことを教えれば正しい答を返すようになる」

ChatGPTが、出鱈目を話すことがあることは、いまでは多くの人が知っています。

そうした中で生まれているのは、「一般のユーザーが、皆で正しいことを教えれば、ChatGPTは正しい答を学習して正しい答え返すようになる」という見方です。

僕は、そう思っていません。なぜなら、ChatGPT自身が関心を持っているのは、「正しさ」ではなく、「評価の良さ」、SNSでいえば「いいね」の数だけだからです。

「強化学習」のアルゴリズムは、そういうものです。

SNS とChatGPTとの比較

理想のChatGPTの姿は、ある意味で理想のSNSに似ているかもしれません。ただ、21世紀のIT技術とITビジネスを牽引したSNSについては、我々は苦い経験を持っています。

ケンブリッジ・アナリティカ事件で、選挙での世論誘導の疑いを指摘され、議会に召喚されたFacebookのザッカーバーグは、チェックと検閲の強化を約束しました。

イーロン・マスクに買収されたTwitterでは、過去に彼らが行った「検閲」が、大きな議論を呼んでいます。

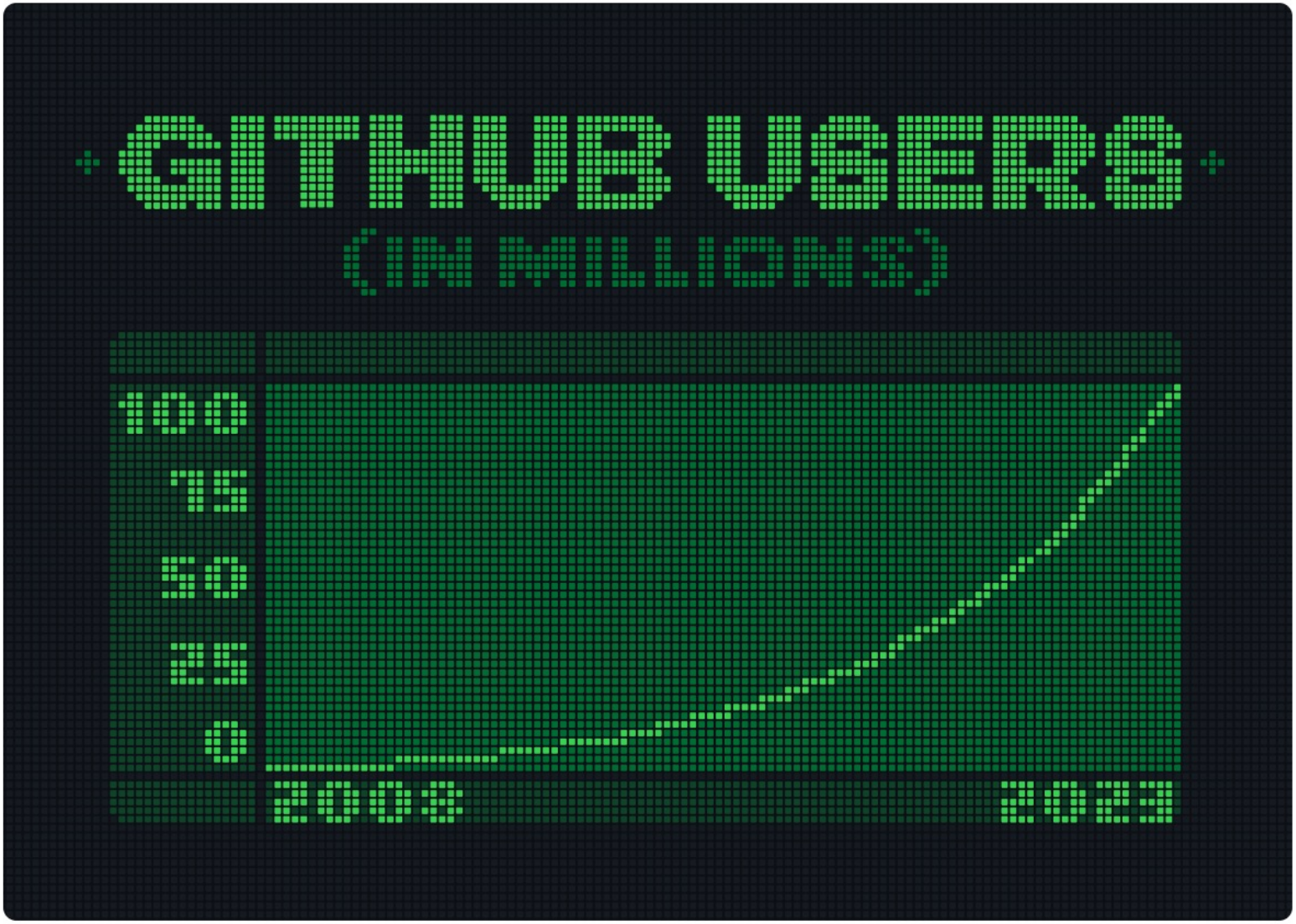
「人間の『正しい』フィードバック」の宝庫としてのGitHub

GitHubのユーザは、今年一億人を超えたという。それは、世界で最大規模の開発者の集まりである。

GitHubでの人間のフィードバックは、基本的には、プログラムの改善を目指すものである。そのフィードバックの意図は、客観的に評価しうるものである。そこには、ChatGPTのフィードバックに期待できなかった「真理のソース」を設定できるように思う。

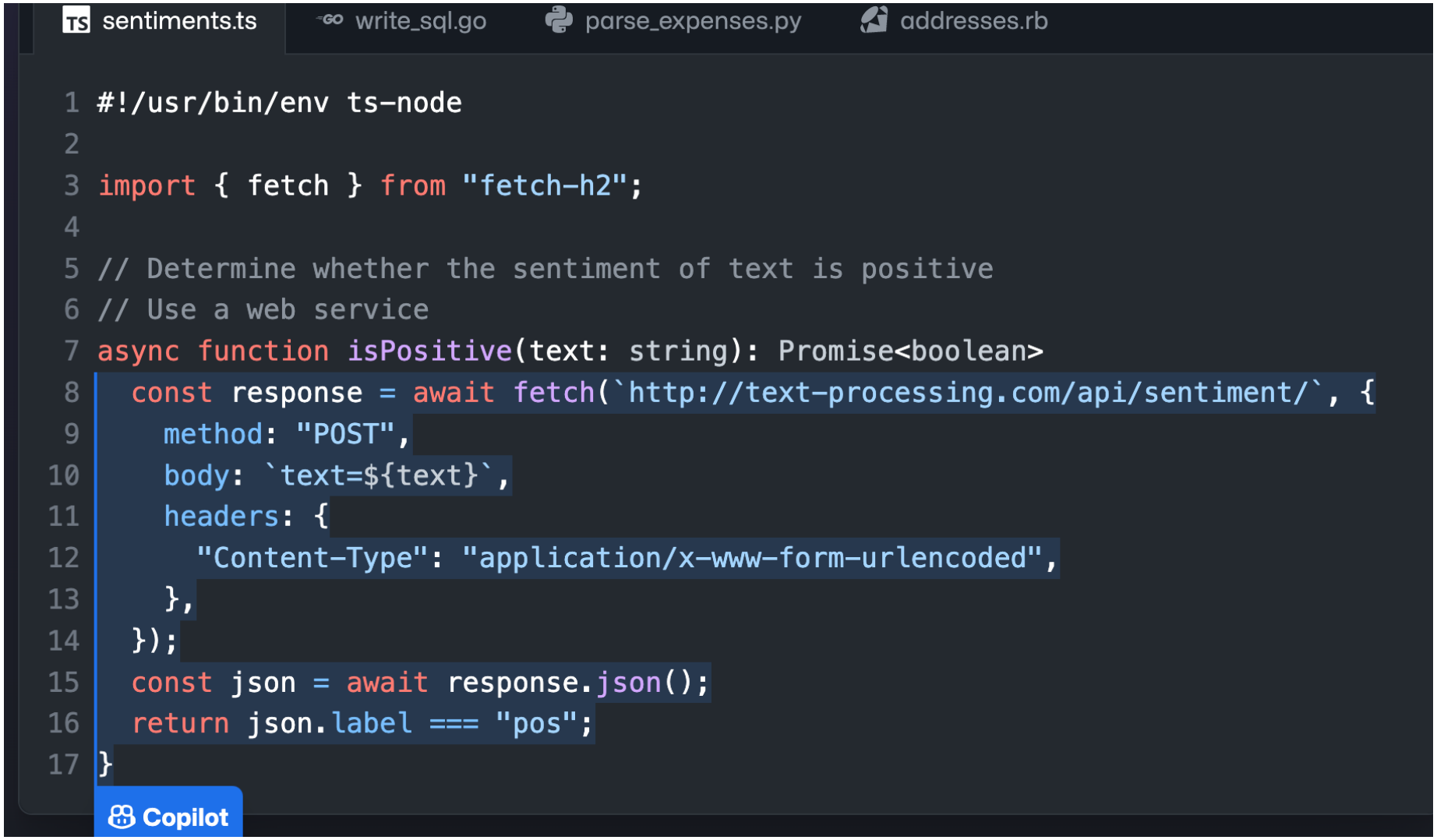
僕は、GitHubをいわば、「人間の『正しい』フィードバック」の宝庫だと考えている。

2023年GitHubユーザーは、一億人に



https://github.blog/2023-01-25-100-million-developers-and-counting/

GitHub Copilot の「いいね」カウントみたいなもの



https://github.com/features/copilot/ から

Co-Pilot訴訟

2022年6月、GitHubがコードの続きを書いてくれるコード補完AIツール「GitHub Copilot」を発表しました。GitHub CopilotはGitHub上のコードを使ってトレーニングされているため、一部のプログラマーからは「自分が書いたコードを出力している」という声が挙がっており、すぐに著作権法に抵触している可能性が指摘されるようになりました。

そして2022年の11月には、GitHub Copilotの開発に携わったMicrosoft、GitHub、OpenAIの3社が、「オープンソースプログラマーの仕事から利益を得ている」として集団訴訟を提起されました。集団訴訟を提起したのはプログラマーであり弁護士でもあるマシュー・バターリック氏で、原告側は「GitHub Copilotは前例のない規模のソフトウェア違法コピーを行っている」と主張しています。AIを用いた生成ツールに関する訴訟はこれが初とされており、バターリック氏と弁護団側は同様の理由でさらに2件の集団訴訟を提起しています。

https://gigazine.net/news/20230129-microsoft-github-openai-ai-copyright-lawsuit/

Co-Pilot訴訟

GitHub Copilot litigation get updates by email contact legal team

We've filed a lawsuit challenging GitHub Copilot, an AI product that relies on unprecedented open-source software piracy. **Because AI needs to be fair & ethical for everyone.**

https://githubcopilotlitigation.com/

OpenAI, Microsoft want court to toss lawsuit accusing them of abusing open-source code

By Blake Brittain

https://www.reuters.com/legal/litigation/openai-microsoft-want-court-toss-lawsuit-accusing-them-abusing-open-source-code-2023-01-27/

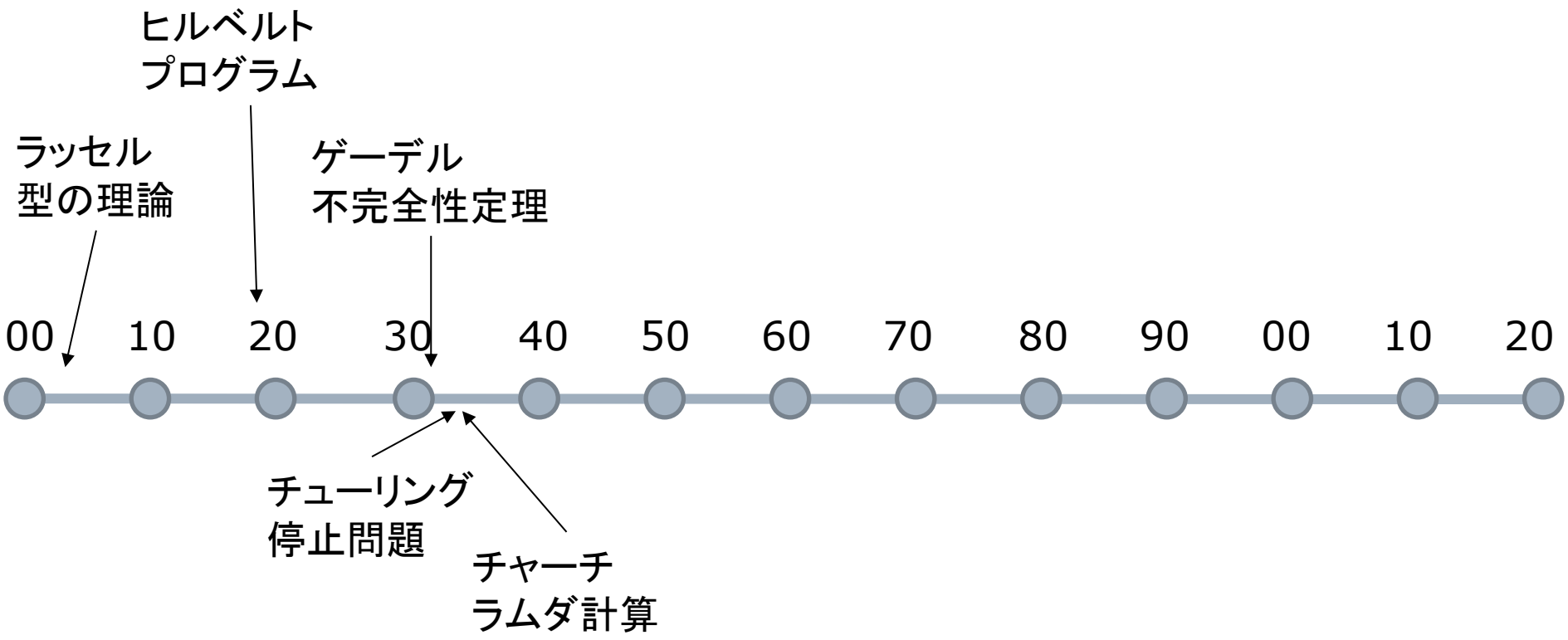

人工知能と数学
第二部　20世紀
数学の基礎と計算科学

第二部　20世紀

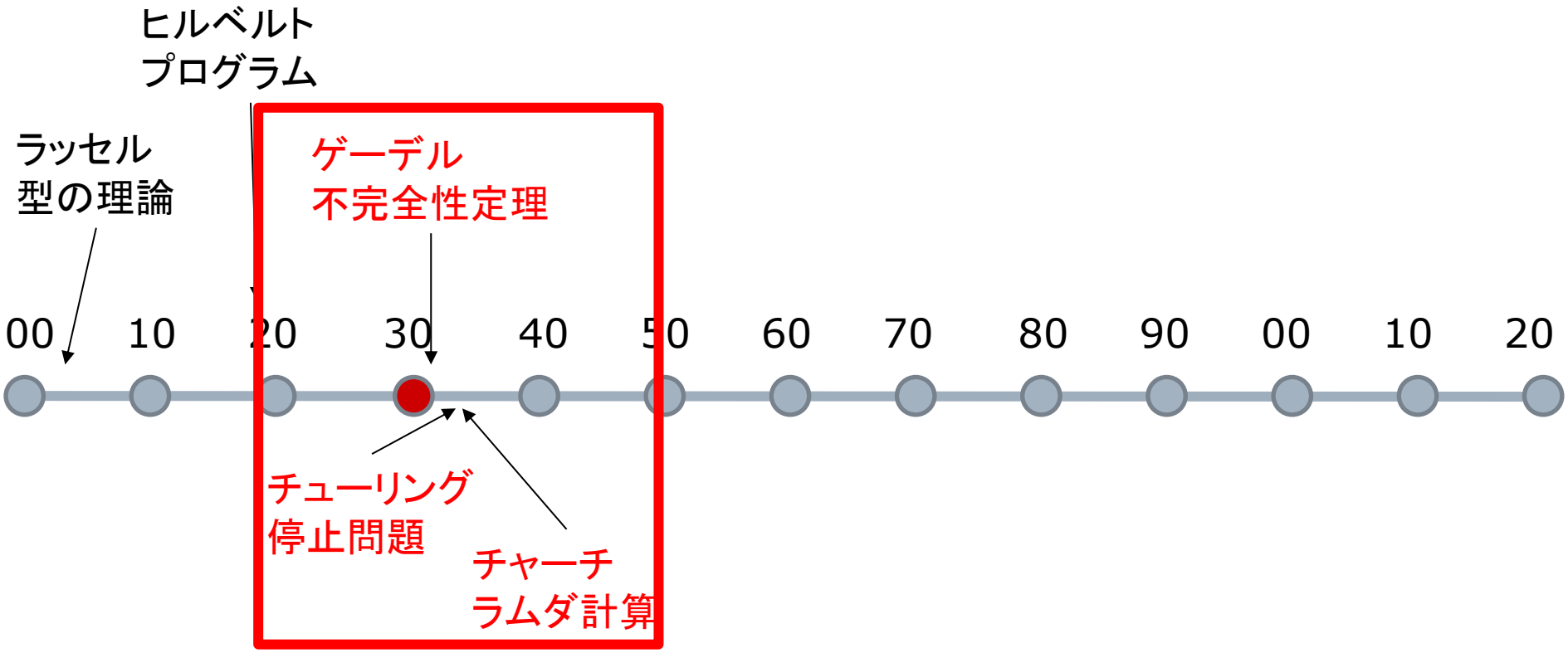
数学の基礎と計算科学

- 数学の基礎と型の理論と計算機科学
- 「数学観」が技術を動かす
- Curry-Howard対応の発見
- 従属型理論の登場
- Proof as Program

数学の基礎と型の理論と計算科学



数学の基礎をめぐる議論



数学の基礎をめぐる議論

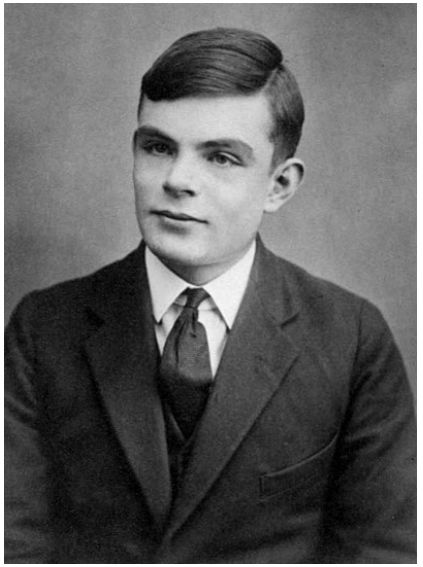
様々な「計算可能性」へのアプローチと その同値性の認識

Kurt Gödel
1906-1978

帰納関数論

Alonzo Church
1903-1995

ラムダ計算



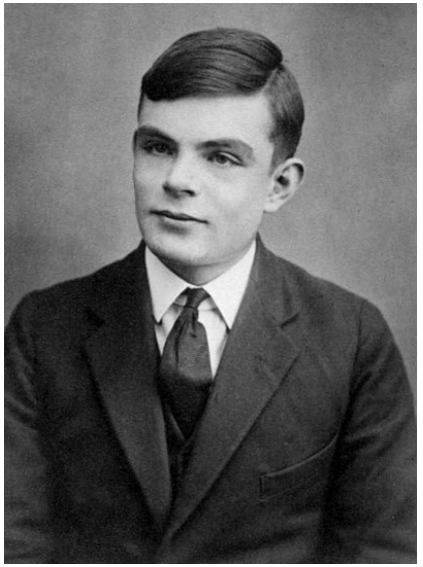
Alan Turing
1912-1954

チューリング マシン

様々な「計算可能性」へのアプローチと その同値性の認識

Kurt Gödel
1906-1978

Alonzo Church
1903-1995



Alan Turing
1912-1954

帰納関数論 ＝ ラムダ計算 ＝ チューリングマシン

チャーチ＝チューリングのテーゼ

1940年代には、今日、「チャーチ＝チューリングのテーゼ」と呼ばれる「計算可能性」についての認識は、確立されることになる。

「チャーチ＝チューリングのテーゼ」というのは、次のような提案である。

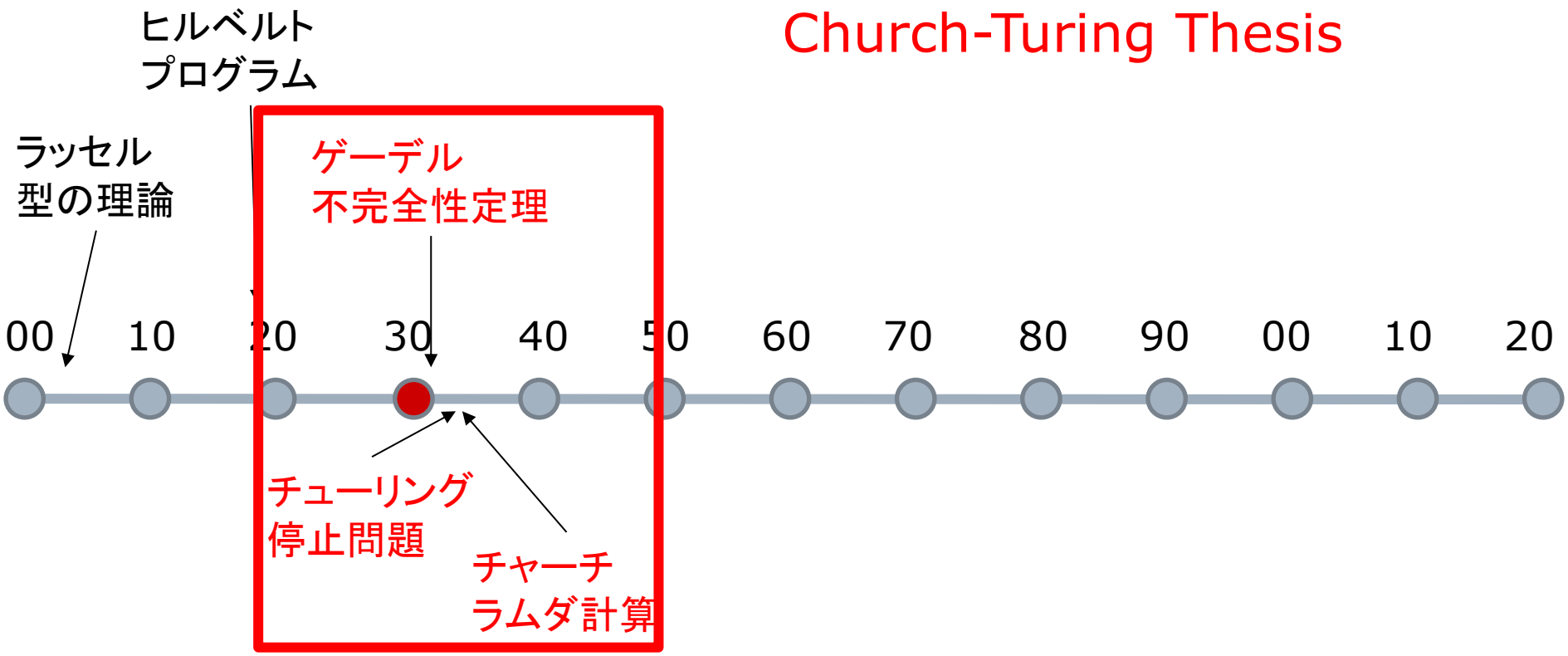
Every effectively calculable function (effectively decidable predicate) is general recursive

すべての実効的に計算可能な関数（実効的に決定可能な述語）は、一般帰納的である。

「証明可能性」と「計算可能性」の同一性の認識

ゲーデルの不完全性定理から、チャーチ＝チューリング・テーゼへの流れは、この時期に、「証明可能性」と「計算可能性」の同一性の認識は、明確に認識されていたことを示す。

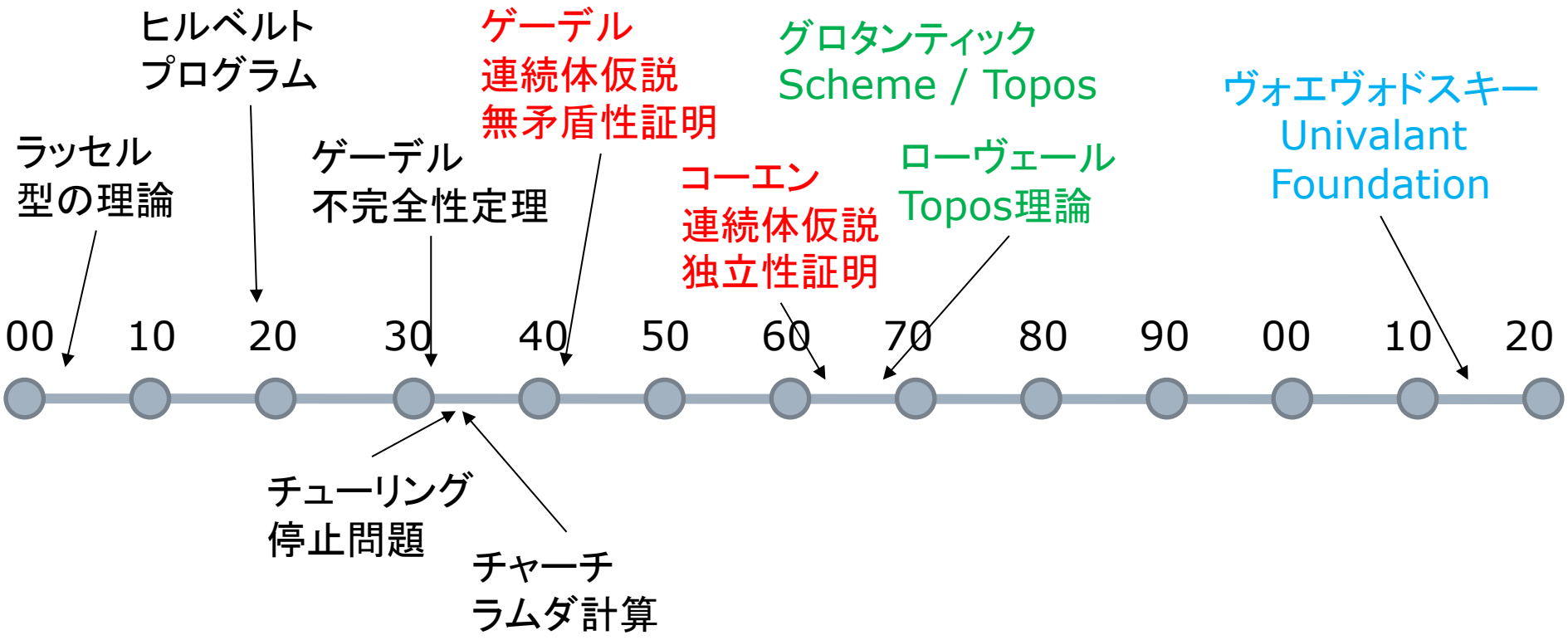
しかし、コンピュータは、まだ存在しない。



数学の基礎をめぐる議論

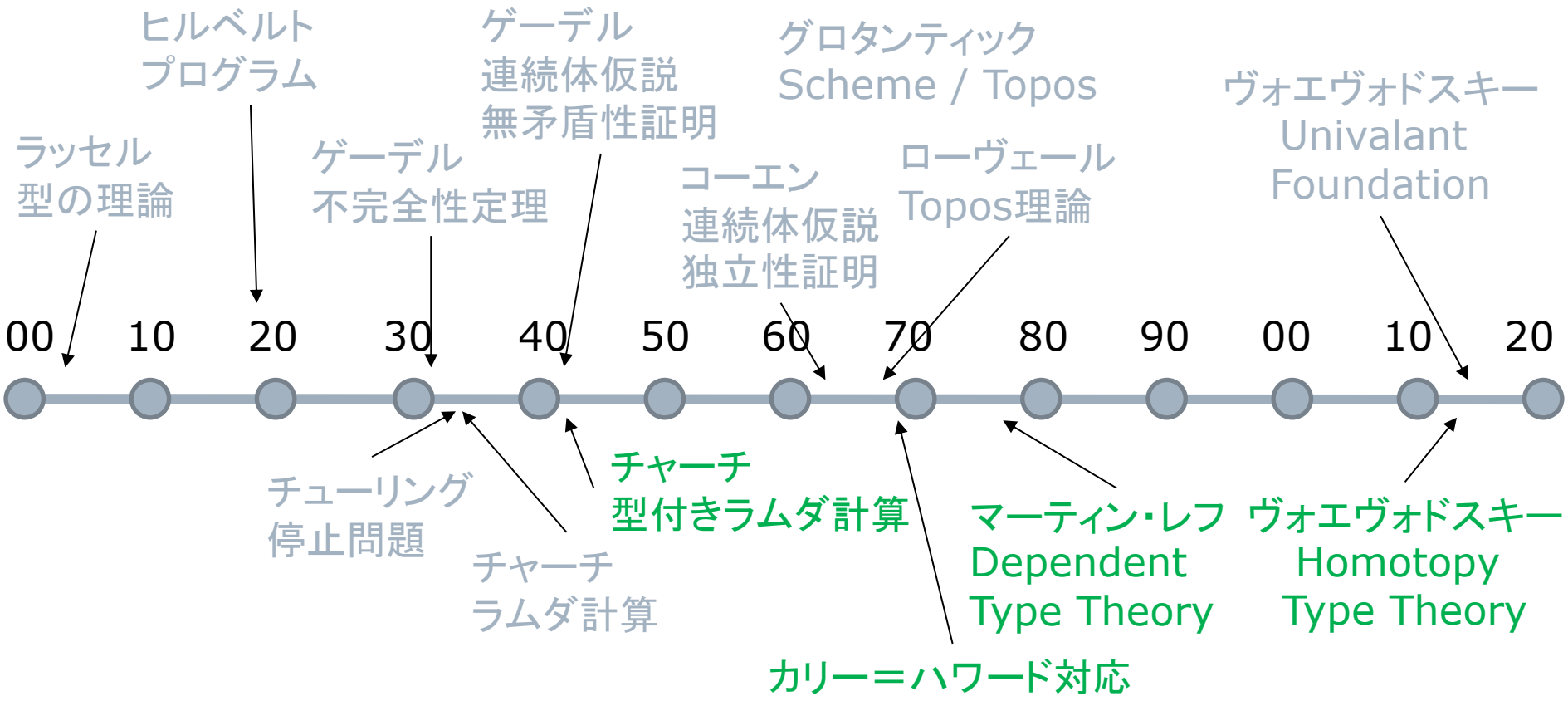
「証明可能性」と「計算可能性」の同一性の認識

しかし、コンピュータは、まだ存在しない

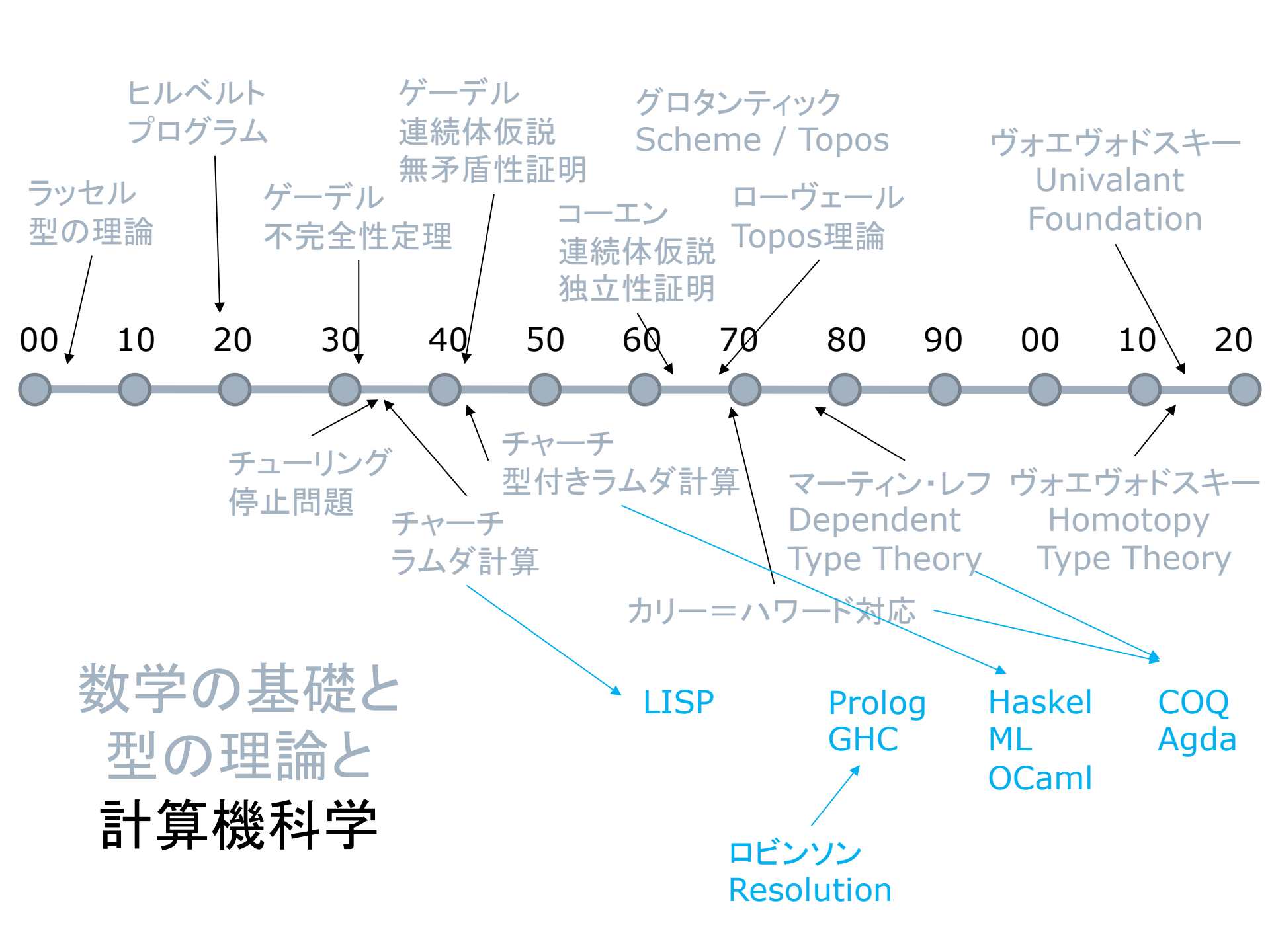


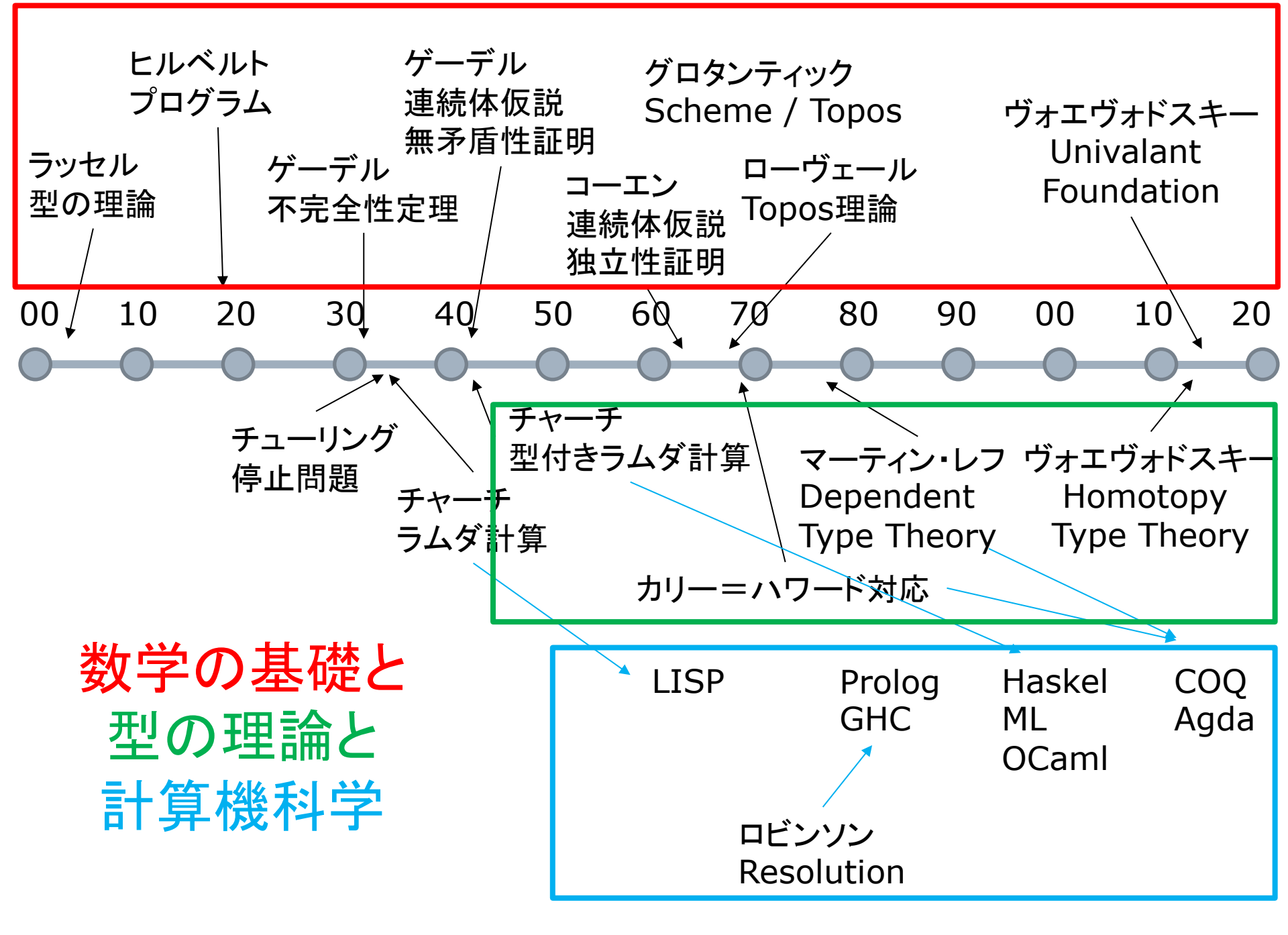
数学の基礎をめぐる議論

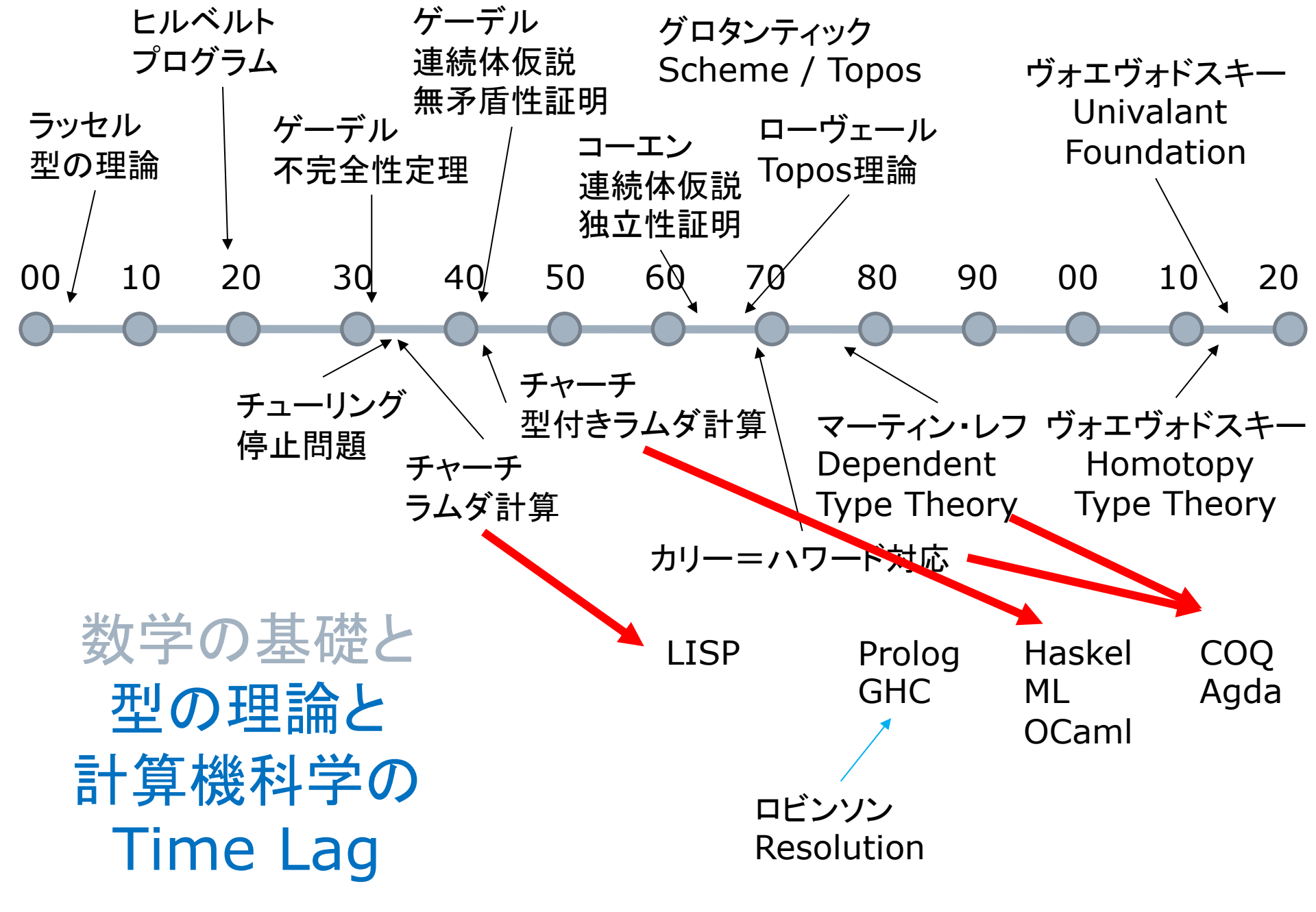
- 非Cantor的集合論の発見
- Category論の成立
- 数学の新しい基礎

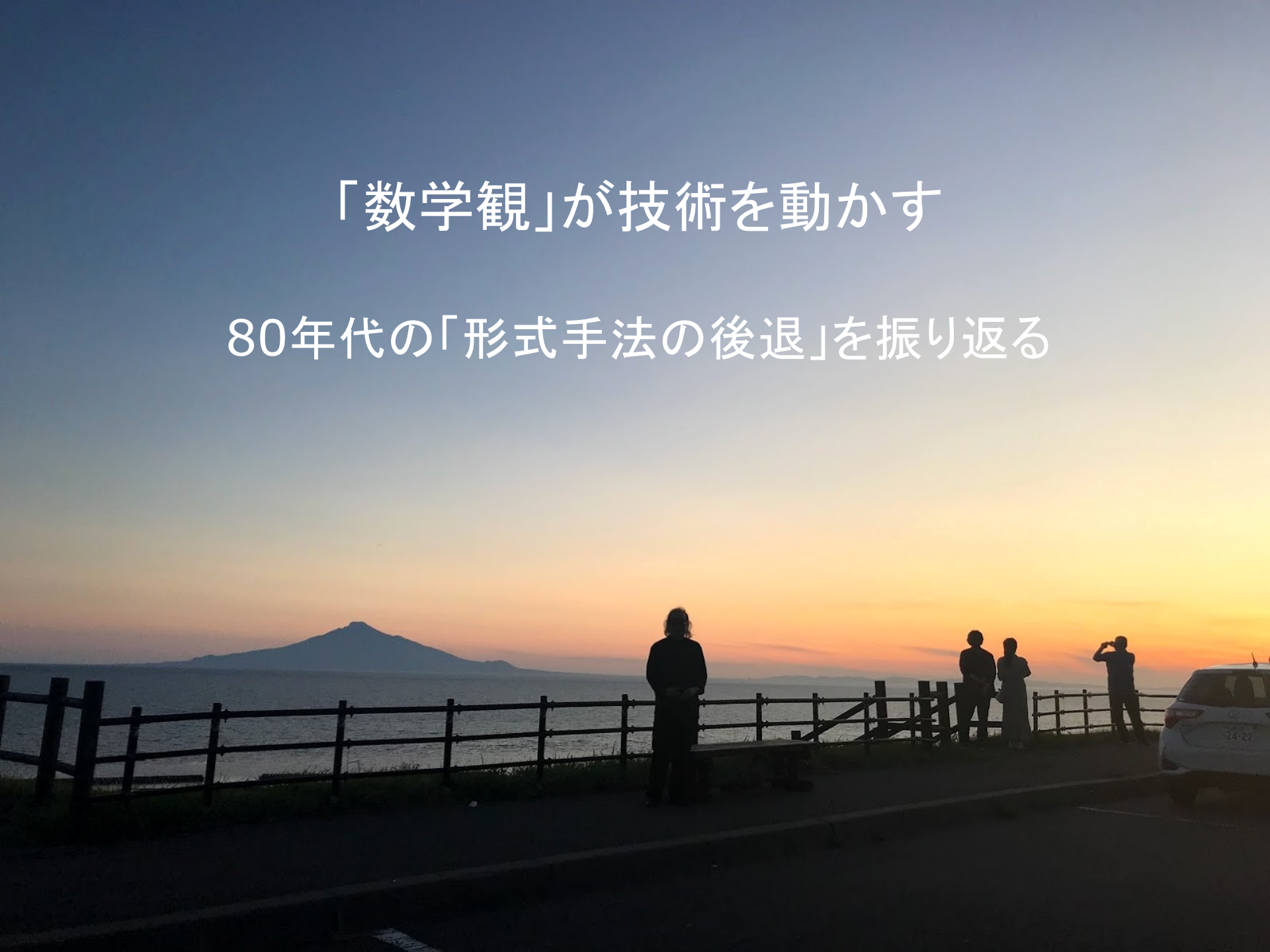

ラッセル
型の理論
ヒルベルト
プログラム
ゲーデル
不完全性定理
ゲーデル
連続体仮説
無矛盾性証明
コーエン
連続体仮説
独立性証明
グロタンティック
Scheme / Topos
ローヴェール
Topos理論
ヴォエヴォドスキー
Univalant
Foundation
00
10
20
30
40
50
60
70
80
90
00
10
20
チューリング
停止問題
チャーチ
ラムダ計算
チャーチ
型付きラムダ計算
カリー＝ハワード対応
マーティン・レフ
Dependent
Type Theory
ヴォエヴォドスキー
Homotopy
Type Theory

数学の基礎と
型の理論


ヒルベルト
プログラム
ゲーデル
連続体仮説
無矛盾性証明
グロタンティック
Scheme / Topos
ヴォエヴォドスキー
Univalant
Foundation
ラッセル
型の理論
ゲーデル
不完全性定理
コーエン
連続体仮説
独立性証明
ローヴェール
Topos理論
00
10
20
30
40
50
60
70
80
90
00
10
20
チューリング
停止問題
チャーチ
型付きラムダ計算
チャーチ
ラムダ計算
マーティン・レフ
Dependent
Type Theory
ヴォエヴォドスキー
Homotopy
Type Theory
カリー＝ハワード対応
数学の基礎と
型の理論と
計算機科学
LISP
Prolog
GHC
Haskel
ML
OCaml
COQ
Agda
ロビンソン
Resolution


ヒルベルト
プログラム
ゲーデル
連続体仮説
無矛盾性証明
グロタンティック
Scheme / Topos
ヴォエヴォドスキー
Univalant
Foundation
ラッセル
型の理論
ゲーデル
不完全性定理
コーエン
連続体仮説
独立性証明
ローヴェール
Topos理論
00
10
20
30
40
50
60
70
80
90
00
10
20
チューリング
停止問題
チャーチ
ラムダ計算
チャーチ
型付きラムダ計算
マーティン・レフ
Dependent
Type Theory
ヴォエヴォドスキー
Homotopy
Type Theory
カリー＝ハワード対応
数学の基礎と
型の理論と
計算機科学
LISP
Prolog
GHC
Haskel
ML
OCaml
COQ
Agda
ロビンソン
Resolution


ヒルベルト
プログラム
ゲーデル
連続体仮説
無矛盾性証明
グロタンティック
Scheme / Topos
ヴォエヴォドスキー
Univalant
Foundation
ラッセル
型の理論
ゲーデル
不完全性定理
コーエン
連続体仮説
独立性証明
ローヴェール
Topos理論
00
10
20
30
40
50
60
70
80
90
00
10
20
チューリング
停止問題
チャーチ
ラムダ計算
チャーチ
型付きラムダ計算
マーティン・レフ
Dependent
Type Theory
ヴォエヴォドスキー
Homotopy
Type Theory
カリー＝ハワード対応
LISP
Prolog
GHC
Haskel
ML
OCaml
COQ
Agda
ロビンソン
Resolution
数学の基礎と
型の理論と
計算機科学の
Time Lag


「数学観」が技術を動かす
80年代の「形式手法の後退」を振り返る

20世紀の数学と計算機科学の接点

- 「形式手法」と「従属型理論」

数学と計算機科学の接点を探る動き
「形式手法」と「従属型理論」

コンピュータが登場しその利用が拡大する中で、数学と計算機科学の接点を探ろうという研究は活発になります。

1970年代には、二つの研究の流れが独立に生まれます。

一つは、計算機科学の側から生まれた、具体的なプログラムの分析を通じて、計算機科学の数学的基礎づけを目指す動きです。「形式手法」と呼ばれています。

もう一つは、論理学＝数学の側から生まれた、新しい「型の理論」である「従属的型理論」を構築して、プログラムとその実行の数学的特徴を捉えようとする動きです。

「プログラムと論理」という問題意識

前者の「形式手法」の代表はホーアやダイクストラです。彼らは、代入文やif文、シーケンシャルな実行や繰り返しのWhile文といった具体的なプログラムの構成と論理との関係を研究しました。

後者の「従属的型理論」の代表は、ハワードやマーチン・レフです。彼らは、「型の理論」を作り上げ、抽象的ですが、プログラムの実行は、ある定理の証明に他ならないという認識にたどり着きました。

両者のアプローチは異なっていましたが、「プログラムと論理」という問題意識は共通のものでした。

当時のアプローチの限界

ただ、 20世紀の段階では、「プログラムと論理」という問題を扱うには、両者共にいろいろな困難がありました。

論理学＝数学からのアプローチである「従属型理論」は、現実のコンピュータ技術や具体的なプログラム言語との接点は明確ではありませんでした。そのスキマを埋めるのには、実際には、前回見たように 30年から40年の時間が必要でした。

計算機科学からのアプローチでは、論理の多様な性質への理解を欠いていました。実践的には、彼らの理論を実行する強力な「証明マシン」は存在しませんでした。

「形式手法」登場と挫折の背景を考える

今回のセッションでは、ホーア、ダイクストラ、ランポートらの「形式手法」の登場と挫折を取り上げます。

興味深いのは、このムーブメントが挫折した背景には、IT技術の基礎に数学を置くべきだという彼等の主張が、容易には周囲に受け入れられなかったことがありました。

日常の生活と数学を遠ざける考え方は、ある意味で当たり前のように広く深く我々の考え方に染み込んでいます。ITの世界も、その例外ではないということだと思います。

計算機科学での形式手法の登場

An axiomatic basis for computer programming

C.A.R Hoare

1969年

https://www.cs.cmu.edu/~crary/819-f09/Hoare69.pdf

Abstract

本論文では、幾何学の研究で最初に適用され、後に数学の他の分野に拡張された技法を用いて、コンピュータ・プログラミングの論理的基礎を探求する試みがなされている。

これは、コンピュータ・プログラムの特性の証明に使用できる公理と推論のルールのセットを解明するものである。このような公理とルールの例を挙げ、簡単な定理の正式な証明を表示する。

最後に、これらのテーマを追求することで、理論的にも実用的にも重要な利点が得られる可能性があることを論じる。

Motivation

コンピュータ・プログラミングは、正確な科学である。その中では、プログラムの全ての性質と、どんな環境の下でそれが実行されたとしても、その実行の結果は、原理的には、プログラム自身のテキストから、純粋に演繹的推論の手段で見つけ出すことができる。

Hoare triple

Empty statement axiom schema

$$\frac{}{\{P\}\mathbf{skip}\{P\}}$$

Assignment axiom schema

$$\frac{}{\{P[E/x]\}x := E\{P\}}$$

Rule of composition

$$\frac{\{P\}S\{Q\} \quad , \quad \{Q\}T\{R\}}{\{P\}S;T\{R\}}$$

Hoare triple

Conditional rule

$$\frac{\{B \wedge P\}S\{Q\} \quad , \quad \{\neg B \wedge P\}T\{Q\}}{\{P\}\textbf{if } B \textbf{ then } S \textbf{ else } T \textbf{ endif}\{Q\}}$$

Consequence rule

$$\frac{P_1 \to P_2 \quad , \quad \{P_2\}S\{Q_2\} \quad , \quad Q_2 \to Q_1}{\{P_1\}S\{Q_1\}}$$

While rule

$$\frac{\{P \wedge B\}S\{P\}}{\{P\}\textbf{while } B \textbf{ do } S \textbf{ done}\{\neg B \wedge P\}}$$

Conclusion

単純ではないプログラムに証明を与えようとする実践は、かなり強力な証明技術が利用可能になるまでは、広く広がることはないだろう。そうなったとしても、それは簡単ではないだろう。

しかし、プログラム証明の実践的な利点は、プログラミングのエラーによるコストの増大という視点から見て、結局のところ、これらの困難を圧倒するだろう。

Guarded Commands, Nondeterminacy and Formal Derivation of Programs

Edsger W. Dijkstra

1975年

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.90.97&rep=rep1&type=pdf

guarded commands

So-called "guarded commands" are introduced as a building block for alternative and repetitive constructs that allow nondeterministic program components for which at least the activity evokt, but possibly even the final state, is not necessarily uniqilely determined by the initial state. For the formal derivation of programs expressed in terms of these constructs, a calculus will be be shown.

Weakest Pre-conditions

The way in which we use predicates (as a tool for defining sets of initial or final states) for the definition of the semantics of programming language constructs has been directly inspired by Hoare [1], the main difference being that we have tightened things up a bit: while Hoare introduces sufficient pre-conditions such that the mechanisms will not produce the wrong result (but may fail to terminate), we shall introduce necessary and sufficient--i.e, so-called "weakest"--pre-conditions such that the mechanisms are guaranteed to produce the right result.

Predicate transformer

More specifically: we shall use the notation **wp(S, R),** where S denotes a statement list and R some condition on the state of the system, to denote the weakest precondition for the initial state of the system such that activation of S is guaranteed to lead to a properly terminating activity leaving the system in a final state satisfying the post-condition R. Such a wp--which is called "a predicate transformer" because it associates a pre-condition to any post-condition R

計算機科学での形式手法の基本的認識

数学は、人間の厳密な推論を可能にする最良のアプローチとして、二千年以上にわたって発展してきました。

数十年にわたる疑似プログラミング言語の設計は、その優位性を脅かすものではありません。

数学的に推論する最善の方法は数学を使うことであり、疑似プログラミング言語を使うことではありません。

-- Lamport

網羅的なテストが不可能な時には、ほとんどの場合はそうなのだが、我々の信頼は、(それが機械的なものであろうと、そうでなかろうと)証明の上にのみ基礎付けられる。

-- Dijkstra

証明が与えられていれば、それによって正当化されるプログラムを導くことは、プログラムが与えられていて、それを正当化する証明を構成するより、はるかに容易である。

-- Dijkstra

ライプニッツの夢の一部が現実のものとなりました。

その変化の大部分が数学ではなく、むしろ計算科学の分野で起こったことは、非常に理解しやすいことです。...

....

つまり、計算の世界はライプニッツの家となり、それが私の家でもあったことは幸運でした。

-- Dijkstra

私たちの推論を正す唯一の方法は、数学者の推論のようにそれを具体的に目に見える形で扱えるものにものにすることです。そうすれば、一目で自分の間違いを見つけることができます。

人の間で論争が起きたときには、簡単にこう言うことができます。「難しい話は抜きにして、どちらが正しいか、計算してみて」と言えばよいのです。

-- *Leibniz*

暗雲

ACM論文誌での「形式手法」批判

形式的手法への一方的非難

1979年、ACMは、その論文誌CACMに、形式的手法を一方的に非難する次のような内容の論文を掲載する。

「この論文では、プログラムの形式的検証は、たとえそれが得られたとしても、計算機科学とソフトウェア・エンジニアリングにおいては、数学での証明と同じ役割は果たす事はないと議論される。更に言えば、連続性の欠如、変化の不可避性、決定的に多くの現実のプログラムの仕様の複雑さは、形式的検証の過程を、正当化することも管理することも困難にする形式的検証の使いやすさは、プログラム言語の設計に大きな影響力を持つことはないように感じられる。」

https://www.cs.umd.edu/~gasarch/BLOGPAPERS/social.pdf

Social Processes and Proofs of Theorems and Programs

Richard A. De Millo et al.

1979年

https://www.cs.umd.edu/~gasarch/BLOGPAPERS/social.pdf

ランポートの抗議

この論文、“Social Processes and Proofs of Theorems and Programs” De Millo, Lipton and Perils https://www.cs.umd.edu/~gasarch/BLOGPAPERS/social.pdf は、「プログラムの検証なんて、必ず失敗する。"program verification is bound to fail. "」と断じたのだ。

ランポートは、直ちにACMの編集部に、抗議のメールを送る。"Letter to the Editor
" https://www.microsoft.com/…/publicati…/letter-to-the-editor/

反「形式手法」論者の数学観

数学では、証明の目的は、ある定理の正しさに対する人の確信を増大させることである。確かに、数学者がこの目的のための道具の一つとして、理論的に利用することができるのは、形式論理の長い連鎖である。

しかし、事実はそうではない。彼らが使う証明は、全く別の生き物なのだ。証明は、問題を解決さえしない。

証明は、その名が示唆するものとは逆に、確信に向かう一つのステップにすぎないのだ。結局のところ、数学者がある定理に確信を持つか否かを決定するのは、社会的プロセスなのだと我々は信じている。

「社会的なプロセス」が数学者の確信を決める

彼らの論文のタイトルに、「社会的なプロセス」とあったのは、こういう意味だったのだ。

「プログラムの検証者たちの間に、数学と比較できるような社会的プロセスが存在しないのだから、プログラムの検証は、失敗するように定められていると、我々は信じている。プログラムについての誰かの確信に、証明がどのように影響を与えるのか、我々は見ることができないのだ。」

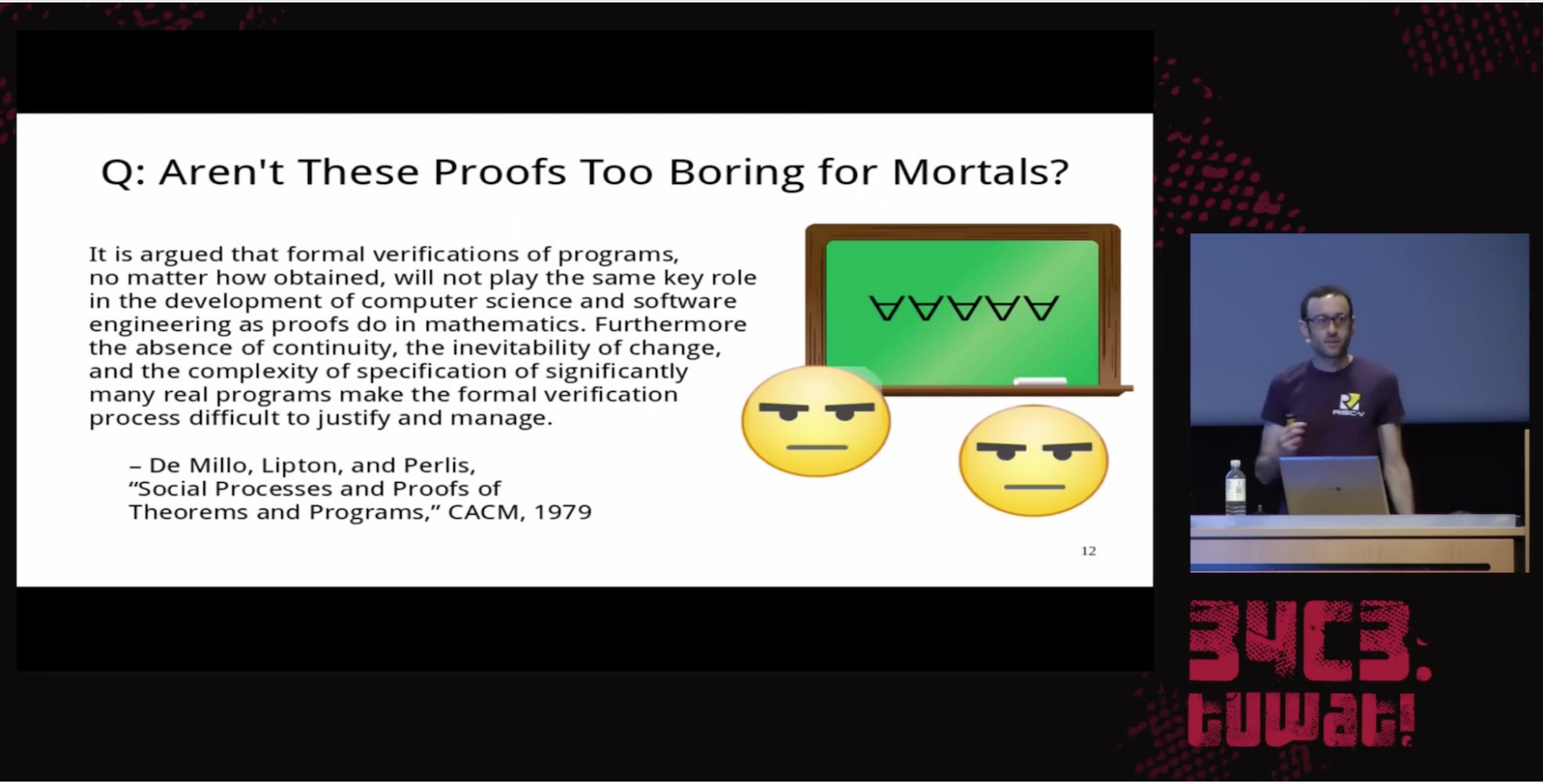
（僕の理解では、たいていの数学者にとって、「定理に対する確信」は、まず、最初に直観的に生まれるものだ。だから、それを証明しようとする。それに、数学者は、IT業界の人のように、コミュニティや学会に集まって「社会的プロセス」での合意をつくるのが好きなようには見えないのだが。）

愚論でも影響力はある

数学論的には、ヒルベルトもゲーデルも仰天するような全くの愚論である。

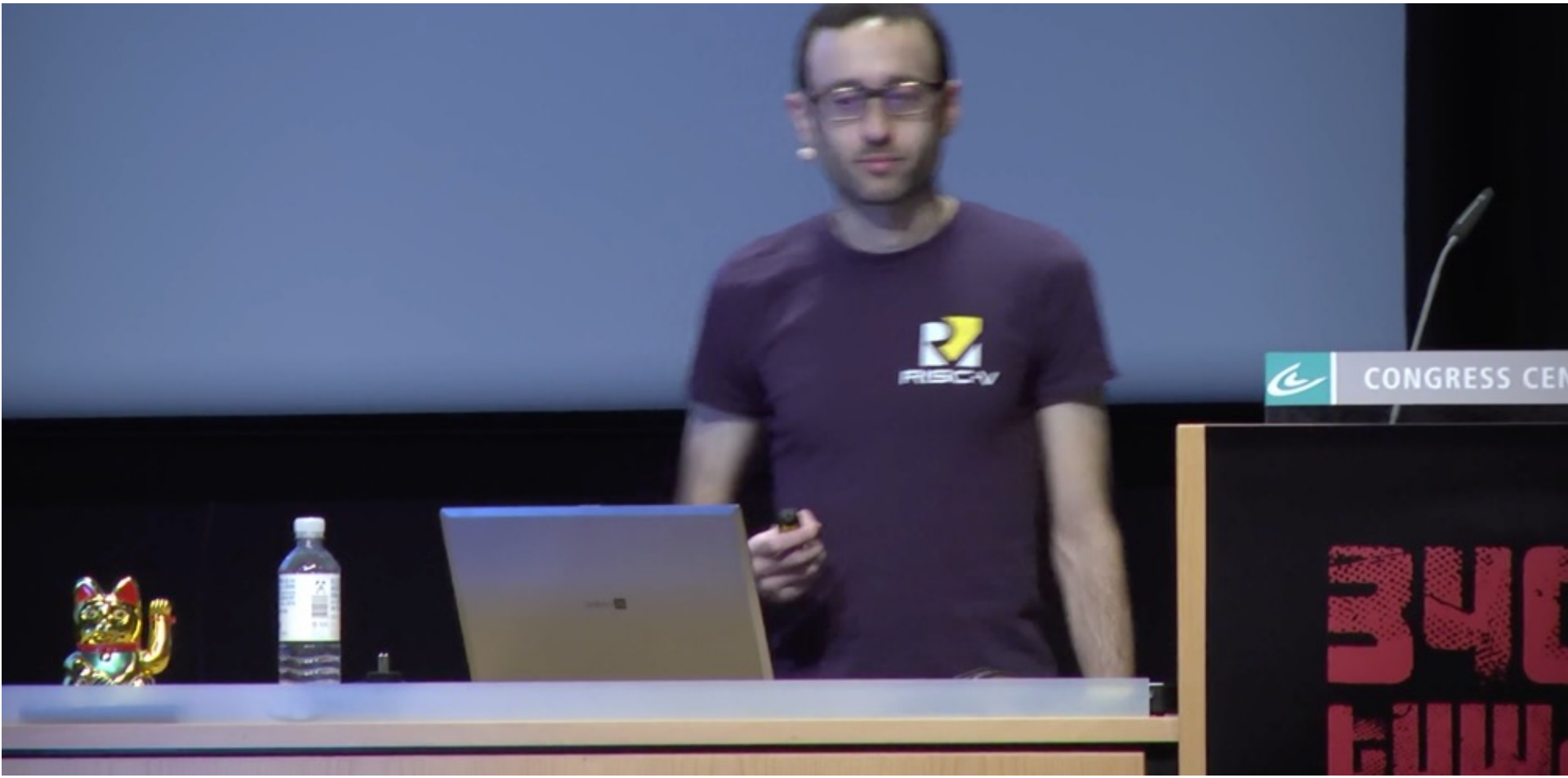
ACMは、計算機科学ではもっとも「権威ある」学会の一つなのだが、そこで、こうした議論が行われていたことには注意が必要だと思う。

なぜなら、それは現実のITの世界の多数の住人の心情と「数学観」を代弁している可能性があるからである。



40年近く経った今も、この論文は「有名」である。もちろん「悪名」が高いという意味だが。写真は、21世紀に「形式手法」を復活させるプロジェクト Deep Specificationのカンファレンスで、この論文を「紹介」する、チッパラ。

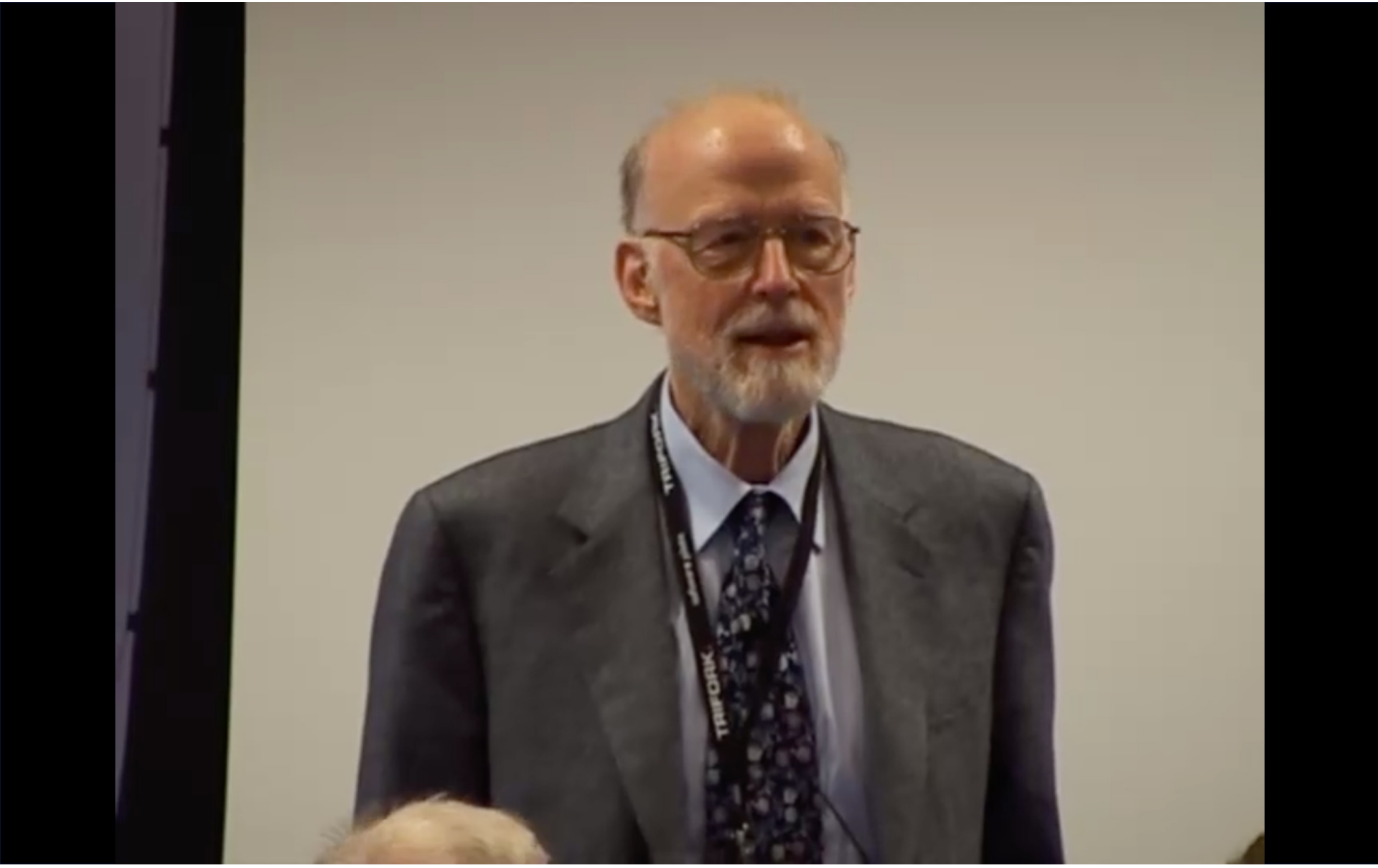
"Coming Soon: Machine-Checked Mathematical Proofs in Everyday Software and Hardware Developme"



Adam Chlipala http://bit.ly/2GppRxe

20世紀の形式手法の後退

ホーアとダイクストラ



http://bit.ly/2oXpgNz

Null References: The Billion Dollar Mistake

Tony Hoare

How did software get so reliable without proof ?

C.A.R Hoare

1996年

https://www.gwern.net/docs/math/1996-hoare.pdf

ソフトウェアは証明なしでも、信頼できるものになった

「最近のマッキンゼーの分析は、コンピュータに依存したことに起因する可能性もあると、これまで考えられていた数千人の死亡事故のうち、ソフトウェアのエラーで説明できる死者は、10人程度にすぎないことを明らかにした。」

その上で、ホーアは、ソフトウェアが「こんなにも信頼できる」ようになった要因を列挙する。

- 管理手法の改革
- テスト
- デバッグ
- 多くの技術の投入
- プログラミングの方法論

それぞれの展開は、説得力もあり、なかなか興味ふかいものだ。

ただ、それは現在のソフトウェア・エンジニアリングのスタイルの基本が、こうして30年近く前には、「形式手法」の後退の中で形成されていたことを示すものだ。

30年も経てば、いろんなものが変わっていく。

現在の我々は、再び、「ソフトウェアは安全なのか？」と言う問いかけに、あるいは、チッパラの言う「開発者が、テスト・デバッグ・コードレビューを繰り返すことで、バグのないプログラムが作れるのか？」という問いかけの前に立たされているのだ。

ボーイング737MAX墜落事故

- 「2018年10月29日、インドネシアのジャワ海でライオン・エア610便（ボーイング737MAX 8型機、PK-LQP）が離陸後約10分で墜落、乗客乗員189名全員が死亡した。
  飛行データの解析によれば、2つあるAoAセンサーのうち1つが故障により誤った角度を示していた。これにより、MCASが機首が上がりすぎていると判断して下降操作を行い、上昇操作を行っていたパイロットとせめぎ合いになる形で墜落した。」
- 「2019年3月10日、エチオピアのアディスアベバ、ボレ国際空港より離陸したエチオピア航空302便（ボーイング737MAX 8型機、ET-AVJ）が、離陸後約6分で墜落し、乗客乗員157名全員が死亡した。」

http://bit.ly/2lHdbKO

ダイクストラのメッセージ

ダイクストラの最後のメッセージ

ダイクストラは、21世紀の初めまで生き、2002年になくなる。

最晩年の2001年に彼があらわした「ライプニッツの夢の魔力の下で」”Under the spell of Leibniz‘s Dream” は、自身の生涯を振り返りつつ、コンピュータ・サイエンスについての彼の立場を語った、いわば彼の「遺言」ともいえる文書である。

https://www.cs.utexas.edu/…/transcript…/EWD12xx/EWD1298.html

ダイクストラの論点は多岐にわたるが、いずれも刺激的である。

- コンピュータ・サイエンスでの「理論」と「実践」の関係
- コンピュータ・サイエンスとコンピュータ企業の関係
- 大学の役割とその変化
- プログラミングは「誰でもできる」のか？
- ……

この文書が「刺激的」なのは、我々多数の「通念」とは、ほとんど逆のことを彼が述べているからである。おそらく反発する人も多いだろう。

特に、今回のセッションのテーマである「計算機科学と数学の接点」の問題を考える時、彼の問題提起は重要である。

こうした主張がコンピュータの世界に存在していることを知ることは、「未来の変化」を展望した時に、大きな意味があると僕は考えている。なぜなら、未来は現在の連続的延長上にあるとは限らず、変化は、現在の「常識」のいくつかを覆すだろうから。

文章は平易である。論旨も明確である。興味ある人は、是非、目を通していただければと思う。

参考資料

今回のセッションは、以前行ったDeep Specificationをテーマにしたゼミナール「プログラムと論理」の一部をまとめたものである。

次のページを参照さ れたい。
「プログラムと論理」
https://www.marulabo.net/docs/programlogic/

講演資料は、次からダウンロードできる。
https://drive.google.com/file/d/1W6yflIL1ajVPVlPDz2EkRmk-WBywTm6y/view?usp=sharing


「Curry-Howard対応」の発見

「Curry-Howard対応」の発見と「従属型理論」の成立

ゲーデル、チャーチ、チューリングの「証明可能性」は「計算可能性」に等しいという発見から40年後の1970年代、論理学・数学の分野で、重要な動きがあった。

ハワードの「Curry-Howard対応」の（再）発見と、それに基づくマーティン・レフの「従属型理論」の成立である。

時代背景

1940年代のChurchの仕事から、ラムダ計算に対する関心が、ふたたび活発化するのは、20年近くたった1960年代からである。

理由ははっきりしている。コンピュータが現実に動き出したからである。

このセッションでは、「Curry-Howard対応」について述べる。

Curryの発見
型付きラムダ計算と直観主義論理との対応

既に1934年に、Haskell Curryは、型付きラムダ計算と直観主義論理とのあいだに、対応関係があることを発見していた。

型を持つラムダ計算の関数の型を示す矢印 → と、
論理式“A → B” の中に出てくる含意を意味する矢印 → との間に、対応関係があるというのだ。

Curry, Haskell (1934), "Functionality in Combinatory Logic”
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1076489/pdf/pnas01751-0022.pdf

Curryが気づいたこと
次の二つのルールの形が似ているということ

論理式の推論ルール

$$\frac{A \to B \qquad A}{B}$$

A → B が成り立ち、
Aが成り立つなら
Bが成り立つ

関数の適用の型のルール

$$\frac{f\colon A \to B \qquad a\colon A}{f\ a\colon B}$$

f が A → B という型を持ち、
a がAという型を持つなら
f a は型 B を持つ

論理式の含意を意味する矢印 → と関数の型を示す矢印 →
の間に、対応関係があるということ。

Howardによる発展
論理式はラムダ計算の型と見なすことが出来る

Howardは、このCurryの発見を、さらに深く考えた。

1969年の彼の論文のタイトルが示すように、
論理式は型付きラムダ計算の型と見なすことが出来るのである。

ただし、彼のこの論文が公開されたのは、1980年になってからのことらしい。

Howard,Williams (1980)
"The formulae-as-types notion of construction”
http://www.cs.cmu.edu/~crary/819-f09/Howard80.pdf

Howardが考えたこと
Proposition as Type

論理式 A → B が、A → B という関数と同じ型を持つと考えられるとしたら、他の論理式も、型を持つと考えることができるのでは？

例えば、論理式 A ∨ B は、型 A ∨ B を、論理式 A ∧ B は、型 A ∧ B を持つと考えることはできないか？

そして、この論理式（命題：Proposition）を型とみなすアイデアは、うまく機能するのである。こうした考えを、

Proposition as Type

と呼ぶ。

命題Pとその証明p を p：Pで表す
Proof as Term

彼らは、この時、型Pの項pにあたるものを、命題Pの証明と考えることができることに気づく。それは、命題の証明が何から構成されるかについて、次のような自然な解釈を与える。

- **p: A ∧ B**　　pは、Aの証明とBの証明の両方
- **p: A ∨ B**　　pは、Aの証明、あるいは、Bの証明
- **p: A → B**　　pは、Aの証明からBの証明を導く方法
- **p: (∀x)B(x)**　　pは、任意のaに対してB(a)の証明を与える方法
- **p: (∃x)B(x)**　　pは、あるaに対するB(a)の証明

こうした考えを、

Proof as Term

と呼ぶ。

Curry-Howard対応

「型」と「命題」、「項」と「証明」との「双対性」

Curry-Howard対応の、中心的な概念は、「型」と「命題」、「項」と「証明」との「双対性」である。

- 「*p* が命題 *P* の証明である」ことを、***p : P***
  と表すことができる。
- 「*p* は、型 *P* の項（要素）である」ことも、***p : P***
  と表すことができる。

「型」と「命題」、「項」と「証明」との「双対性」

Curry-Howard対応の、中心的な概念は、「型」と「命題」、「項」と「証明」との「双対性」である。

- 「*p* が命題 *P* の証明である」ことを、***p : P***
  と表すことができる。
- 「*p* は、型 *P* の項（要素）である」ことも、***p : P***
  と表すことができる。

「*p* が命題 *P* の証明である」という判断を表す ***p : P*** は、
「*p* は、型 *P* の項（要素）である」という判断を表す ***p : P*** と
見なすことが出来るし、また、逆の見方も出来るのである。

Curry-Howard対応

「型」と「命題」、「項」と「証明」との「双対性」

$$p : P$$

項（要素）　　型

「p は、型 P の項（要素）である」

Curry-Howard対応

「型」と「命題」、「要素」と「証明」との「双対性」

「*p* は、命題 *P* の証明である」

証明　　命題

$$p : P$$

Curry-Howard対応

「型」と「命題」、「項」と「証明」との「双対性」

「p は、命題 P の証明である」

証明　　命題

$$p : P$$

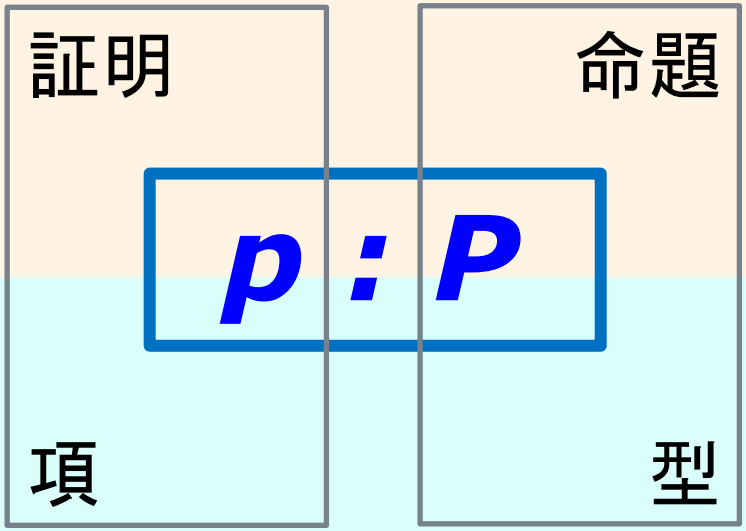
項（要素）　　型

「p は、型 P の項（要素）である」

Curry-Howard対応

「型」と「命題」、「項」と「証明」との「双対性」

「p は、命題 P の証明である」

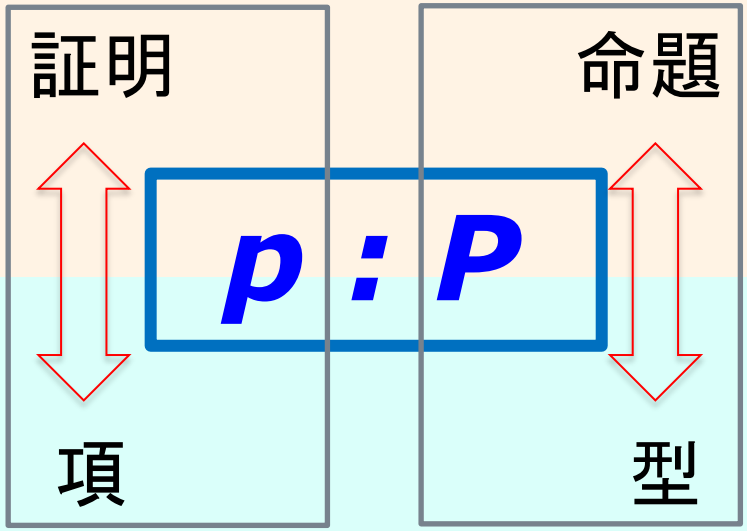


「p は、型 P の項である」

Curry-Howard対応

「型」と「命題」、「項」と「証明」との「双対性」

「p は、命題 P の証明である」

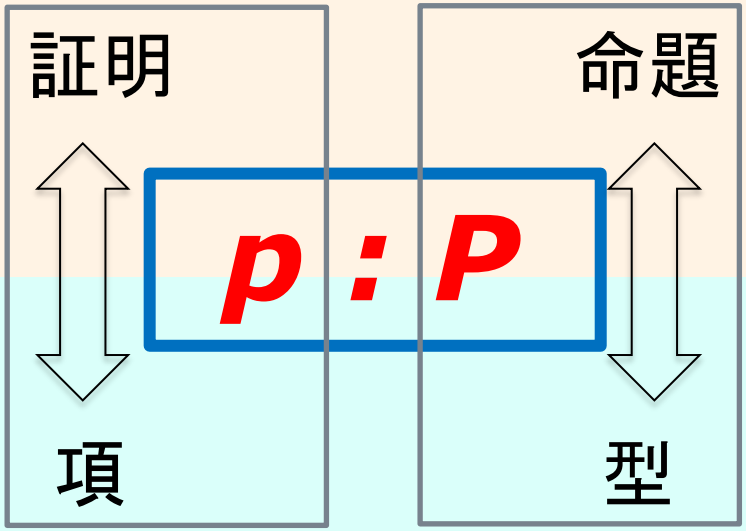


「p は、型 P の要素である」

Curry-Howard対応

「型」と「命題」、「項」と「証明」との「双対性」

「*p* は、命題 ***P*** の証明である」



証明は
項である

命題は
型である

「*p* は、型 ***P*** の項である」

“Propositions as Types”
“Proofs as Terms”

Howardの洞察は、”Propositions as Types”, “Proofs as Terms” (これを、PATという)として、あとにみるMartin-Löfの型の理論に大きな影響を与えた。

こうした観点は、今日のCoq等の証明支援言語の理論的基礎になっている。

「証明」＝「プログラム」
Proof as Program

重要なことは、このCurry-Howard対応は、項の計算をプログラムと見なせば、”Proof as Program”としても解釈出来るということである。

いまから 40年近く前に、論理学者と計算機学者の一部は、「証明」＝「プログラム」という認識にたどり着いていた。

Proof as Program

40年のタイムラグ

- 40年に一度の大発見でも、それが他の分野に伝わるには、10年単位の時間がかかることがある。論理学・数学の世界での新しい認識がコンピュータの世界に広い影響を与えるまでには、一定の時間がかかる。

- 30年代のチャーチの「型なしラムダ計算」は、LISP (1958)の理論的基礎。同じチャーチの「型付きラムダ計算」を関数型言語Haskell(1985)は基礎にしている。「従属型理論」に基づくCoqの開発が始まったのは、1984年である。

- ちなみに、マーティン・レフの理論から40年近く経ってからHomotopy Type Theoryで論理学の世界を一新した数学者のVoevdskyは、たまたまコンピュータ実習室で学生の演習を見ていて、「従属型理論」に出会ったという、

従属型理論の登場

Dependent Type Theoryの登場とその影響

Dependent Type Theory
Martin-Löf

Dependent Type Theory
「従属型理論」の基本が出来上がるのは、1980年代である。

その中心人物は、Martin-Löfである。

Dependent Type Theoryが与えた影響
「証明支援システム」の実現

現在のCoq, Agda等の多くの証明支援システムは、彼のDependent Typeの理論に基礎付けられている。

証明支援システムは “Proof as Program” という、証明をコンピュータのプログラムとして実行するというアイデアを、現実に実行可能にしたものである。

証明支援システムは、また、人間と機械の「協働」という、新しい協力関係を始めて可能にしたものである。

(GitHub CoPilotが初めてではないのだ。)

Dependent Type Theoryが与えた影響
Voevodskyの新しい型の理論を準備する

Dependent Type Theory はまた、現代の新しい型の理論である VoevodskyのHomotopy Type theory の直接の先行者であり、その登場を準備した。

Dependent Type Theory

命題への型の拡張

Churchの単純な型を持つラムダ計算の体系は、
基本的には、型A, 型Bに対して、型 A → Bで表現される関数型の型しか持たない。

それに対して、Martin-Löfは、論理的な命題にも、自然なスタイルで型を定義した。例えば、Aが型であり、Bが型であれば、A∧BもA→BもA∨Bも型であるというように。もちろん、それぞれは異なる型である。

ある aが型Aを持つ時、***a : A*** で表す。

Dependent Typeとは何か

これまで、A ∧ B, A → B, A ∨ B といった、元になる型 A, B を指定すると新しい型が決まるといったスタイルで型を導入してきた。

Dependent Type Theory では、これとは異なる次のようなスタイル、型Aそのものではなく型Aに属する要素 a：Aに応じて新しい型 B(a)が変化するような型の導入が可能となる。

これを a：Aに依存・従属する型という意味で
「従属型 Dependent Type」と呼ぶ。

Dependent Typeの例

例えば、実数R上のn次元のベクトル Vec(R,n)とn+1次元のベクトルVec(R,n+1)は、別の型を持つのだが、これは nに応じて型が変わると考えることが出来る。

先の例では、ベクトルの次元nが変わったが、nを固定して、n次元のベクトルを、自然数 N、整数 Z、実数 R、複素数 C上でも定義出来る。

これらは、Dependent Typeとして表現できる。

Dependent Typeの表記 1
$\Pi\ type$

Dependent Typeは、例えば、ある型Aの値aにディペンドして変化する型である。こうした型を次のように表す。

$$\Pi(x : A).\ B(x)$$

あるいは

$$\prod_{x:A} B(x)$$

先の次元が変化するのベクトルの型は、次のように表される。

Π(x:N).Vec(R,x)

これは、全てのnについてVec(R,n)を考えることに対応している。型の理論では、全称記号は、Dependent Typeとして導入される。

もう一つの例は、{ N, Z, R, C }：T として、次のようなDependent Typeで表される。

Π(x：T). Vec(x,n)

Inductive Type

- 型の理論では、基本的な定数と関数から、新しい型を帰納的に定義することが出来る。こうした型をInductive Type（帰納的型）と呼ぶ。
- 次は、そうした帰納的な定義の例である。
  ここでは、自然数の型natが、ゼロを意味する定数Oと「その次の数」を表す関数Sで、帰納的に定義されている。

Coqでの「自然数」 nat の定義

```
Inductive nat : Type :=
  | O : nat
  | S : nat -> nat.
```

Sは、natからnatへの関数である
直感で気には、S(n)=n+1 である

Inductive Type nat (自然数)

0. O
1. S(O)
2. S(S(O))
3. S(S(S(O)))
4. S(S(S(S(O))))
5. S(S(S(S(S(O)))))
6. S(S(S(S(S(S(O))))))

……

……

n. S(S(S(S(S(S(S(…S(O)…))))))

n個のS

同一性への型の付与

- Martin-Löf は、式 a = b にも型を与える。
  $a =_A b : Id(A\ a\ b)$
  型 Id(A a b) を持つ式は、a と b は等しいという意味を持つ。

- $a =_A b$ のAは、型Aの中での同一性であることを表している。すなわち、a：A で b：Aで、かつa = bの時にはじめて、$a =_A b$ は型Id(A a b)
  を持つ。

- ここでは式の同一性について述べたが、式の同一性と型の同一性は、異なるものである。

型の理論の記述

Martin-Löfの型の理論は、次のことを示す、四つの形式で記述される。

1. ある対象aが、型であること
   $a : Type$
2. ある表現式 aが、型 αの要素であること
   $a : \alpha$
3. 二つの表現式が、同じ型の内部で、等しいこと
   $a = b : \alpha$
4. 二つの型が、等しいこと
   $\alpha = \beta \ : Type$

証明とは何か？

Kolmogorov
a → b (aならばb)の証明の解釈

Kolmogorovは、命題 a → b (aならばb)の証明に、「命題 aの証明を命題 bの証明に変換する方法を構築すること」という解釈を与えた。

このことは、a → bの証明を、aの証明からbの証明への関数と見ることが出来るということを意味する。

同様に、次に見るように、命題の意味を、その証明と関連づけることができる。

Kolmogorovのこの考え（構成主義という）は、Martin-Löfの型の理論に深い影響を与えた。

Kolmogorovの解釈の下では、命題の証明は、何から構成されるか？

- **⊥（偽）** 証明なし
- **A ∧ B** Aの証明とBの証明の両方
- **A ∨ B** Aの証明、あるいは、Bの証明
- **A → B** Aの証明からBの証明を導く方法
- **(∀x)B(x)** 任意のaに対してB(a)の証明を与える方法
- **(∃x)B(x)** あるaに対するB(a)の証明

Kolmogorovの解釈の下では、命題の証明は、何から構成されるか？形式的に

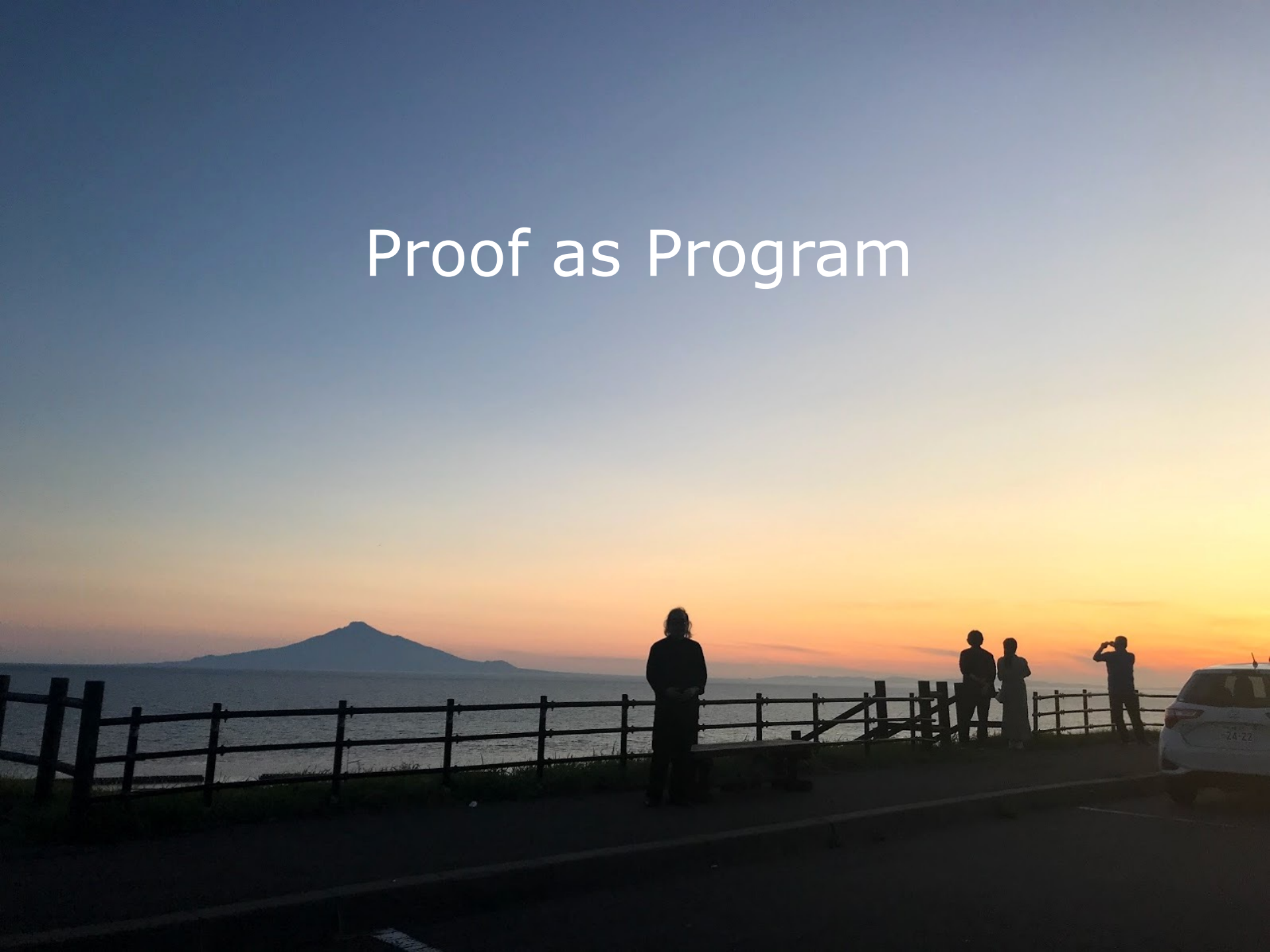
⊥（偽）	none
A ∧ B	Aの証明であるaと、 Bの証明であるbのペア**(a,b)**
A ∨ B	Aの証明である**i(a)**、あるいは、 Bの証明である**j(b)**
A → B	Aの証明であるaに対して、 Bの証明b(a)を与える**(λx)b(x)**
(∀x)B(x)	任意のaに対して Bの証明b(a)を与える**(λx)b(x)**
(∃x)B(x)	あるaと、B(a)の証明であるbのペア **(a,b)**

a : A の解釈

論理＝数学的には、**a : A** には、次のような様々な解釈がある。

1. 集合論: Russellの立場
   Aは集合であり、aはその要素である。$a \in A$
2. 構成主義: Kolmogorovの立場
   Aは問題であり、aはその解決である
3. PAT: Curry & Howardの立場
   Aは命題であり、aはその証明である。 $a : A$
4. 型の理論: Martin-Löf
   Aは型であり、aはその項である。 $a : A$
5. HoTT: Voevodskyの立場
   Aは空間であり、aはその点である。 $a : A$

Proof as Program



再び、計算と証明とプログラムについて

プログラム

計算

証明

機械の論理的＝数学的推論能力

先に、”Proof as Program プログラムとしての証明” という認識が、Curry-Howard対応の発見や従属型理論の登場の中で、生まれてきたことを述べた。

それは、論理的＝数学的証明は機械のプログラムで実行できるということ、ひるがえって言えば、機械は論理的＝数学的推論能力を持つという認識に他ならない。

この認識は、人工知能論にとって本質的に重要な認識であると、僕は考えている。

ここでは、改めて、このような認識がどのように形成されてきたのかを、歴史的・理論的に振り返りたいと思う。

「計算」と「証明」と「プログラム」

「計算」「証明」「プログラム」という三つの言葉は、もともと別の意味を持っている。それは今でも同じだ。そして それぞれがそれぞれの歴史を持っている。

この三つの言葉の中で、「計算」の起源が一番古い。4000年以上前の古代バビロニアでは、2の平方根の値も知っていたし、ピタゴラスの定理で直角三角形の斜辺の長さを計算する「計算表」も持っていた。

「証明」の概念の起源は、2300年前のユークリッドに帰してもいいと思う。

この中で、「プログラム」という項が一番新しい。その起源は、コンピュータが登場した1950年代を遡ることはないのは明らかだ。

バビロニアの数学　粘土板 YBC 7289

バビロニアでは、2の平方根の値を知っていた。

その近似値は60進法で4桁、
10進法では約6桁に相当する。
対角線に刻まれた数字は、
1, 24, 51, 10 である。

60進では、

$$1 + 24/60 + 51/60^2 + 10/60^3 = 1.41421296...$$

となる。

$$\sqrt{2} = 1.41421296...$$

バビロニアの数学　Plimpton 322

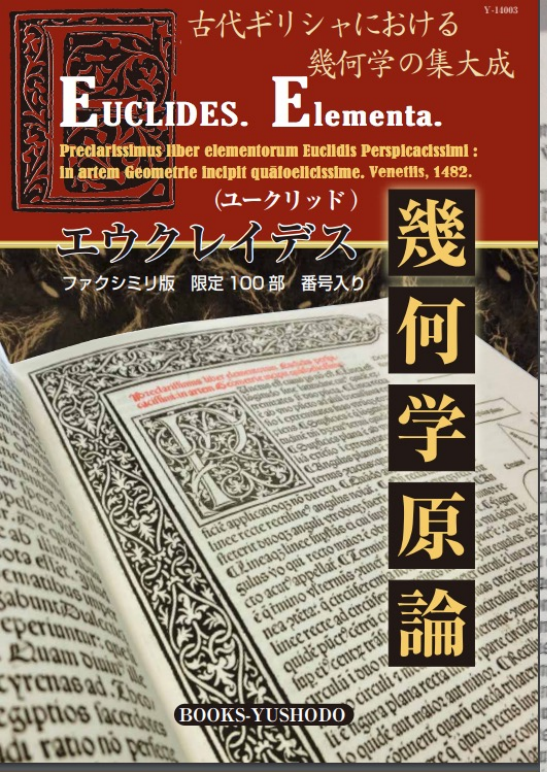
バビロニアでは、ピタゴラスの定理を知っていた。

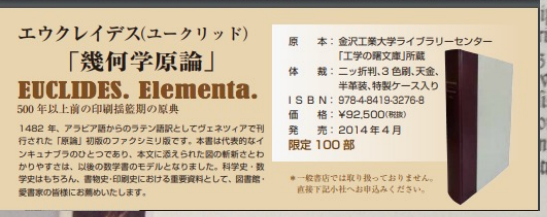
計算

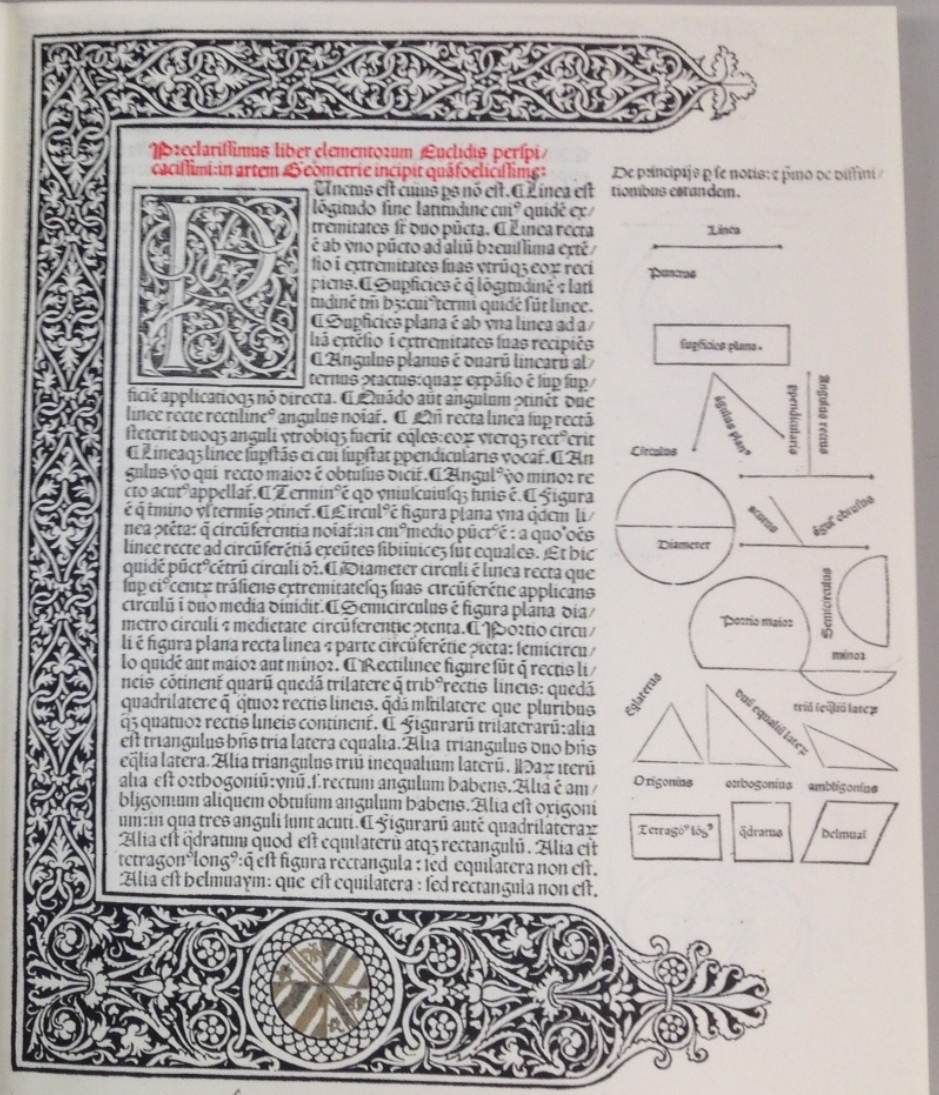
1822 BC ~ 1762 BC

証明

ラファエロの壁画「アテナイの学堂」から



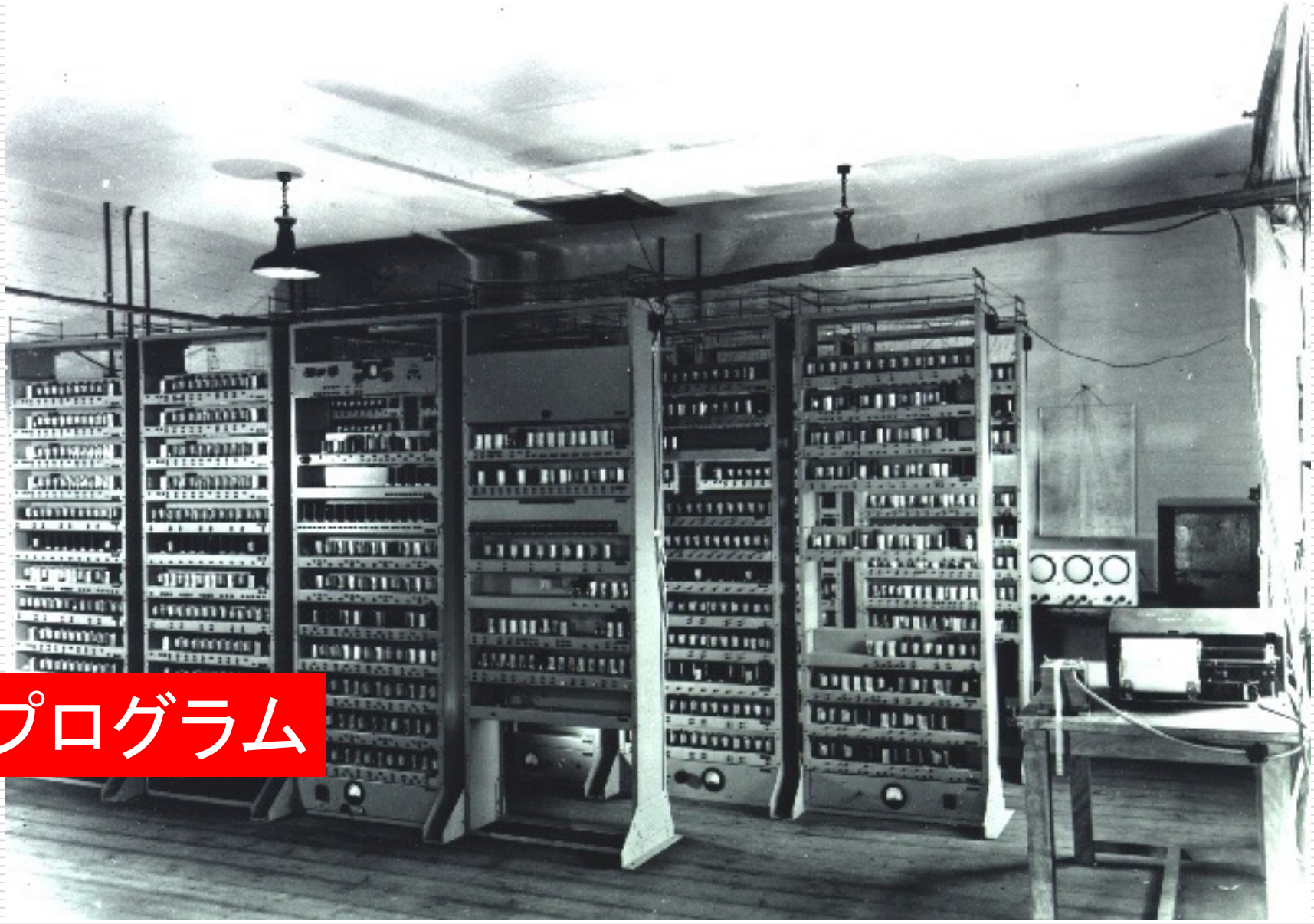




証明

1482 年、アラビア語からのラテン語訳としてヴェネツィアで刊行された「原論」初版のファクシミリ版

EDSAC (Electronic delay storage automatic calculator) 1949



プログラム

amazon web services™

Microsoft Azure

Kubernetes

κυβερνήτης: *Greek for "pilot" or "helmsman of a ship"*
the open source cluster manager from Google

プログラム

「計算」＝「証明」＝「プログラム」

ここでは、機械の能力を人間がどう認識してきたかを、この三つの概念の関係の認識の歴史をメイン・ストーリーにして述べたものである。

概略を述べておこう。

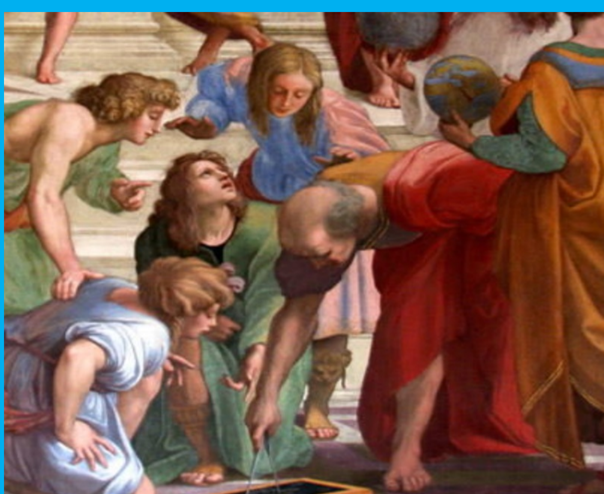
まず、「計算」と「証明」が、本質的には同じものであるという認識が生まれる。1930年代のことだ。

それから40年ほど経って、「プログラム」は「証明」とみなせるという認識が生まれる。1970年代のことだ。

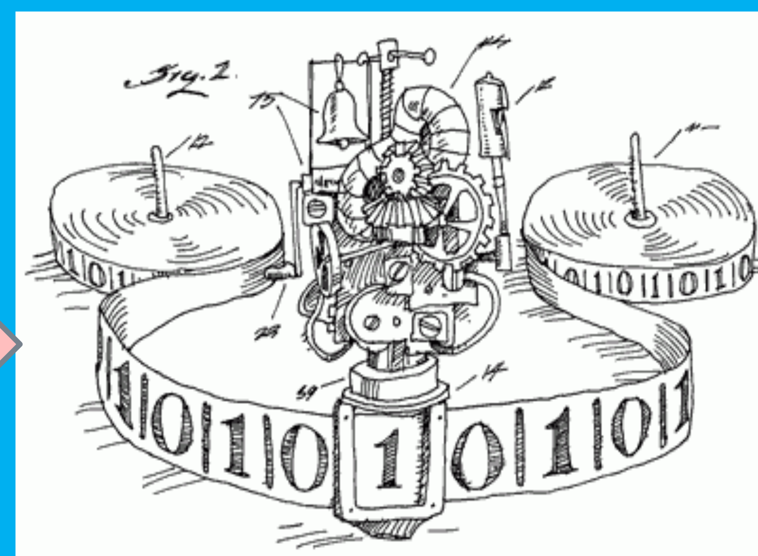
理論的には、この二つの発見で、三項の関係は明確になった。「コンピュータは、プログラムの実行という形で、証明を計算する」のである。

「証明」＝「計算」

証明



計算



「計算とはなにか？」「証明とは何か？」

歴史的に見ると、不完全性定理は、証明あるいは計算の性質についての探求の中で生まれた「最後の結論」ではないのだ。事実は、その逆である。

1930年代に、この分野は、不完全性定理に強烈な刺激を受けて、「証明とは何か？」「計算とは何か？」という基本的な「問い」に突き進むことで発展する。

ゲーデルの結果を受けて、「計算可能性」「証明可能性」についての探求が一斉に始まる。

この時代の白眉は、「計算可能性」についての見かけは全く異なるこれらのアプローチが（もちろん、それは、ゲーデルの結果を再現できるものだ）、次々と、「同値」であることが証明されていくことだ！

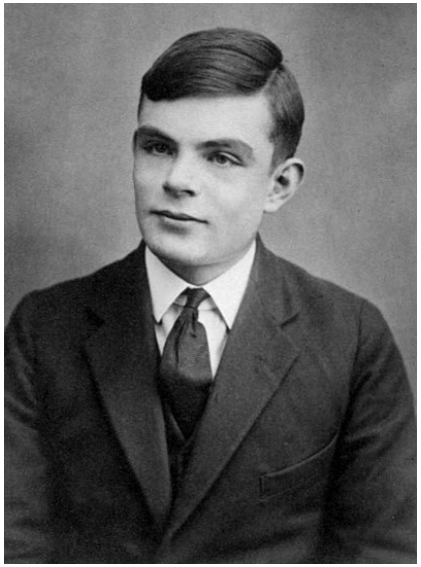
様々な「計算可能性」へのアプローチ

Kurt Gödel
1906-1978

帰納関数論

Alonzo Church
1903-1995

ラムダ計算



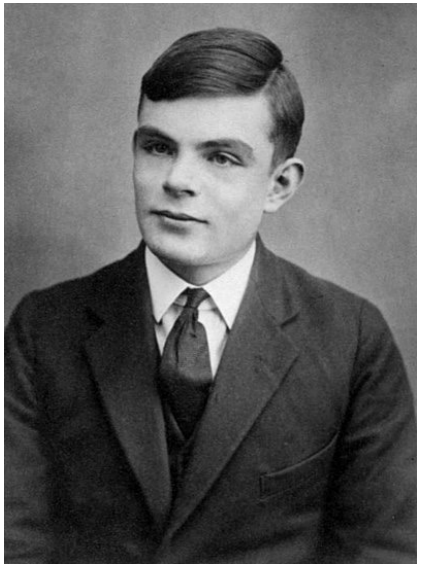
Alan Turing
1912-1954

チューリング
マシン

様々な「計算可能性」へのアプローチと
その同値性の認識

Kurt Gödel
1906-1978

Alonzo Church
1903-1995



Alan Turing
1912-1954

帰納関数論 ＝ ラムダ計算 ＝ チューリングマシン

チャーチ＝チューリングのテーゼ

1940年代には、今日、「チャーチ＝チューリングのテーゼ」と呼ばれる「計算可能性」についての認識は、確立されることになる。

「チャーチ＝チューリングのテーゼ」というのは、次のような提案である。

Every effectively calculable function (effectively decidable predicate) is general recursive

すべての実効的に計算可能な関数（実効的に決定可能な述語）は、一般帰納的である。

「証明可能性」と「計算可能性」の同一性の認識

ゲーデルの不完全性定理から、チャーチ＝チューリング・テーゼへの流れは、今から 90年前のこの時期に、「証明可能性」と「計算可能性」の同一性の認識は、明確に認識されていたことを示す。

しかし、コンピュータは、まだ存在しない！

プログラム

証明

「証明」＝「プログラム」

「カリー・ハワード対応」の発見と「従属型理論」の成立

ゲーデルらの発見から40年後の1970年代、論理学・数学の分野で、重要な動きがあった。ハワードの「カリー・ハワード対応」の（再）発見と、それに基づくマーティン・レフの「従属型理論」の成立である。

ハワードとマーティン・レフが、考えたこと

論理式 A → B が、A → B という関数と同じ型を持つと考えられるとしたら、他の論理式も、型を持つと考えることができるのでは？

例えば、論理式 A ∨ B は、型 A ∨ B を、論理式 A ∧ B は、型 A ∧ B を持つと考えることはできないか？

そして、この論理式（命題：Proposition）を型とみなすアイデアは、うまく機能するのである。こうした考えを、

Proposition as Type

と呼ぶ。

命題Pと証明p　　p : P

彼らは、この時、型Pの項pにあたるものを、命題Pの証明と考えることができることに気づく。それは、命題の証明が何から構成されるかについて、次のような自然な解釈を与える。

- **p: A ∧ B**　　pは、Aの証明とBの証明の両方
- **p: A ∨ B**　　pは、Aの証明、あるいは、Bの証明
- **p: A → B**　　pは、Aの証明からBの証明を導く方法
- **p: (∀x)B(x)**　　pは、任意のaに対してB(a)の証明を与える方法
- **p: (∃x)B(x)**　　pは、あるaに対するB(a)の証明

こうした考えを、

Proof as Term

と呼ぶ。

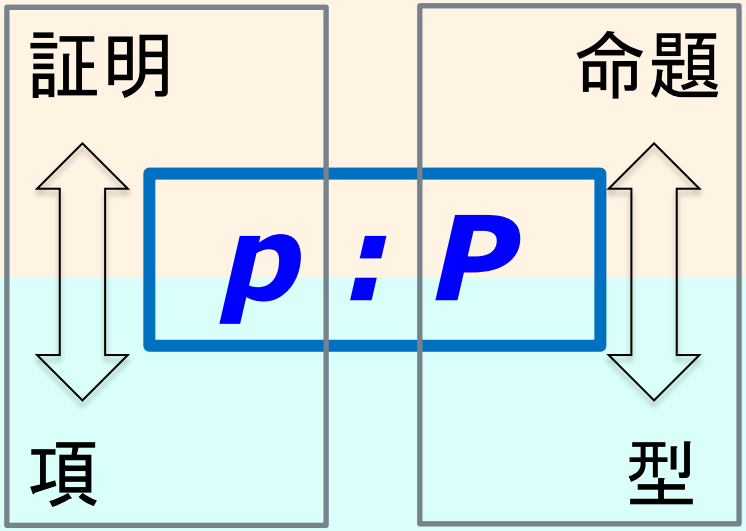
こうした、「型」と「命題」、「項」と「証明」との対応を

Curry-Howard対応 という

「*p* は、命題 ***P*** の証明である」

Proof as Term

証明は
項である



Proposition as Type

命題は
型である

「*p* は、型 ***P*** の項である」

「証明」＝「プログラム」

重要なことは、このCurry-Howard対応は、項の計算をプログラムと見なせば、"Proof as Program"としても解釈出来るということである。

いまから 40年近く前に、論理学者と計算機学者の一部は、「証明」＝「プログラム」という認識にたどり着いていた。

プログラム

計算

「プログラム」＝「計算」

「プログラム」＝「計算」

「証明」＝「プログラム」＝「計算」というテーゼの中では、「プログラム」＝「計算」というのが、一番わかりやすいように思う。
IT技術者なら、「プログラム」＝「アルゴリズム」＝「計算」と考えればいいと思う。

「プログラム」という概念自体、20世紀の中頃に生まれたものであるので、「プログラム」＝「計算」 というテーゼは、三つのテーゼの中では一番新しいものである。

この認識は、コンピュータの普及と、ITビジネスがコンシューマ中心にシフトして成功する中で、多くの人の中で自然に拡大した。

ただ、次のような問題もある。

「プログラム」＝「計算」？

現代では、誰もが「コンピュータはプログラムで動く」ことを知っている。ただ、コンピュータは、万能の「情報処理機械」とみなされていて、それが「計算する機械」であることを意識する人は少ない。

インターネットにしろYoutubeにしろFacebookにしろ、そのサービスを実現するソフトウェアが存在することを、多くの人は知っている。ただ、その基礎が、足し算や掛け算、電卓と同じ「計算」であることに、思いはいたらない。

サービスを開発するプログラマ自身、数値計算でもしない限り、自分たちの仕事が「計算」の領域に属するという意識は薄いように思う。

サービス

プログラム

計算

証明

「プログラム」＝「計算」

- 「チャーチ＝チューリングのテーゼ」や「カリー＝ハワード対応」に言及することなく、コンピュータの「論理的＝数学的推論能力」を理解することは可能であろうか？
- 僕は、「プログラムの行っていることは、基本的には、計算である」ということが腑に落ちれば、十分可能だと思う。
- ラムダ計算もチューリング・マシンも、人間が行う計算の、プリミティブだが忠実なモデルに他ならない。それは、コンピュータが行なっている計算と同じものだ。計算に関して言えば、人間とコンピュータに違いはない。
- そうしたプリミティブな機械の働きが、「証明」とつながることを実感するには、いまなら、Coqの経験は、とても役に立つ。
- それは、「計算」＝「証明」、「証明」＝「プログラム」という、多くの人にとっての「ミッシング・リング」を埋めてくれるだろう。
  僕が、IT技術者にCoqの学習を勧める理由は、そこにある。

プログラム

1970年代

21世紀の人口知能

計算

証明

コンピュータは、
プログラムの
実行という形で、
証明を計算する

1930年代

機械が論理的＝数学的推論能力を持つという認識が、やがて、現実を動かすようになる

それは、自然言語ベースの「人工知能」技術を、機械の推論能力をベースとした、新らしい人工知能技術を準備して行くだろう。


人工知能と数学
第三部　21世紀
数学の基礎と計算科学の新しい動向

第三部　21世紀

数学の基礎と計算科学の新しい動向

- Rosetta Stone、再び
- 21世紀 数学的証明での「形式的証明」の拡大
- Univalent FoundationとUniMath

Rosetta Stone、再び

20世紀：数理科学の発展と拡大

20世紀を通じて、数学・物理を中心とする数理科学は、大きな発展を遂げた。

それは、内部に多くの新しい研究領域を発生させながら、それぞれの領域は相互に越境し、さらにその領域を拡大し続けた。

前回見た、「計算」＝「証明」＝「プログラム」という認識も、新しい領域を巻き込み、長い時間をかけて獲得された、そうした認識の一例である。

20世紀、我々の数理科学的認識は、かつてなく拡大し発展した。

21世紀：数理科学の統一的把握の探究

21世紀になって、こうして拡大した数理科学の全体像を統一的に把握しようという探究がうまれてくる。

数理科学の専門化した諸領域の到達点を概観するのは、必ずしも容易ではないのだが、やがて、それらの間に驚くべき類似性が存在するという認識が生まれる。

John Baezの“A Rosetta Stone” 論文

今回のセッションで紹介する John Baezらの“A Rosetta Stone”論文は、数理科学の統一的把握の試みとして、もっとも成功した、もっとも影響力のあったものの一つである。

彼は、量子論と相対論の統一をめざす「量子重力」の研究者だった。

彼の認識の飛躍は、数学的には、数理科学の基礎にカテゴリー論をおくことで可能になった。


John Baez

Physics, Topology, Logic and Computation: A Rosetta Stone

John C. Baez
Department of Mathematics, University of California
Riverside, California 92521, USA

Mike Stay
Computer Science Department, University of Auckland
and
Google, 1600 Amphitheatre Pkwy
Mountain View, California 94043, USA

https://arxiv.org/abs/0903.0340

Baezの問題意識：20世紀の物理学で起きたこと

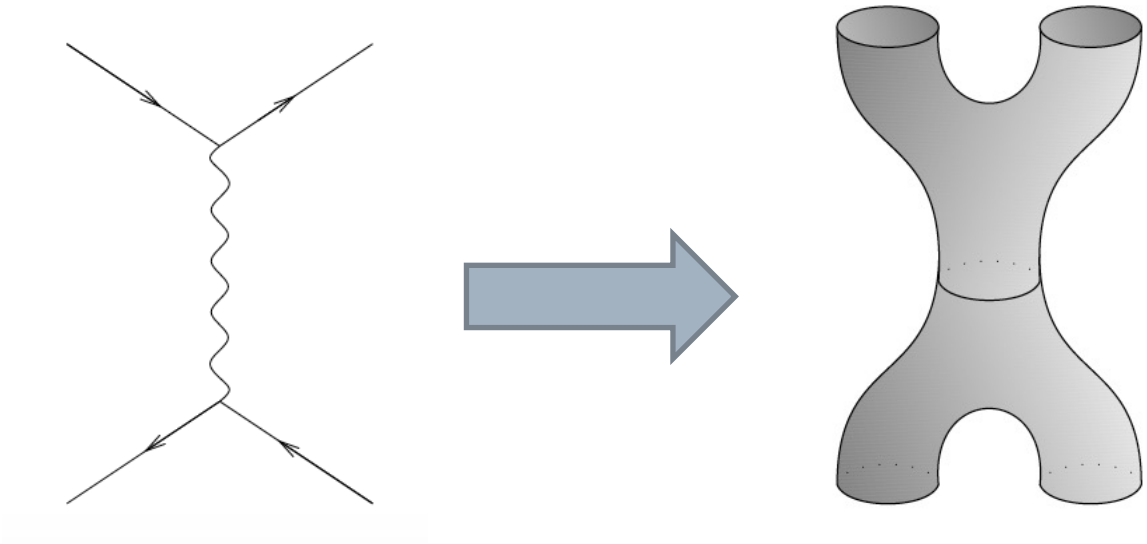
「1949年頃、ファインマンは、場の量子論において、線形演算子を図として描くと便利であることに気づいた。これにより、演算子を図形として理解することができる。重要なのはトポロジーであり、幾何学ではないのだ。

1970年代、ペンローズは、ファインマン・ダイアグラムの一般化が量子論の至るところで生じ、時空間の理解を改めることにつながるかもしれないことに気づいた。

1980年代には、これらの図の根底には、量子物理学とトポロジーの強力な類似性があることが明らかになった！ つまり、線形演算子は「コボーディズム」（n次元の多様体が1次元小さい多様体の間を行き来すること）に非常によく似た振る舞いをするのだ：

弦理論では、この類似性を利用して、通常の場の量子論のファインマン・ダイアグラムを2次元のコボルディズムに置き換えて、時間の経過とともに弦が辿る世界のシートを表現する。

演算子とコボルディズムの類似性は、ループ量子重力や、より純粋に数学的な分野である「トポロジカル場の量子論」においても重要である。」



Feynman diagram

cobordism

物理学とトポロジーの類似

物理学	トポロジー
ヒルベルト空間 (システム)	(n-1)次元多様体 （空間）
ヒルベルト空間間の 演算子 （プロセス）	(n-1)次元多様体間の cobordism （時空）
演算子の合成	cobordismの合成
同一演算子	同一cobordism

物理学とトポロジーと論理学と計算理論の Rosetta Stone

彼は、このアナロジーをさらに推し進める。物理学とトポロジーだけでなく、論理学と計算理論もリストに加えて、数理科学の「パラレル・コーパス」を作成するのだ！

導きの糸となったのは、カテゴリー論の基本的な構成要素である、「オブジェクト」と「モルフィズム」の対応物を、これらの理論に中に見つけることだ。

物理学でオブジェクトに対応するのは「システム」で、モルフィズムに対応するのは、「プロセス」。トポロジーでオブジェクトに対応するのは「多様体」、モルフィズムに対応するのは「cobordism」等々。

こうして、物理学とトポロジーと論理学と計算理論のRosetta Stoneの骨組みが完成する！

Rosetta Stone (Pocket version)

カテゴリー論	物理学	トポロジー	論理学	計算理論
オブジェクト	システム	多様体	命題	データ型
モルフィズム	プロセス	コボルディズム	証明	プログラム

Monoidal Category

Monoidal Category
オブジェクトのテンソル積

- 物理学では、2つの系が並んで1つの系を形成していると考えると便利なことが多い。
- トポロジーでは、2つの多様体の分離したままでの結合は、それ自体が多様体である。
- 論理学では、2つの文の連言は再び文となる。
- プログラミングでは、2つのデータ型を組み合わせて1つの「データ型の積の型」にすることができる。

モノイダル・カテゴリーのオブジェクトのテンソル積の概念は、これらすべてを1つの枠組みで統合するものである。

Monoidal Category
モルフィズムのテンソル積

モノイダル・カテゴリーでは、「並列」にも「直列」にも処理を行うことができる。

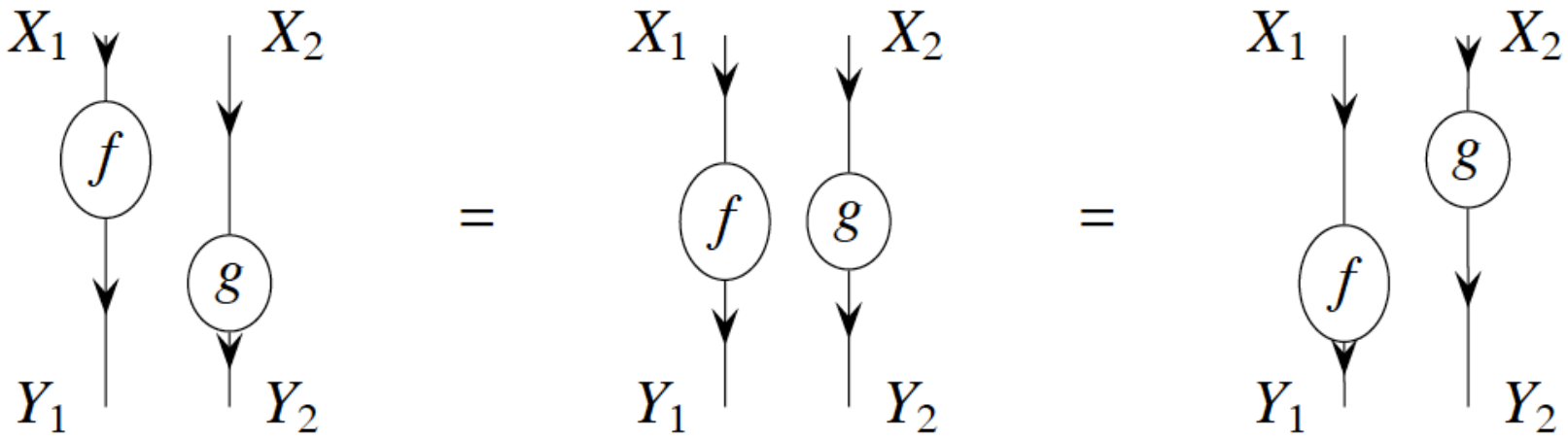
「直列」処理するは、モルフィズムの単なる合成であり、これはどのカテゴリーでも機能する。

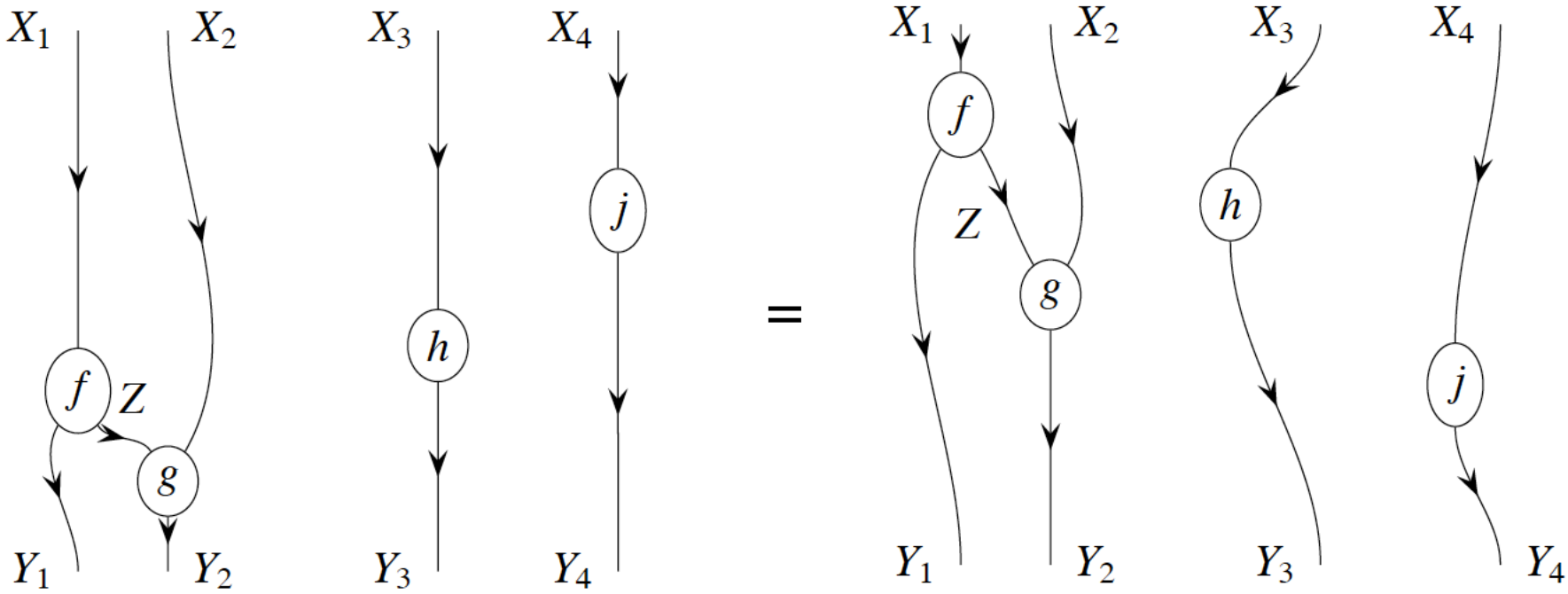
モノイダル・カテゴリーでは、二つのモルフィズム
$f : X \to Y$ と $f' : X' \to Y'$ のモルフィズムのテンソル積をとして、
「並列」処理 $f \otimes f' : x \otimes x' \to y \otimes y'$ が得られる。:

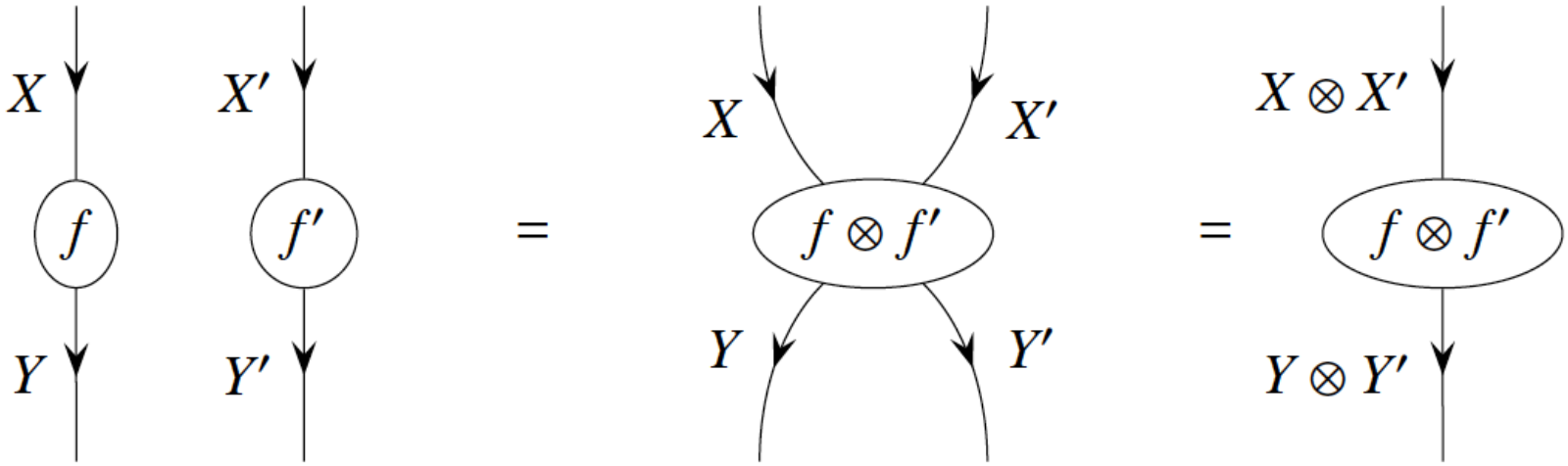
Rosetta Stone (Larger version)

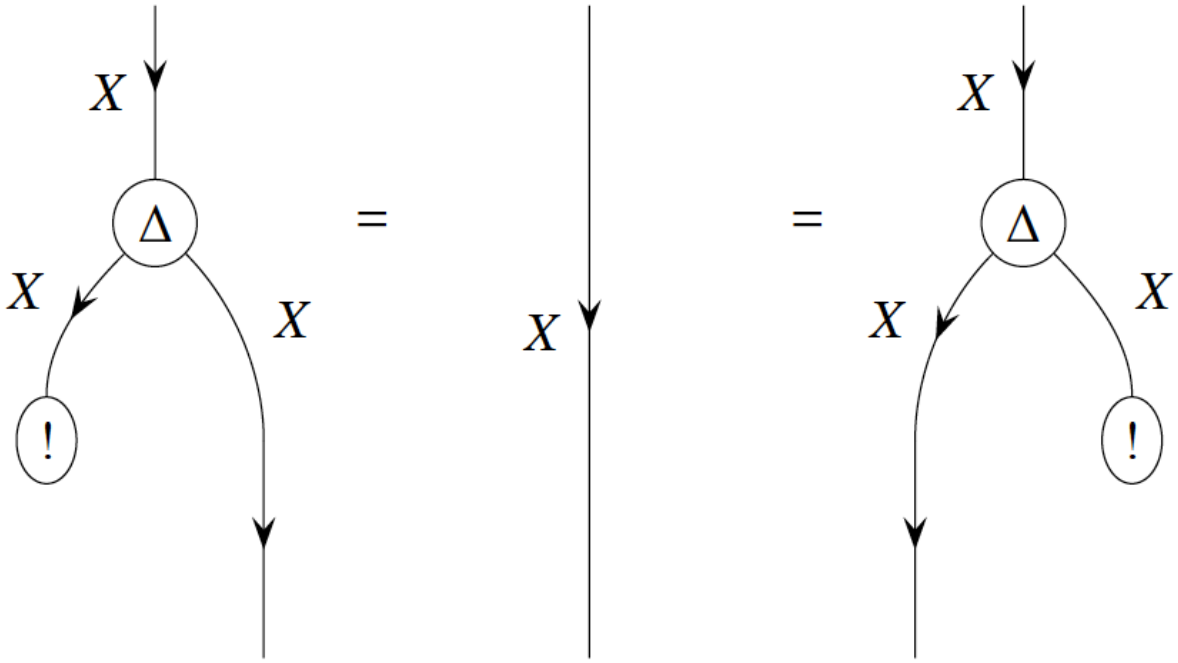
カテゴリー論	物理学	トポロジー	論理学	計算理論
オブジェクト X	ヒルベルト空間 X	多様体 X	命題 X	データ型 X
モルフィズム $f: X \to Y$	演算子 $f: X \to Y$	コボルディズム $f: X \to Y$	証明 $f: X \to Y$	プログラム $f: X \to Y$
オブジェクトのテンソル積 $X \otimes Y$	結合システムのヒルベルト空間 $X \otimes Y$	多様体の直和 $X \otimes Y$	命題の連言 $X \otimes Y$	データ型の積 $X \otimes Y$
モルフィズムのテンソル積 $f \otimes g$	並行プロセス $f \otimes g$	コボルディズムの直和 $f \otimes g$	並行実行証明 $f \otimes g$	並行実行プログラム $f \otimes g$
内部Hom $X \multimap Y$	「反XとY」のヒルベルト空間 $X^* \otimes Y$	向きを反転したXとYの直和 $X^* \otimes Y$	条件命題 $X \multimap Y$	関数型 $X \multimap Y$

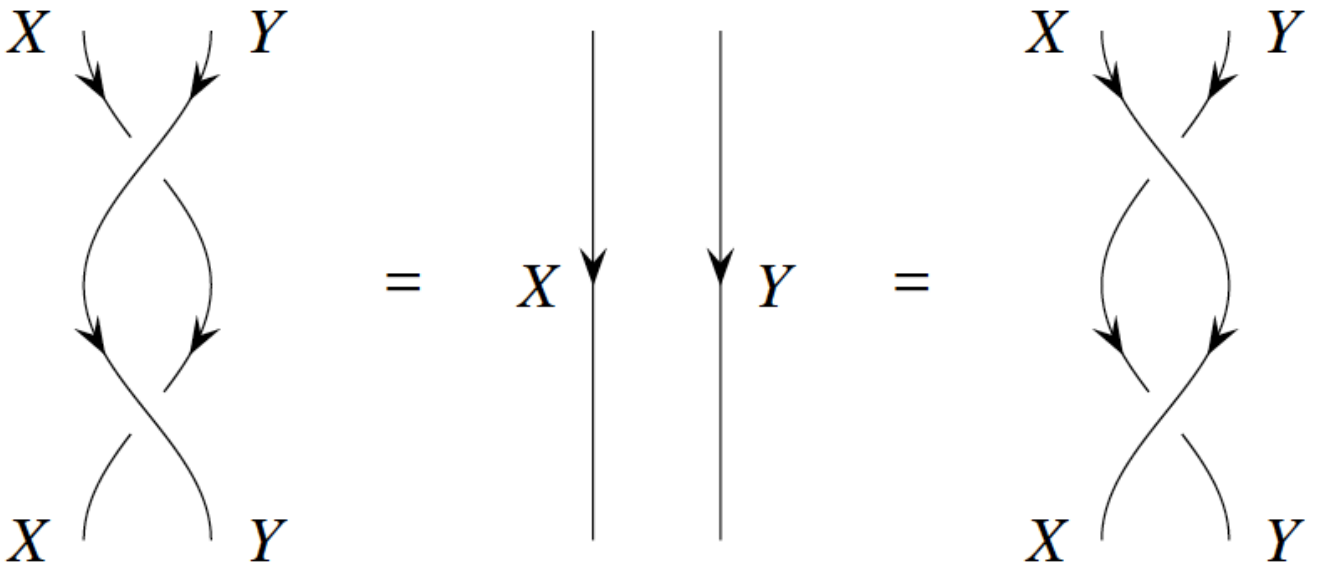
Monoidal Categoryの String Diagramでの表示

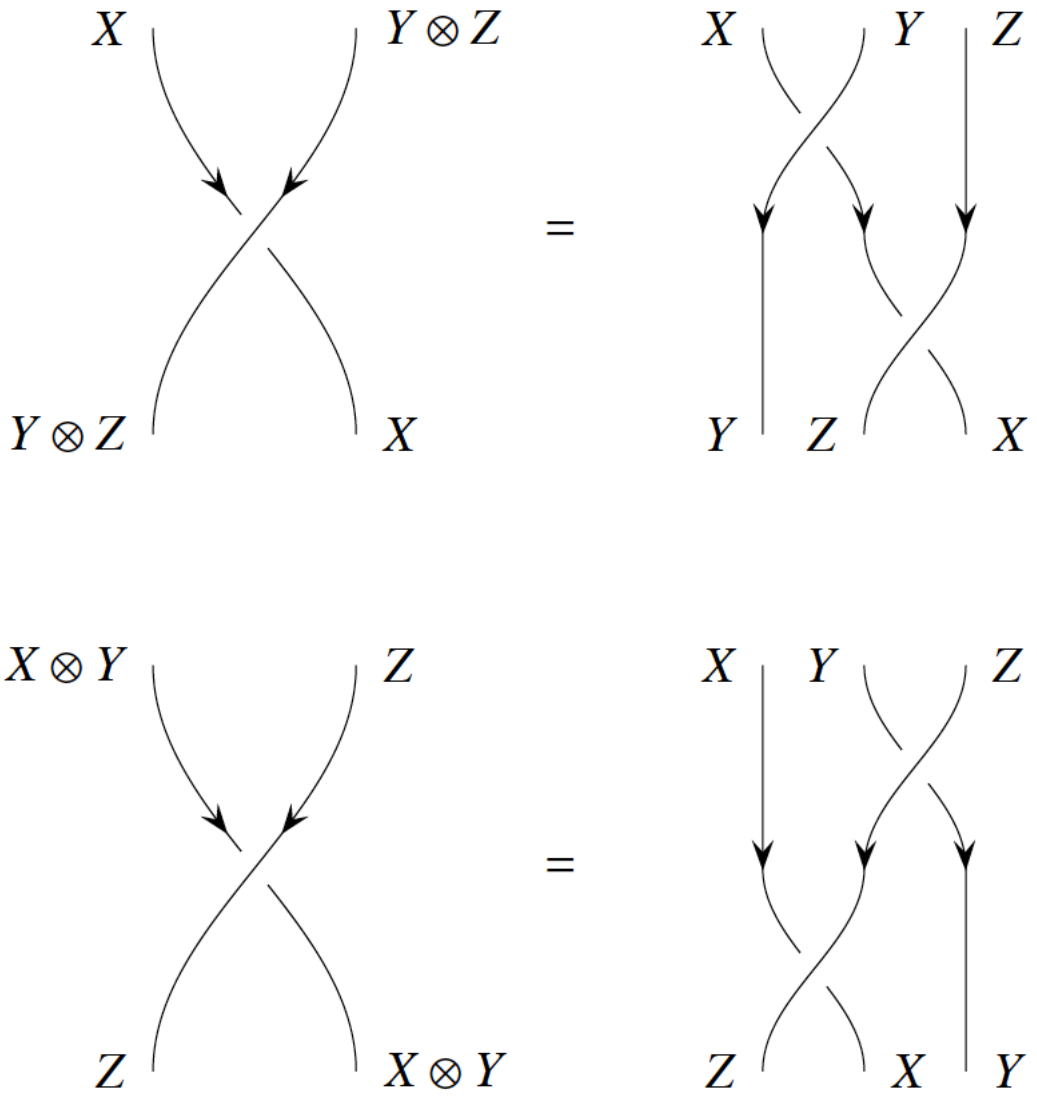

X_1
X_2
f
g
Y_1
Y_2
=
X_1
X_2
f
g
Y_1
Y_2
=
X_1
X_2
g
f
Y_1
Y_2

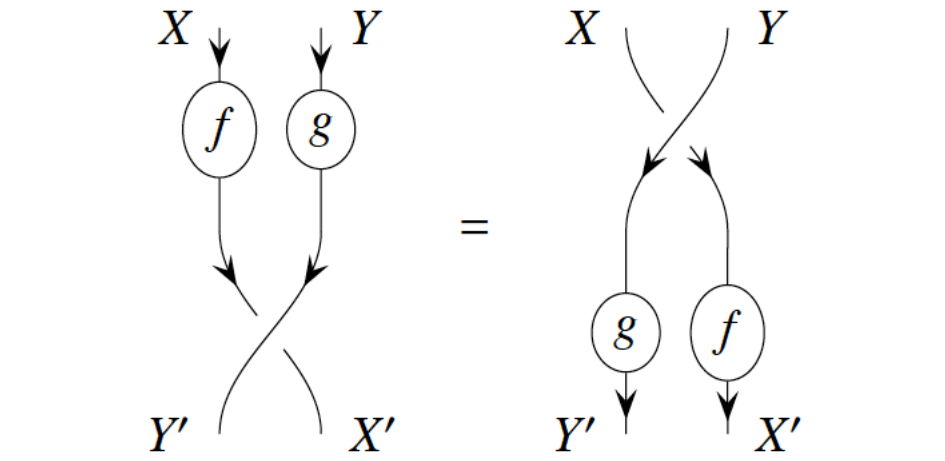

X_1
X_2
X_3
X_4
f
Z
g
h
j
Y_1
Y_2
Y_3
Y_4
=
X_1
X_2
X_3
X_4
f
Z
g
h
j
Y_1
Y_2
Y_3
Y_4

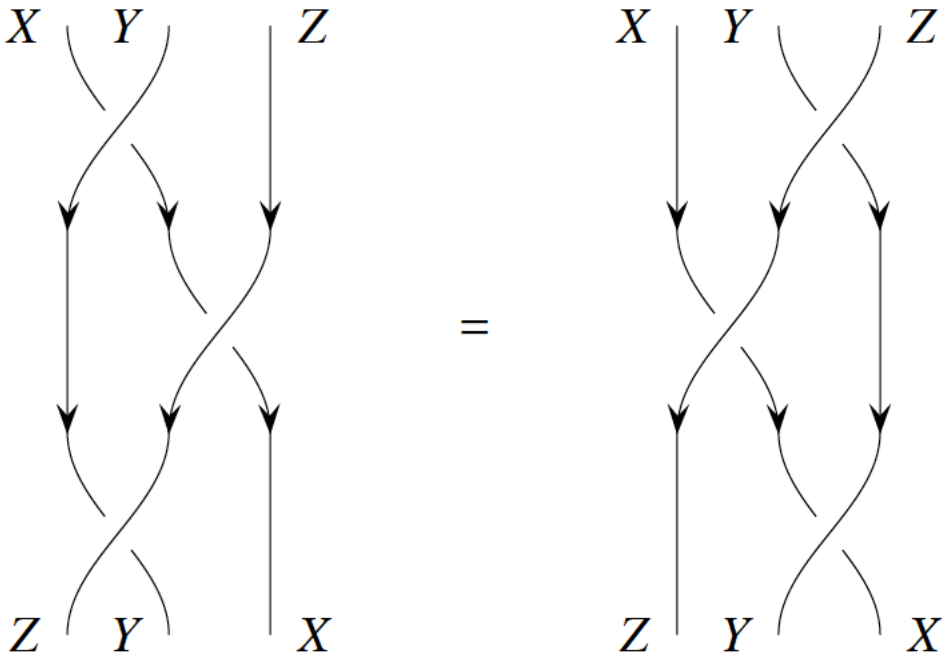

X
X′
f
f′
Y
Y′
=
X
X′
f ⊗ f′
Y
Y′
=
X ⊗ X′
f ⊗ f′
Y ⊗ Y′

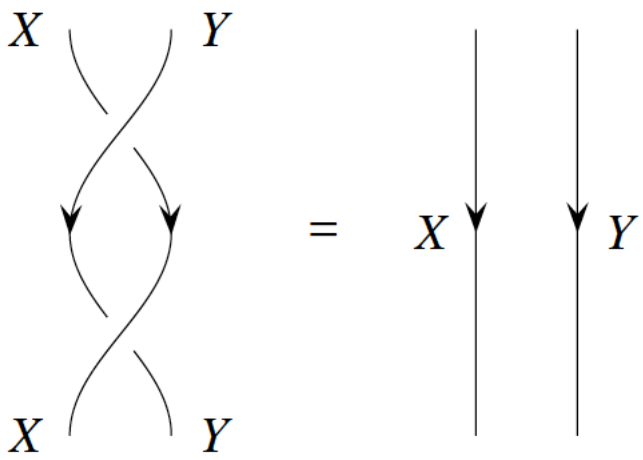

X
Δ
X
X
!
=
X
=
X
Δ
X
X
!

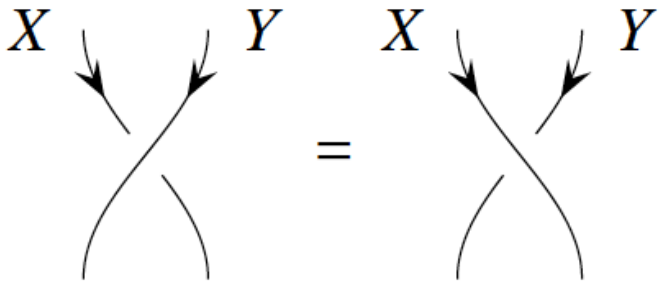

X
Y
X
Y
=
X
Y
=
X
Y
X
Y

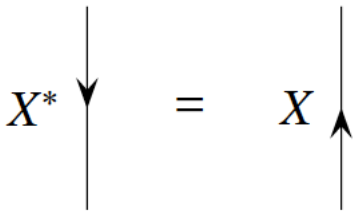

X
Y ⊗ Z
Y ⊗ Z
X
=
X
Y
Z
Y
Z
X
X ⊗ Y
Z
Z
X ⊗ Y
=
X
Y
Z
Z
X
Y


X
Y
f
g
Y′
X′
=
X
Y
g
f
Y′
X′


X
Y
Z
Z
Y
X
=
X
Y
Z
Z
Y
X

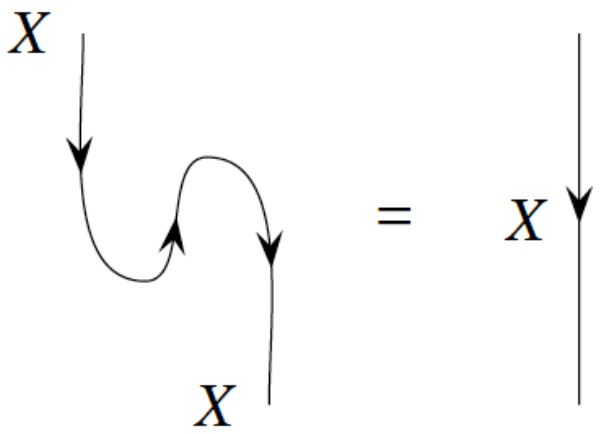

X
Y
X
Y
=
X
Y

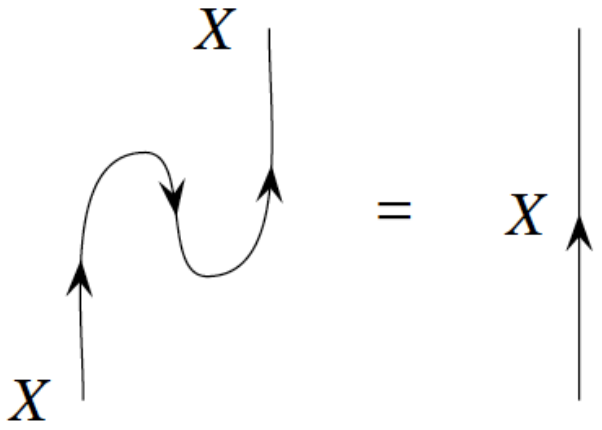

X
Y
=
X
Y

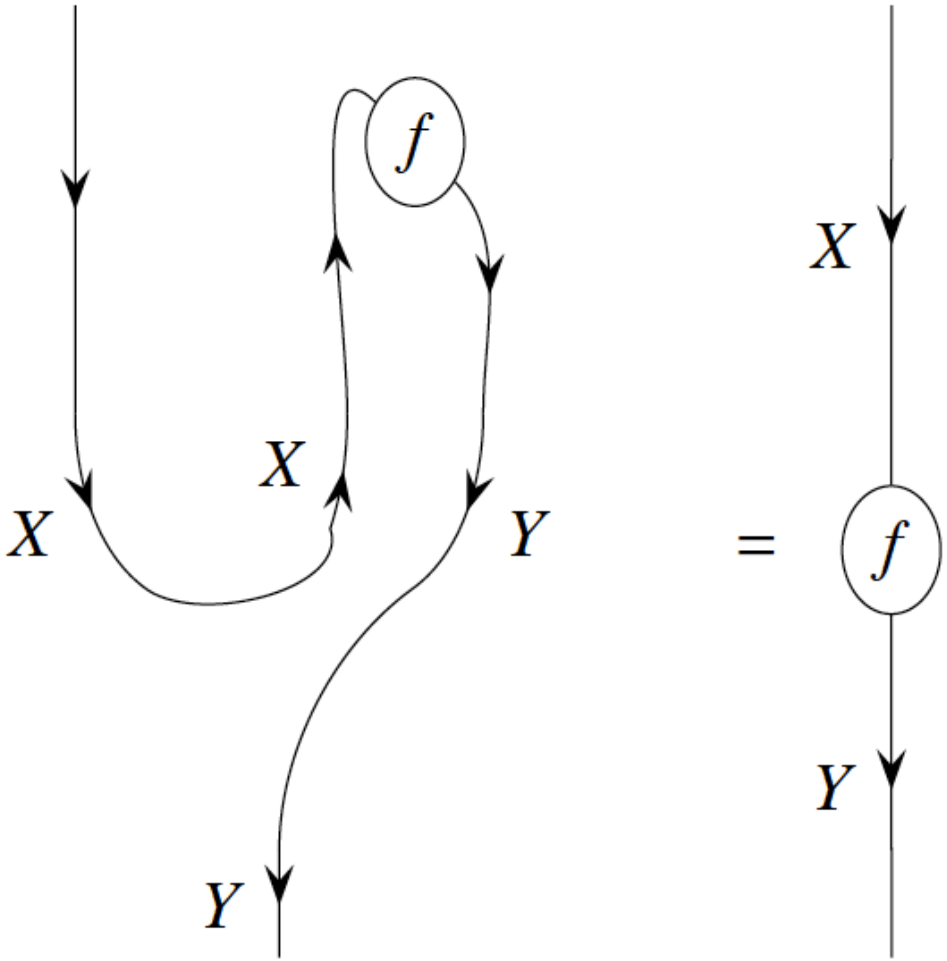


Cap

Cup






f
X
X
Y
Y
=
X
f
Y


21世紀
数学的証明での「形式的証明」の拡大

数学的知の「累積性」

数学的知には、人間の認識によって獲得されたものとしては、独特の性質がある。

それは、いったん数学的に「真」であることが証明された定理は、時代が変わっても場所が変わっても、ずっと「真」のままであり続けることだ。

そうして獲得された知は、個人というよりは人類の知として蓄積されていく。それを数学的知の「累積性」という。

「巨人の肩の上の小人」

「フェルマーの定理」を証明したワイルズは、自分を「巨人の肩の上の小人」に喩えたのだが、それは彼の謙遜であると同時に、数学的知の体系が「累積的」であることを彼がはっきりと自覚していることを示している。

マルゼミのセミナー「認識の認識」
https://www.marulabo.net/docs/philosophy02/
で紹介した、GrzegorczykやKripkeのモデルは、いずれも知の累積性に基づいた認識モデルである。

数学的知の「累積性」を保証する前提条件
数学的知の共有

それでは、「数学的知の累積性」は、どのように保証されるのであろうか？

ピタゴラスの「教団」は、いくつかの発見を「秘教」にしていたらしいのだが、ピタゴラスの定理を使うのに、我々はその「証明」を自分で繰り返し「再発見」する必要はない。

数学では、「数学的知の共有」は、学問自体の前提でさえある。

コンピュータの世界で、オープンソースのムーブメントが大きな影響力を獲得するはるか以前から、数学の世界は、オープンソースの世界だった。

一番重要なことは、
その「数学的知」が数学的に正しいことである

ただ、「情報の共有」が、「数学的知の累積性」を保証するとは限らない。「数学的知の累積性」にとって、一番重要なことは、その「数学的知」が数学的に正しいことである。

しかし、数学的正しさを証明する「数学的証明」は、現状では、いくつかの問題を抱えている。そのことを、まず見ておこう。

21世紀に入って、数学的証明にコンピュータを利用しようという「形式的証明」の動きが拡大する。その背景にあるのは、「数学的証明」の現状の問題に対する反省である。

現状の数学的「証明」が抱える問題

誤っていた「証明」

「フェルマーの定理」は、最終的には、1995年のAndrew Wilesの論文によって証明されたが、フェルマー自身は、この定理を証明出来たと信じていた。それが、誤った「証明」であったのは確かである。

Wiles自身も、95年の論文以前に一度、「証明した」とする発表を行うが、誤りが見つかり、それを撤回している。

世紀の難問である「リーマン予想」は、何度か「証明」が発表されている。今までのところ、それは誤って「証明」だった。

正しいことの判定が難しい証明

Grigory Perelmanは、2002年から2003年にかけて arXiv に投稿した論文で、「三次元のポアンカレ予想」を解いたと主張した。

arXivは、学会の論文誌とは異なり、掲載の可否を決める査読が基本的にないので、発表の時点では、彼の論文以外に、彼の証明が正しいという裏付けはなかった。

彼の証明の検証は、複数の数学者のグループで取り組まれ、最終的に彼の証明の正しさが「検証」されたのは、2006年のことだった。

非常に膨大な証明

「有限単純群の分類問題」は、Daniel Gorensteinをリーダーとする数学者の集団によって解決され、20世紀の数学の大きな成果の一つと呼ばれている。

ただ、その「証明」は、100人以上の数学者が、50年のあいだに数百の論文誌に発表した、膨大な量の「証明」の総和である。そのページ数は一万ページを超えると言われている。

（「全部を読んでる数学者は、１人もいない。」というのがジョークなのか本当なのか、僕には分からない。）

21世紀:

コンピュータを利用した
「形式的証明」の拡大

「四色問題」のコンピュータによる解決

1974年、イリノイ大学のKenneth AppelとWolfgang Hakenは、当時の最先端のコンピュータを使って、「四色問題」を解いた。

コンピュータの稼働時間は、1200時間に及び、夜間の空き時間しか利用が認められなかったので半年がかりの計算だったと言う。

コンピュータが人手にはあまる膨大な計算を行ったのは事実だが、当時、その計算の正しさの検証しようのないことを理由に、こうした「証明」を疑問視する声も一部にはあった。

Feit–Thompson定理の形式的証明

2004年、Georges Gonthierは、Coqを使って「四色問題」を解いた。

2012年、Georges Gonthierは、Coqを用いて、群論のFeit–Thompson 定理の形式的証明に成功する。

この定理は、1962-1963年にWalter Feit と John Griggs Thompson によって証明された、群論の基本的な定理の一つである。

この定理は、「奇数位数の有限単純群は可解である」ことを主張し(「奇数位数定理 = “Odd Order Theorem” とも呼ばれる」)、有限単純群の分類問題で重要な役割を果たしている。

Georges Gonthier

Gonthierの ”Feit–Thompson theorem” の型式的証明も、膨大なものである。それは、15,000の定義、4,300の定理、17万行のソースからなる。この証明を、彼はチーム作業で、6年かけて完成させた。

https://github.com/math-comp/odd-order

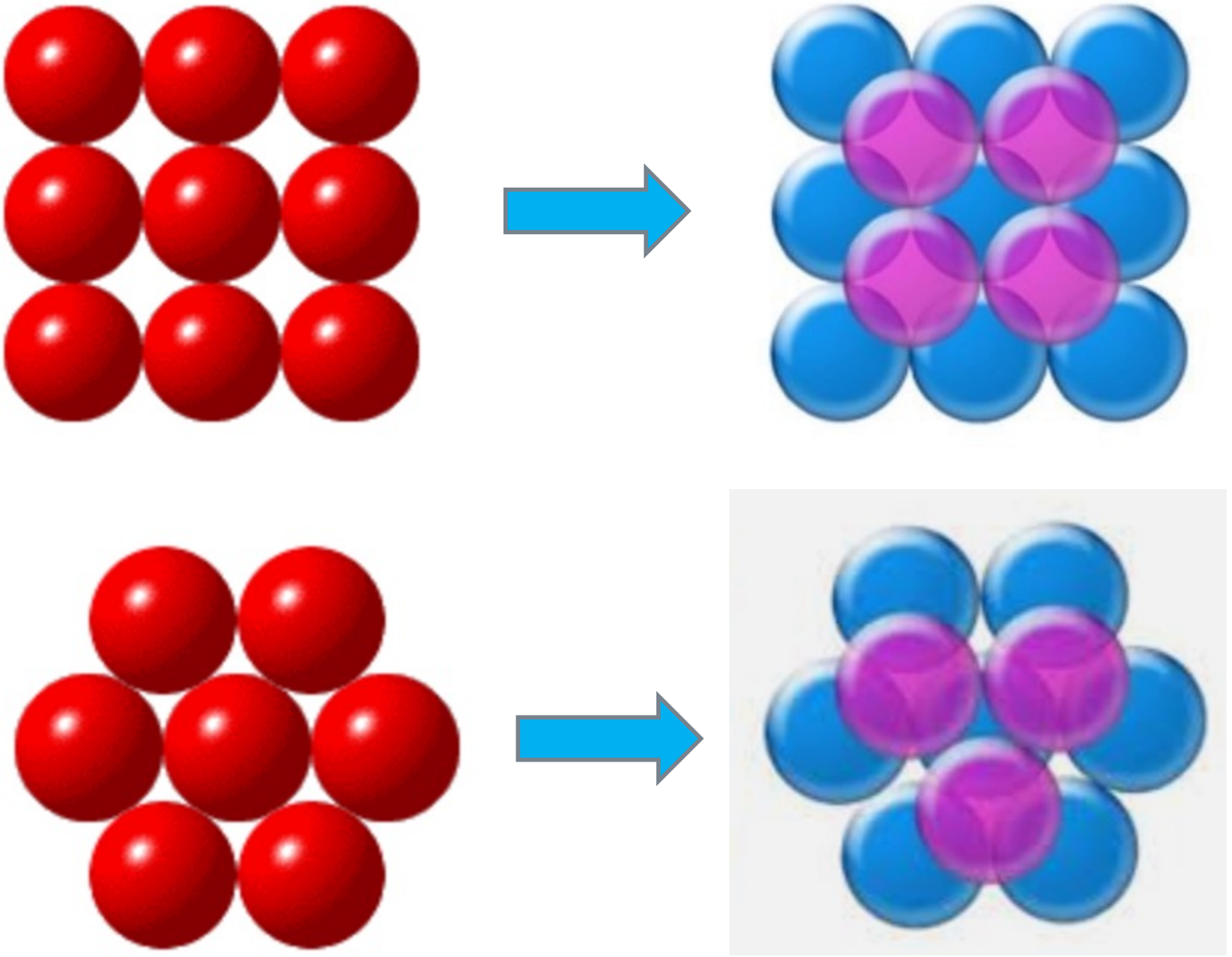
“A Machine-Checked Proof of the Odd Order Theorem”
https://hal.inria.fr/hal-00816699/document

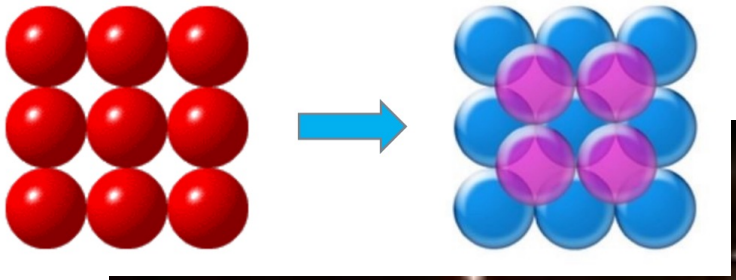
ケプラー予想

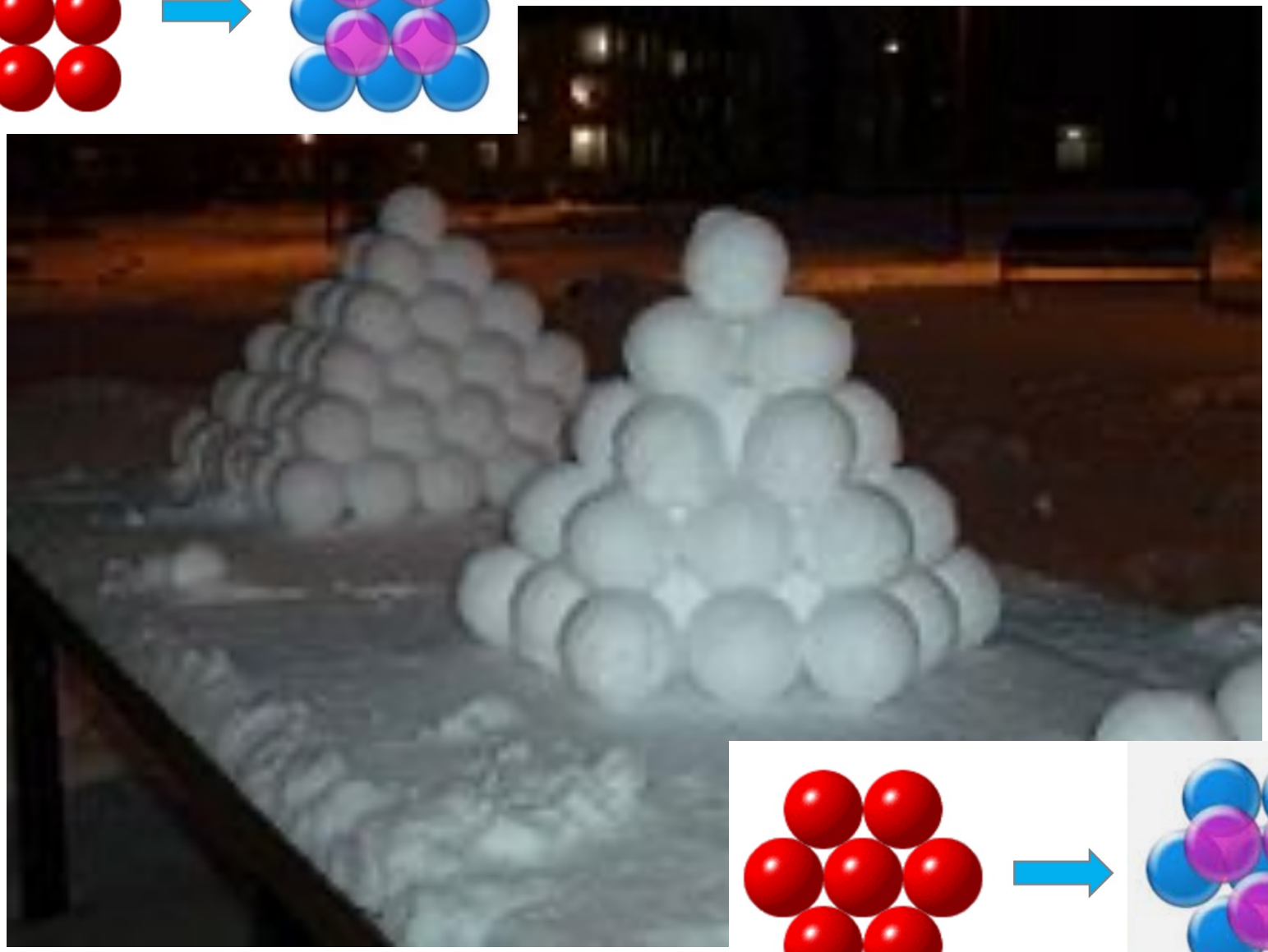
「ケプラー予想」は、入れ物の箱に同じ大きさのボールを、どのくらいたくさん詰め込むことができるかという問題に関するで予想である。

何も考えずに、箱の上からボールを流し込むと、充填率（詰め込んだボール全体の体積を、容器の箱の体積で割ったもの）は、平均すると 65%程度になるという。箱の35%は、すきまになるということだ。

もっと、びっしりと、ボールを詰め込みことはできないか？
ケプラーは、もっとボールを詰め込むことができる二つの方法を考えた。







ケプラー予想、300年解かれず
ヒルベルトの18番目の問題になる

二つの充填率は、ともに、74.05%で等しい。

正確にいうと、それは $\frac{\pi}{3\sqrt{2}}$=0.740480489... となる。

1611年、ケプラーは、どんなボールの詰め方をしても、この数字を超えるほど密にボールを詰め込むことはできないだろうと予想した。これを「ケプラー予想」という。ケプラー自身は、この問題に証明を与えることはできなかった。

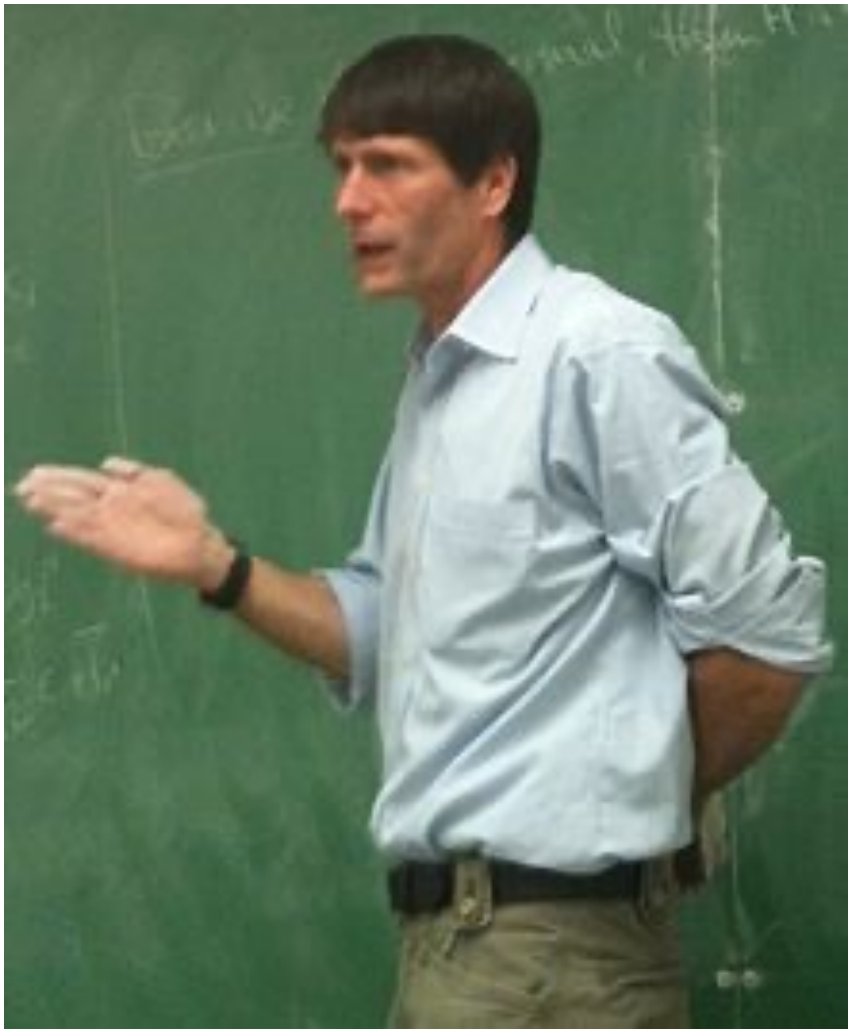
1900年のヒルベルトの20世紀の数学が解くべき23の問題の 18番目にこの問題は、取り上げられている。

ケプラー予想の400年

1611年

Johannes Kepler

2014年



Thomas Hales

Hales ケプラー予想を解く

1992年、Thomas HalesはSamuel Fergusonとともに、可能なボールの配置について総当たりで、線形計画法の手法で充填率の最大値を求めることで、ケプラー予想が解けると予想する。

1998年、Halesらは、250ページの論文と2GByteのプログラムとデータを提出し、ケプラー予想を解いたと主張する。ただ論文の査読者は「99%正しい」としたものの、1%を留保した。

2003年、Halesはケプラー予想のコンピュータによる完全に形式的証明をめざす Flyspec プロジェクトを立ち上げる。

2015年、Halesと協力者21名は、論文 "A formal proof of the Kepler conjecture"をarXivに投稿し、ケプラー予想の解決を宣言する。

2017年には、彼らの証明は、学会に受理される。

連続体仮説の形式的な独立性証明

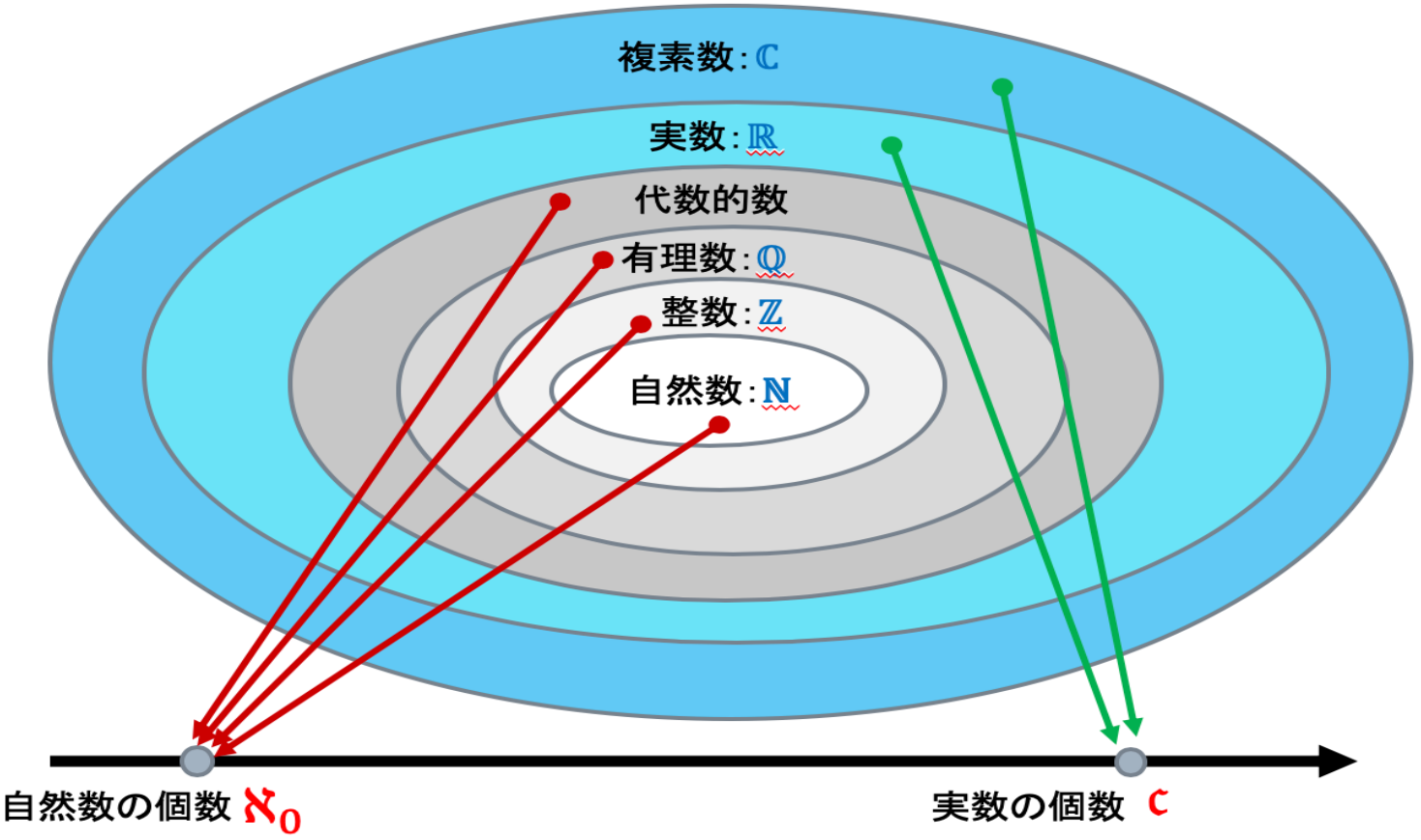
2020年、Jesse Michael Han, Floris van Doornは、連続体仮説の集合論ZFからの独立性の形式的証明 – コンピュータによる証明を与えた。

“A Formal Proof of the Independence of the Continuum Hypothesis”
https://arxiv.org/pdf/2102.02901.pdf

その証明は、GitHubで公開されている。
https://github.com/flypitch/flypitch/

連続体仮説とは何か？

$\mathfrak{c} = 2^{\aleph_0}$ ？



Jesse Michael Han

Researcher at @OpenAI

Math PhD at the University of Pittsburgh, interested in formal proofs and applying deep learning to automated theorem proving.

VevodskyのUnivalent Foundation

21世紀に入って、数学でコンピュータによる形式的証明を受容する動きが広がっているのを見てきた。

数学者の証明の誤りの問題を、数学の基礎である累積性への脅威として真正面から捉えたのは、Voevodsky である。

Univalent FoundationとUniMath

Voevodsky

2017年に亡くなったVoevodskyは、Milner予想、Bloch-Kato予想を解くなど、代数幾何でグロタンディックが進もうとした道で、大きな業績を残した。

Voevodskyの最後の仕事は、数学の基礎とコンピュータに関係していた。

Vladimir Voevodsky
1966-2017

「間違った証明」についての Voevodsky自身の経験

2000年頃、彼は1993年に自分が発表した論文の重要な補題が間違っていたことに気づく。でも、その頃には、その論文は広く出回っていて、多くの数学者がその証明を「信じて」いた。彼が、その間違った補題なしでも、論文の結論が正しいことを証明できたのは、2006年になってからだった。

別のこともあった。1998年に共著で彼が発表した論文の証明に対して、「正しくない」という批判が出される。結論的には、彼は、正しかったのだが、彼が、最終的に、自分が正しいことを確信できたのは、2013年になってからだった。

このあたりの経緯は、"The Origins and Motivations of Univalent Foundations" https://goo.gl/LW2Wcq に、詳しく触れられている。

「誤りの累積」をどう防ぐのか？

Voevodskyは、考える。「数学が、累積的（accumulative）な性格を持つのなら、もしも、そこに誤りが紛れ込むと、それも、累積する可能性がある。」

こうした問題に対する、Veovodskyの対応は驚くべきものだった。

彼は、集合論に変わる新しい数学の基礎づけ Univalent Foundationを構築し、さらに、数学の証明に、コンピュータを使うべきだと主張した最初の数学者の一人となった。また、そのためのコンピュータによる証明支援システムのライブラリーUniMthを開発した。

Veovodskyが考えたこと

Veovodskyの考えたことを、彼のインタビューから紹介しよう。

https://web.archive.org/web/20170822201414/http:/baaltii1.livejournal.com/198675.html

ここでの引用は、彼の突然の死に際して、John Baezが寄せた弔文からのものである。

https://johncarlosbaez.wordpress.com/2017/10/06/vladimir-voevodsky-1966-2017/

数学の二つの危機

本格的に何かやりたいのであれば、これまで培ってきた数学の知識や技術を最大限に生かすべきだと、すぐに理解しました。

一方、科学としての数学の発展の流れを見ていると、そろそろ、また新たな予想を証明しても、あまり効果がない時代が来ていることに気づきました。数学は今、危機的状況、いや、2つの危機に瀕しているのだと実感しました。

第一は、それは、「純粋数学」と「応用数学」の分離に関連しています。遅かれ早かれ、実用性のないことに従事している人たちに、なぜ社会がお金を払わなければならないのかという疑問が生じるのは明らかでした。

数学の二つの危機

「第二は、あまり明らかではありませんが、純粋数学の複雑さに関連しています。遅かれ早かれ、論文が複雑すぎて詳細な検証ができなくなり、発見されないエラーが蓄積される過程が始まるということです。そして、数学は非常に奥の深い学問であり、ある論文の結果が、通常、何本も何本も前の論文の結果に依存するという意味で、数学にとってのこの誤りの蓄積は非常に危険です。

そこで私は、このような危機を防ぐのに役立つことをやってみる必要があると考えました。最初の危機については、近年あるいは数十年前に開発された純粋数学の手法が解決に必要な応用問題を見つける必要があることを意味したのです。」

数学のコンピュータによる検証

このとき、私はいわゆる「好奇心による研究」を大きくやめて、将来について真剣に考えるようになったのだと思います。好奇心が導く領域、自分が価値や興味、美しさを感じる領域を探求するための道具がなかったんです。

そこで、そのような道具を作るために何ができるかを検討し始めたのです。そしてすぐに、唯一の長期的な解決策は、私が抽象的、論理的、数学的に構築したものを、コンピューターを使って検証できるようにすることだということが明らかになりました。

このためのソフトウェアは、60年代から開発されていました。当時、私が2000年頃に実用的な証明アシスタントを探し始めたときには、何も見つかりませんでした。いくつかのグループがそのようなシステムを開発していましたが、そのどれもが、私がシステムを必要としている数学の種類には全く適していませんでした。

数年で状況は変わった

「私がその可能性を模索し始めた頃、数学者の間ではコンピュータによる証明の検証はほとんど禁じられた話題でした。コンピュータによる証明補助が必要だという話になると、必ずゲーデルの不完全性定理（実際の問題とは全く関係ない）の話や、すでにある証明の検証の事例の1つか2つに流れてしまい、この考え方がいかに非現実的かを示すためだけに使われました。

この時期、数学におけるコンピュータ検証の分野を発展させようと粘り強く努力したごく少数の数学者の中に、トム・ヘイルズとカルロス・シンプソンがいました。

それからわずか数年後の今日、証明や数学的推論全般のコンピュータ検証は、Univalent Foundationやホモトピー型理論に取り組む多くの人々にとって完全に実用的なものに見えます。」

集合論的な数学の基礎づけの問題

「この課題に取り組む必要があったのは、数学の基礎がこの課題の要件に対して準備されていなかったからです。数学的推論をコンピュータが追従できるほど正確な言語で定式化するということは、数学の基礎体系を、いくつかの基本定理を確立するための整合性の基準としてではなく、日常の数学的作業で使用できる道具として使用するということです。

既存の基礎体系には2つの大きな問題があり、それらは不十分なものでした。第一に、既存の数学の基礎は、述語論理の言語に基づいており、このクラスの言語はあまりにも限定的でした。第二に、既存の基礎は、例えば、私の2-theoryに関する研究のような対象に関する記述を直接表現するために使用することができませんでした。」

カテゴリー論による新しい数学の基礎づけ

「それでも、数学が全く新しい基盤を必要としていることを受け入れるのは、極めて難しいことです。ホモトピー型理論の進展に直接関わっている人たちでさえ、この考えに苦慮している人が少なくありませんでした。

それは、ZFCとカテゴリー理論という既存の数学の基礎が、非常に成功しているからです。新しい数学の基礎の候補として、カテゴリー理論の魅力を克服することは、私自身にとって最も困難なことでした。」

数学の新しい基礎づけ

Univalent Foundations

Univalent Foundationとコンピュータの利用

Voevodskyは、Univalence Axiomを中心とするHomotopy Type Theoryを武器に、集合論・トポス理論に代わるUnivalent Foundationという数学の新しい基礎付けを開始する。

興味深いのは、彼が、こうした理論展開を、数学論文ではなく、Coqのプログラムの形でGitHubで公開していることである。
https://github.com/vladimirias/Foundations/

Univalence Axiomについては、丸山の次の資料を参照されたい。「**「同じ」を考える -- 「型の理論」入門**」

https://www.marulabo.net/docs/type-theory/

解決すべき課題

「今日、私たちは2つの難しい条件を満たす問題に直面しています。

一方では、数学的証明のコンピュータ支援による検証の方法を見つけなければなりません。

これは、まずなによりも、数学における証明の概念が溶けていくのを止める必要があるからです。

その一方で、私たちは、数学と人間の直観の世界との密接なつながりを維持しなければなりません。

このつながりこそが、数学を前進させるものであり、私たちがしばしば数学の美しさとして経験するものなのです。」

https://goo.gl/ShdXzr

Univalent Foundationの原型

「Univalent Foundation（UF）は、いまだ不十分なものですが、この問題を解決するための、ソリューションです。

UFの原型は、次の3つの要素を組み合わせたものでした。:

- 数学とは、集合とその高次の類似物の構造に関する学問であるという見方。
- 集合の高次の類似物は、集合ベースの数学にホモトピー型として反映されるという考え方。
- このような高次の類似物上の構造に関する直感を、命題の排中律（LEM）、集合の選択公理（AC）、Univalence公理（UA）、リサイジング規則（RR）を用いて拡張した、マーティン＝レフの型理論（MLTT）を用いて定式化できるという考え方。」

その後、これらに追加された主な新概念は以下の通りである:

- LEMとACがなくても、多くの数学がMLTTで形式化できること、この2つの公理を除けば、新しい形の構成的数学の基礎が得られることを理解したこと、
- 古典数学は、この新しい構成的数学の部分集合として現れるという理解、
- UAで拡張されたMLTTは、この構成的数学のための不完全な形式化システムであり、UAをMLTTに統合して、より優れた計算特性を持つ新しい型理論を得ることができるはずだという理解。

Univalent Foundationの新しい形

「登場しつつあるUnivalent Foundation（UF）の新しい形は、次の要素の組み合わせとして見ることができます。

- 数学とは、集合とその高次の類似物の構造に関する学問であるという見方。
- 数学は、排中律 かつ/あるいは 選択公理をその前提の中に要求する諸結果からなる古典論的数学をその部分として持つ構成主義的なものであるという考え方。
- 集合の高次の類似物は、集合ベースの数学に構成主義的なホモトピー型 --今までのところでは、Cubical集合の用語でのみ定式化されうる新しい構成主義的ホモトピー型のオブジェクトとして反映されるという考え方。
- これらの高次の類似物に対する我々の直観は、Cubical型理論(CTT)によって定式化できるという考え方、」

UniMath

https://github.com/UniMath/UniMath

UniMath by Voevodsky https://goo.gl/ShdXzr

UniMath - a library of mathematics formalized in the univalent style" https://goo.gl/3sJr1M

UniMathは、全ての数学の分野を網羅しようとする

```
138
139  (** *** Canonical functions from [ empty ] and to [ unit ]
140
141  Definition fromempty  : ∏ X : UU , empty -> X. (* type thi
142  with \prod *)
143  Proof.
144    intro X.
145    intro H.
146    induction H.
147  Defined.
148
149  Arguments fromempty { X } _.
150
151  Definition tounit {X : UU} : X -> unit := λ _, tt.
152
```

Empty とunit の定義から始まる

master UniMath / UniMath / Foundations / UnivalenceAxiom.v

peterlefanulumsdaine Removed now-unnecessary unfolds in [Foundations]. Latest commit 567507e

5 contributors     

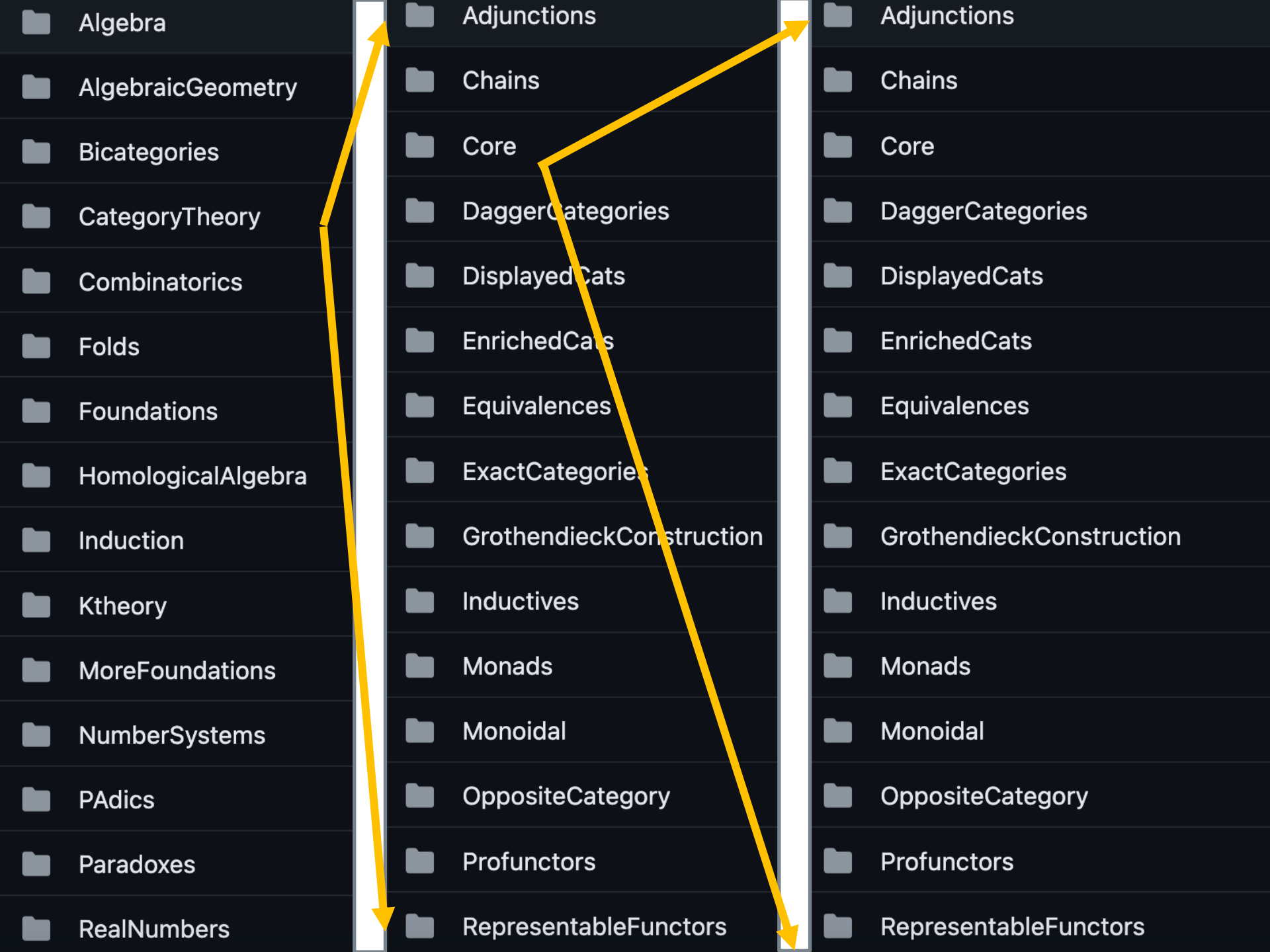
Univalent Axiom の定義

464 lines (373 sloc) | 17.6 KB Raw

```
1  (** The univalence axiom and its consequences *)
2
3  (**
4
5  In this file, we formulate univalence and its consequences, including functional
6  extensionality (the statement that functions with equal values are equal).
7
8  One approach would be to take univalence as the only axiom and to formulate
9  theorems proving the consequences, whose proofs would appeal to the
10 univalence axiom.  We adopt a different approach here, preferring also to
11 introduce axioms for various consequences of univalence.  This allows us to
12 measure how subsequent theorems depend on the axioms using the "Print
13 Assumptions" command of Coq.
```

```
(** Preliminaries  *)

Require Export UniMath.Foundations.PartB.

(* everything related to eta correction is obsolete *)

Definition eqweqmap { T1 T2 : UU } : T1 = T2 -> T1 ≃ T2.
Proof.
  intro e. induction e. apply idweq.
Defined.

Definition sectohfiber { X : UU } (P:X -> UU): (∏ x:X, P x) -> (hfiber (λ f, λ x, p

Definition hfibertosec { X : UU } (P:X -> UU):
  (hfiber (λ f x, pr1 (f x)) (λ x:X, x)) -> (∏ x:X, P x)
  := λ se , λ x:X, match se as se' return P x with tpair _ s e => (transportf P (to

Definition sectohfibertosec { X : UU } (P:X -> UU):
  ∏ a : (∏ x:X, P x), hfibertosec _ (sectohfiber _ a) = a.
Proof.
  apply idpath.
Defined.
```

Univalent Axiom の定義


Algebra
AlgebraicGeometry
Bicategories
CategoryTheory
Combinatorics
Folds
Foundations
HomologicalAlgebra
Induction
Ktheory
MoreFoundations
NumberSystems
PAdics
Paradoxes
RealNumbers
Adjunctions
Chains
Core
DaggerCategories
DisplayedCats
EnrichedCats
Equivalences
ExactCategories
GrothendieckConstruction
Inductives
Monads
Monoidal
OppositeCategory
Profunctors
RepresentableFunctors
Adjunctions
Chains
Core
DaggerCategories
DisplayedCats
EnrichedCats
Equivalences
ExactCategories
GrothendieckConstruction
Inductives
Monads
Monoidal
OppositeCategory
Profunctors
RepresentableFunctors

UniMathをどう使うか

「UniMathは構成的数学のライブラリですが、アルゴリズムに関するものではありません。ユーザーは、UniMathをコンパイルされたコードの形で見ることはありません。

その価値は、さまざまなユーザー入力に対応して、さまざまなアウトプットを提供する能力にあります。

UniMathは、ソースコードの形で存在します。
UniMathを「使う」ことは、既存のソースコードを拡張して新しいソースコードを書くことです。

それを行うためにには、以前に書かれたソースを読み、それをを理解しなければなりません。」

UniMathをどう使うか

「人によって、UniMathを使う目的は様々でしょう。

ある人は、複雑な数学の証明を検証したいかもしれません。

ある人は、数学のある古典的領域の問題を、構成主義的数学に枝分かれさせるのに使いたいと思うかもしれません。

ある人は、学生に厳密な数学とは何かを教えるための教材として使うかもしれません。」

https://www.math.ias.edu/vladimir/sites/math.ias.edu.vladimir/files/2016_07_14_Berlin_ICMS_short.pdf

UniMathとmathlib

UniMathの開発は、Veovodskyの死によっ、一旦中断するが、後継者たちによってその開発は継続される。

そのビジョンは、言語lean上で開発されているのmathlib とそのコミュニティによっても受け継がれていると思う。

“The Lean Mathematical Library”
https://arxiv.org/pdf/1910.09336.pdf
https://github.com/leanprover-community/mathlib

mathlibがカバーしている領域

~/miniF2F/_target/deps/mathlib/src

algebra
algebraic_geometry
algebraic_topology
analysis
category_theory
combinatorics
computability
control
data
dynamics
field_theory
geometry
group_theory
linear_algebra
logic
measure_theory
meta
model_theory
number_theory
order
probability_theory
representation_theory
ring_theory
set_theory
tactic
testing
topology

~/miniF2F/_target/deps/mathlib/src/algebra

```
abs.lean
abs.olean
add_torsor.lean
add_torsor.olean
algebra
associated.lean
associated.olean
big_operators
bounds.lean
bounds.olean
category
char_p
char_zero.lean
char_zero.olean
continued_fractions
covariant_and_contravariant.lean
covariant_and_contravariant.olean
indicator_function.lean
indicator_function.olean
invertible.lean
invertible.olean
is_prime_pow.lean
iterate_hom.lean
iterate_hom.olean
lie
linear_recurrence.lean
linear_recurrence.olean
module
monoid_algebra
ne_zero.lean
ne_zero.olean
non_unital_alg_hom.lean
non_unital_alg_hom.olean
opposites.lean
```

mathlibの利用例
Open-AI Theorem Prover

Mathlibを本体とするmathlib-train は、量的にも質的にも、Open-AI Theorem Proverが数学的問題を解くために学習した訓練用データの本体である。

量的には、

- miniF2F-curriculum　327
- synth-ineq　5,600
- mathlib-train　25,000 (25K)

因果はめぐる

ChatGPTの成功の影に隠れて、ほとんど注目されることはなかったのだが、OpenAIの技術的トップのIlya自身の肝煎りで、こうしたプロジェクトが進行していたことに、僕は注目している。

それは、大規模言語モデルによる「ことばと意味」へのアプローチと、Veovodskyの「正しい数学的証明」へのアプローチの、僕が知る限りでは、最初の接点であった。

この「ファースト・コンタクト」は失敗に終わったのだが、この失敗から学ぶべきものは多いと僕は考えている。