

定理証明支援系 Coq を用いたアクチュアリー数学の形式化

SOMPO ひまわり生命保険株式会社

伊藤 洋介*

2021 年 3 月 13 日

概要

定理証明支援系とは、形式的な表現を用いて数学の証明をコンピュータ上で行うツールである。近年はさまざまな定理証明支援系が開発され、現代数学の形式化やプログラムの検証に利用されている。本稿では代表的な定理証明支援系である Coq を紹介するとともに、それを使用したアクチュアリー数学の形式化について報告する。最後に、Coq や本研究の産業界への応用可能性について考察する。

1 定理証明支援系 Coq

20 世紀後半から現代にかけて、数学の証明をコンピュータ上に実装する研究が盛んに行われている。そうしたツールは定理証明支援系と呼ばれ、これまでに数々のソフトウェアが開発されてきた。本節では代表的な定理証明支援系である Coq を紹介するとともに、数学の形式化の意義や現状について論じる。

1.1 定理証明支援系とは

1.1.1 コンピュータを用いた定理証明

現在の文部科学省の指導要領によれば、日本の義務教育で初めて証明に触れるのは中学校第 2 学年である。平面図形の証明においては、対頂角が等しいことや、平行ならば同位角が等しいこと等の性質を議論の出発点として、論理的な推論を用いて新たな図形の性質を導く。このように、“与えられた命題から、論理的形式に頼って推論を重ね、結論を導き出すこと”を演繹と呼ぶが、平面図形の様々な性質を演繹的に導く方法論は、古代ギリシアの数学者 Euclid によって編纂されたいわゆる“ユークリッド原論”によって一応の完成を見、その後長い間西欧諸国における理論体系の手本として不動の地位を保ってきた。

図形を研究する幾何学において新しい性質を発見するには、種々の具体例を通じて幾何学的な直感を養うことが重要であろう。一方で、図形の性質を証明する場合には、直感に頼らずあくまで論理的に推論を積み重ねていくことが求められる。もし途中で論理的に飛躍した場合、最終的に誤った結論を導いてしまう恐れがある。

数学における“証明”については 19 世紀頃から活発な研究が開始され、現代では証明自体を数学的対象として研究する“証明論”という数理論理学の一分野がある。証明論においては、英語や日本語といった自然言語で書かれた数学の証明を記号論理の枠組みで形式化する。形式的な証明はごく限られた推論規則によって機械的に行われるため、適切なプログラミングを行うことでコンピュータ上に実装することが可能である。一つ一つの論証のステップをコンピュータが確かめることで、数学における証明に誤りがないことを保証することがで

* 本論文の内容は全て筆者個人の見解であり、所属する組織としての見解を示すものではない。

きる。このような定理証明の補佐ツールが定理証明支援系である。

数学を形式化する試みについては、数学者によって受け止め方は様々である。ラドバウド大学の Freek Wiedijk は AMS (American Mathematical Society) が発刊する雑誌 Notices において次のように述べている [26, pp. 1413–1414; 拙訳]。

数学においては今までに 3 度の革命があった:

- 紀元前 4 世紀にギリシア人によって**証明**が導入され、Euclid の“**原論**”により完結されたこと。
- 19 世紀に数学に**厳密性**が導入されたこと。この時期には、当時厳密性に欠けていた微分積分学が Cauchy 等により厳密化された。また、同時期には Frege により数理論理学が、Cantor により集合論が発展した。
- 20 世紀後半から 21 世紀初めにかけて、**形式的数学**が導入されたこと。

多くの数学者はこの第 3 の革命が既に起こっていることに気づいておらず、またこの革命が必要であるということにさえ反対するだろう。しかし、今後数世紀の間に数学者は我々の時代をこの革命の時代として振り返るだろう。その未来においては、多くの数学者は数学が完全に形式化されない限りそれを信頼の置けるものとは考えないだろう。

In mathematics there have been three main revolutions:

- The introduction of *proof* by the Greeks in the fourth century BC, culminating in Euclid’s *Elements*.
- The introduction of *rigor* in mathematics in the nineteenth century. During this time the nonrigorous calculus was made rigorous by Cauchy and others. This time also saw the development of mathematical logic by Frege and the development of set theory by Cantor.
- The introduction of *formal mathematics* in the late twentieth and early twenty-first centuries.

Most mathematicians are not aware that this third revolution already has happened, and many probably will disagree that this revolution even is needed. However, in a few centuries mathematicians will look back at our time as the time of this revolution. In that future most mathematicians will not consider mathematics to be definitive unless it has been fully formalized.

通常は紙とペンで行われる数学をコンピュータに実装することに違和感を覚える人は多いかもしれない。しかし、近年のコンピュータの進展や人工知能の著しい発展を目の当たりにする時、これを数学の推論に応用しようと思えることは、ごく自然なことだと思われる。

1.1.2 定理証明支援系の紹介

定理証明支援系はこれまで多くの種類が開発されており、それぞれ独自の特徴を備えている。ここでは本研究で使用する Coq を中心に解説する。それに加え、その他の代表的な定理証明支援系として Isabelle と Lean を紹介する。

■Coq Coq の基盤となる理論 Calculus of Constructions は 1980 年代に Thierry Coquand と Gérard Huet によって考案された [6]。この理論はその後の様々な拡張を経て、1989 年に Coq の名が冠せられることとなった [24, p. 579]。Coq はその後も長きにわたり世界的な影響を及ぼし続け、2013 年には世界最大の計算科学系の学会である ACM (Association for Computing Machinery) から ACM ソフトウェアシステム賞を受賞している。

Coq の公式 Web サイト

<https://coq.inria.fr/>

には Coq の概要, インストール方法, リファレンスマニュアル [24], コミュニティへの案内等が掲載されている。また, インストールをせずに Web 上で Coq を体験できる jsCoq ^{*1}も利用可能である。Coq にはあらかじめ基礎的な数学が形式化されており, 標準ライブラリ^{*2}に格納されている。具体的には

- **Arith**: 自然数のライブラリ
- **ZArith**: 整数のライブラリ
- **QArith**: 有理数のライブラリ
- **Reals**: 実数・微分積分学のライブラリ
- **Sets**: 集合論のライブラリ

等がある。これらをインポートすれば, 既に形式化された定義や定理についてはそのまま使用することができる。

標準ライブラリに格納されていない理論 (群論等) については別途形式化する必要がある。Coq では今までに多くの貢献者によって様々な理論が形式化されており, それらは主に Coq Package Index^{*3} から利用可能である。中でも SSReflect/MathComp^{*4}は Coq の有用な拡張として広く利用されており, 日本においても入門書 [12] が出版されている。本研究では Coq の標準ライブラリに加えて SSReflect/MathComp および Coquelicot ^{*5}を用いる。

■**Isabelle** Isabelle は 1986 年に Lawrence C. Paulson により導入された定理証明支援系である [22]。公式 Web サイト

<https://isabelle.in.tum.de/index.html>

からインストール可能であり, リファレンスマニュアルや学習資料等も閲覧可能である。Archive of Formal Proofs^{*6}には Isabelle により形式化された結果が多数収録されている。本稿で紹介している定理証明支援系の中では確率論の形式化が最も進んでおり, 2017 年には Jeremy Avigad 等により中心極限定理が Isabelle により形式化されている [2]。

■**Lean** Lean は 2013 年にマイクロソフトリサーチの Leonardo de Moura によってプロジェクトが開始された新しい定理証明支援系である [7]。公式 Web サイト

<https://leanprover.github.io/>

からインストール可能であるが, オンラインで実行できる JavaScript 版も用意されている^{*7}。インペリアル・カレッジ・ロンドンの Kevin Buzzard が現代数学の形式化に積極的に関与しており, AMS が発刊する雑誌

^{*1} <https://jscoq.github.io/>

^{*2} <https://coq.inria.fr/distrib/current/stdlib/>

^{*3} <https://coq.inria.fr/packages.html>

^{*4} <https://math-comp.github.io/>

^{*5} <http://coquelicot.saclay.inria.fr/>

^{*6} <https://www.isa-afp.org/>

^{*7} <https://leanprover.github.io/live/latest/>

Notices において Lean を紹介している [5]. Lean のライブラリである `mathlib` ^{*8}には専門的な数学も数多く投稿されており [20], コミュニティも急速に拡大している. 成長著しい言語であるが, 発展段階ということもあり後方互換性は必ずしも保証されない.

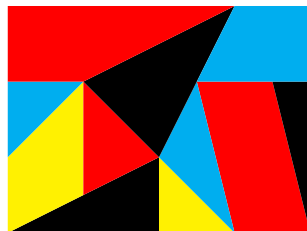
1.1.3 定理証明支援系の応用例

定理証明支援系は今までに多くの応用が試みられ, 数々の実績を残してきた. ここではその一部を紹介する.

■**四色定理** Coq が重要な役割を果たした数学の難問として四色問題が挙げられる. 現在では既に証明が得られているため四色定理と呼ばれるが, その主張は次のようなものである.

定理 (四色定理 [1]). 平面上の地図は 4 色で塗り分けられる.

詳細に述べると, “いかなる地図も, 隣接する領域が異なる色になるように塗るには 4 色あれば十分である” ということである.



四色定理は 1976 年に Kenneth Appel と Wolfgang Haken によって証明が与えられたが, それはコンピュータプログラムを用いて膨大な場合分けを行うものであったため物議を醸した. その後, より改良された証明が発見されたものの, それらは依然としてコンピュータに依存したものであり, プログラム自体にバグがないことを検証するのが困難であった. 2000 年になり Georges Gonthier は四色定理の Coq による形式化を開始し, 約 5 年もの歳月を経て完成に至った [10]. これは四色定理の主張と証明を全て Coq で記述したということであり, プログラムの正当性は疑いの余地がなくなった. この応用例は, 人間による査読が困難な証明の検証にコンピュータが役立つことを示す重要な事例であると言えよう.

■**seL4** 定理証明支援系は数学の定理証明だけでなく, 実用的なシステムプログラムの品質確保にも応用されている. 具体的には, システムが所望の仕様を満たすことを定理証明支援系により証明するのである. 中でも seL4 は定理証明支援系による形式検証を経た世界初の OS (オペレーティング・システム) カーネルとして知られ [18], 高い信頼性を獲得している. なお, この形式検証には Isabelle が利用された.

1.2 数学の形式化

本節では, 数学を形式化するとはどのようなことなのか, 具体例を交えながら説明するとともに, 数学を形式化することの意義について論じる. また, 現代数学の形式化が現時点でどの程度進んでいるのかについても触れる.

^{*8} https://leanprover-community.github.io/mathlib_docs/

1.2.1 形式的証明とは

数学の論理については 19 世紀末から 20 世紀初頭にかけて活発な研究がなされ、現代数学は集合論に基づく記号論理で記述できるという認識が広まっていった。記号論理とは、論理学を形式的な記号で記述する方法のことである。

具体的な例を命題論理で見てみよう。表 1 は現代の論理学で一般的に用いられる記法である。命題^{*9}や論理記号を組合せて得られる文を論理式と呼ぶ。例えば、自営業を営んでいる太郎という人物がいたとして、彼の

表 1 命題論理の記号

論理記号	意味
$\neg P$	P でない
$P \wedge Q$	P かつ Q
$P \vee Q$	P または Q
$P \rightarrow Q$	P ならば Q
$P \leftrightarrow Q$	P の場合かつこの場合に限り Q

iDeCo への加入資格を考えてみる。論文執筆時点において、第 1 号被保険者の iDeCo への加入資格は以下のようになっている^{*10}。

- 満 20 歳以上 60 歳未満である。
- 国民年金保険料を納付している。
- 農業者年金基金に加入していない。

今、命題 P_1, P_2, Q, R を次で定める。

P_1 : 太郎は満 20 歳以上である。

P_2 : 太郎は満 60 歳未満である。

Q : 太郎は国民年金保険料を納付している。

R : 太郎は農業者年金基金に加入している。

すると、太郎の加入資格は次のように表せる。

$$(P_1 \wedge P_2) \wedge Q \wedge (\neg R)$$

このような単純な例では記号化の恩恵がわかりにくいだが、さらに複雑な条件になった場合、論理式は主張を簡潔かつ正確に表現する手段として非常に有効である。

また、記号化することで論理式に対する機械的な操作が可能になるという利点もある。例えば、太郎に加入資格がないとはどういうことか、具体的に見てみよう。それには先ほどの論理式を否定すればよい。

$$\neg((P_1 \wedge P_2) \wedge Q \wedge (\neg R))$$

^{*9} 数学においては通常、“命題”とは証明された論理式を表すが、ここでは単なる主張という意味で用い、その真偽にかかわらず“命題”と呼ぶ。

^{*10} 説明の都合上一部簡略化している。

ここで論理学における De Morgan の法則

$$\neg(A \wedge B) \iff (\neg A) \vee (\neg B)$$

および二重否定律^{*11}

$$\neg(\neg C) \iff C$$

を用いれば,

$$\begin{aligned}\neg((P_1 \wedge P_2) \wedge Q \wedge (\neg R)) &\iff (\neg(P_1 \wedge P_2)) \vee (\neg Q) \vee (\neg(\neg R)) \\ &\iff ((\neg P_1) \vee (\neg P_2)) \vee (\neg Q) \vee R\end{aligned}$$

と変形することができる. すなわち, 太郎に iDeCo の加入資格がないとは次のいずれかを満たすということである.

- 太郎は満 20 歳未満である.
- 太郎は満 60 歳以上である.
- 太郎は国民年金保険料を納付していない.
- 太郎は農業者年金基金に加入している.

以上が命題論理の例であるが, 現代数学を記述するには命題論理だけでは不十分であり, それを拡張した述語論理の枠組みが必要である. そこで, 表 2 に掲載した新たな記号 \forall と \exists を導入する. 例えば先ほどの命題

表 2 述語論理の記号

論理記号	意味
$\forall x L(x)$	任意の x に対し $L(x)$
$\exists x L(x)$	ある x に対し $L(x)$

P_1 : 太郎は満 20 歳以上である.

を主語 “太郎” と述語 “満 20 歳以上である” に分解することを考えよう. 今,

$L_1(x)$: x は満 20 歳以上である.

と定めれば, 命題 P_1 は $L_1(\text{太郎})$ と表すことができる. このように述語部分を抽出すると, 具体的な対象 “太郎” に捉われずに次のような主張を展開することが可能になる.

$\exists x L_1(x)$: ある x に対し, x は満 20 歳以上である.

これはつまり “少なくとも 1 人は満 20 歳以上の人が存在する” ということを意味する.

以上の枠組みは現代数学を記述する基礎的な言語として使用できる. 例えば “4 の倍数は偶数である.” という命題を表すには, 例えば二項関係記号 $x \mid y$ を “ x は y を割り切る.” と定めた上で

$$\forall x(4 \mid x \rightarrow 2 \mid x)$$

^{*11} 二重否定の除去は直観論理では認められていないが, ここでは通常の数学で使用される古典論理を前提として議論を進める.

とすればよい。

それでは、このように形式化された命題を“証明”するにはどうすればよいだろうか。記号論理学においては、理論の前提となる“公理”と証明を進めるための“推論規則”が規定されており、それらを用いて命題を導くことを“証明”と呼ぶ。例えば、論理記号 \wedge に関する推論規則は次の2つである。

\wedge 導入則：命題 P と命題 Q の両方が得られているとき、命題 $P \wedge Q$ を得る。

\wedge 除去則：命題 $P \wedge Q$ が得られているとき、命題 P と命題 Q を得る。

また、論理記号 \rightarrow に関する推論規則は次の2つである。

\rightarrow 導入則：命題 P を仮定して命題 Q が得られたとき、命題 $P \rightarrow Q$ を得る。

\rightarrow 除去則：命題 $P \rightarrow Q$ と命題 P の両方が得られているとき、命題 Q を得る。

これらを用いると、三段論法

$$((P \rightarrow Q) \wedge (Q \rightarrow R)) \rightarrow (P \rightarrow R)$$

は次のようにして証明できる。

- 1: $(P \rightarrow Q) \wedge (Q \rightarrow R)$ を仮定する。
- 2: P を仮定する。
- 3: 1行目より $P \rightarrow Q$ と $Q \rightarrow R$ を得る。(\wedge 除去)
- 4: 2行目と、3行目の $P \rightarrow Q$ から Q を得る。(\rightarrow 除去)
- 5: 4行目と、3行目の $Q \rightarrow R$ から R を得る。(\rightarrow 除去)
- 6: 2行目と5行目から、 $P \rightarrow R$ を得る。(\rightarrow 導入)
- 7: 1行目と6行目から、 $((P \rightarrow Q) \wedge (Q \rightarrow R)) \rightarrow (P \rightarrow R)$ を得る。(\rightarrow 導入)

この例からもわかるように、形式化された数学においては許された推論規則以外証明に用いてはならない。記号論理に初めて触れる者には不便だと感じられるかもしれないが、推論規則を限定することで論理の飛躍を防止することができ、またコンピュータへの実装も容易になるのである。

なお、議論の出発点となる公理として何を採用するかは理論体系により異なる。現代数学は集合論を基礎に展開できると考えられているが、その際に仮定する公理としては Zermelo-Fraenkel の公理系が有名である。ただし、Coq で数学を基礎付ける場合、一般には Zermelo-Fraenkel 集合論と異なる枠組みを用いる。数学の基礎付けについては専門的な内容となるためこれ以上は立ち入らない。前述の通り、Coq の標準ライブラリには既に初等的な数学が形式化されているため、定理証明支援系が基礎としている公理については詳細を把握していなくても応用上問題ないと思われる。

1.2.2 形式化の意義

前節で形式的証明について概説したが、ここで定理証明支援系を用いた形式化の意義について改めて考えてみようと思う。

■無謬性の保証 形式化された数学においては、前述の通り限られた推論規則でのみ証明を進めることができる。これを定理証明支援系で実装することができれば、その証明には一切の論理的飛躍のないことが確かめられる。

読者の中には、数学はそもそも厳密な学問なのだから今更それを確認しても無意味だと思われるかもしれない。しかし、最先端の数学研究においては査読を経た論文に誤りや論理の飛躍が見つかることは実際にある。

例えば Sébastien Gouëzel と Vladimir Shchur は学術誌 Journal of Functional Analysis において、2013 年に証明された定理を Isabelle で形式化する際に論理の飛躍を発見したことを報告している [11]。近年の数学は高度に細分化されていることに加え、他の研究者により証明された結果に依存して新たな定理を証明することも多い。このことは、新たな論文の査読において誤りを見つけること自体が困難になっていることを意味する。もし定理証明支援系が発展して査読に利用されるようになれば、誤りを検出する精度が飛躍的に向上することになる。そもそも著者自身が論文を投稿する際に定理証明支援系による形式化を済ませていれば、それは査読を経るよりも遥かに高い信頼性を獲得することになるだろう。

■システムの形式的検証 定理証明支援系はソフトウェアの品質確保にも利用可能である。例えば、コンピュータプログラムが満たすべき性質を形式論理で記述し、ソースコードがその仕様を満たすことを定理証明支援系で証明する、といった方法がある。

形式的に検証されたソフトウェアの信頼度は非常に高い。仮に数多くのサンプルテストをクリアしたソフトウェアであったとしても、想定される全てのパターンを検証しない限りバグの存在は否定できない。しかし、定理証明支援系を用いると仕様通りにプログラムが動作することを数学的に証明できるため、形式化された部分に関してはバグのないことが保証される。

実際、形式的な証明は情報セキュリティで最も高いレベルの国際認証を得る際に必要とされている。“コモンクライテリア”とは情報システム等のセキュリティを評価するための基準を定めた国際規格であるが、そこで定める保証評価レベル (EAL: Evaluation Assurance Level) は EAL1 から EAL7 までの 7 段階に分かれており、最も高いレベルの EAL7 の認証を取得するには形式的検証を経ることが求められている。

EAL7 (形式的検証済み設計及びテスト)

リスクが非常に高いか、高い資産価値により、さらに高い開発コストが正当化される場合に適用される。EAL7 は、EAL6 の保証に加え、数学的検証を伴う形式的表現と対応、広範囲のテストを使用する包括的分析を要求する。[14]

ビッグデータや人工知能の発展が著しい現代、ソフトウェアの誤作動が社会に与える影響はますます大きくなってきている。定理証明支援系を用いた形式的検証は一切のバグを除去する手段として今後も期待が高まっていくものと考えられる。

■自動推論 定理証明支援に隣接した分野として自動定理証明がある。Coq を始めとした定理証明支援系では、形式的証明を行う際には原則としてユーザが証明のプロセスを指示する。それに対し、自動定理証明機は目的の定理を与えるだけでコンピュータが自動的に推論を行い、形式的証明を完成させる。

自動定理証明も世界的によく研究されており、中でも William McCune による自動定理証明機 EQP を用いた Robbins 予想^{*12}の解決は、コンピュータが人間の助けを借りずに数学の問題を解いた成功例として知られる [21]。ただ、数学全般で見れば、現時点においてコンピュータは数学者に取って代わるような水準には達していない。現在、自動推論の質やスピードを向上させる研究が行われているが、その方法の一つとして定理証明支援系で形式化された証明を自動定理証明機に学習させる方法がある。Cezary Kaliszyk と Josef Urban は Kepler 予想^{*13}を形式化する Flyspeck プロジェクトによって得られた大量の定義や補題からなるライブラリを、自動定理証明への機械学習に利用している [16]。この例からもわかるように、形式化された数学は人工知能

^{*12} “Robbins 代数がブール代数になる”という予想。20 世紀前半に Herbert Robbins によって提示された。

^{*13} “無限の空間において同半径の球の敷き詰めを行ったとき、最密充填は面心立方格子である”という予想。Johannes Kepler により 1611 年に予想された。

の学習に活用できる貴重な財産なのである。

1.2.3 数学の形式化の進捗

現代数学の形式化はさまざまな定理証明支援系で行われている。表 3 は Thomas C. Hales が AMS の雑誌 Notices で紹介した主要な定理の形式化に関する一覧である。この記事が書かれたのは 2008 年であり、現在で

表 3 形式化された証明の例 [13, p. 1372; 拙訳]

年	定理	定理証明支援系	実装者
1986	第一不完全性定理	Boyer-Moore	Shankar
1990	平方剰余の相互法則	Boyer-Moore	Russinoff
1996	微分積分学の基本定理	HOL Light	Harrison
2000	代数学の基本定理	Mizar	Milewski
2000	代数学の基本定理	Coq	Geuvers et al.
2004	四色定理	Coq	Gonthier
2004	素数定理	Isabelle	Avigad et al.
2005	Jordan 曲線定理	HOL Light	Hales
2005	Brouwer の不動点定理	HOL Light	Harrison
2006	Flyspeck I	Isabelle	Bauer-Nipkow
2007	留数定理	HOL Light	Harrison
2008	素数定理	HOL Light	Harrison

はさらに多くの定理が形式化されている。直近の進捗を把握する目安として、Freek Wiedijk が自身のホームページに掲載している Formalizing 100 Theorems がある^{*14}。ここには数学の主要な 100 の定理が掲載され、現時点でどの程度形式化されたかがまとめられている。論文執筆時点での進捗率は 97% となっており、多くの定理が形式化されていることがわかる。数学の形式化は現役の数学者だけでなく学生を含めた世界中の貢献者によって進められており、古典的な結果だけでなく、最先端の数学についても形式化が意欲的に試みられている。

しかしながら、アクチュアリー数学については今まで形式化が行われていない。本研究は発展の著しい定理証明を生命保険数学に応用する初めての試みである。

2 アクチュアリー数学の形式化

本研究では定理証明支援系 Coq を用いて生命保険数学の基本的な定義や公式を形式化し、その内容を Actuary パッケージとしてリリースした。本節ではまず Coq の文法を簡単に紹介した上で、Actuary パッケージの大まかな内容や応用例について解説する。最後に、Actuary パッケージに関する今後の課題について論じる。

^{*14} <http://www.cs.ru.nl/F.Wiedijk/100/index.html>

2.1 Coq の文法

Coq を用いて数学の定理を証明するには、まず証明したい命題を論理式で記述し、タクティクと呼ばれる命令で証明を組み上げていくことになる。Coq は対話型の証明支援系であり、証明を完成させるにはステップごとにユーザから指示を送ることになる。通常のプログラミング言語と同じく、Coq を正しく動作させるには決められた文法に従ってスクリプトを記述する必要がある。

まず第一に、Coq で命題を記述するには第 1.2.1 節で紹介した論理式を Coq における記法 (表 4) に従って入力することになる。前述の三段論法

表 4 論理式の記法

論理記号	Coq の記法
\neg	<code>~</code>
\wedge	<code>/\</code>
\vee	<code>\ </code>
\rightarrow	<code>-></code>
\leftrightarrow	<code><-></code>
\forall	<code>forall</code>
\exists	<code>exists</code>

$$((P \rightarrow Q) \wedge (Q \rightarrow R)) \rightarrow (P \rightarrow R)$$

を Coq で表記すれば、次のようになる。

$$((P \rightarrow Q) /\ (Q \rightarrow R)) \rightarrow (P \rightarrow R)$$

続いて三段論法の証明を Coq で形式化することを考える。結論から述べると、次の四角内部がその証明の一例である。

```

1 Theorem syllogism : forall (P Q R : Prop), ((P -> Q) /\ (Q -> R)) -> (P -> R).
2 Proof.
3   intros P Q R.
4   intro H_PQR.
5   intro H_P.
6   destruct H_PQR as [H_PQ H_QR].
7   apply H_QR.
8   apply H_PQ.
9   assumption.
10  Qed.
```

Coq で三段論法を証明する場合は上記のスクリプトを 1 行ずつ実行していけばよい。無事に最後までエラーがなく実行できれば証明が完了したことを意味する。もし途中で推論に誤りがあったり論理の飛躍があったりすると、Coq はエラーを出力するようになっている。

まず 1 行目で証明したい定理をコマンド `Theorem` の後に続けて宣言している。ここで `syllogism` と書かれている部分は定理の名前であり、原則として自由に設定できる^{*15}。コロンの後に証明したい論理式を記述する。なお、`forall (P Q R : Prop)`、の部分は“任意の命題 P, Q, R に対して”と読む^{*16}。Coq は定理を受け取ると証明モードに入り、以下のような出力をする。

```
1 subgoal

=====
forall P Q R : Prop, (P -> Q) /\ (Q -> R) -> P -> R
```

出力画面の二重線はゴールラインと呼ばれる。ゴールラインより下には今証明すべき命題が表示され、これはサブゴールと呼ばれる。次に 3 行目 `intros P Q R.` を入力すると出力画面が以下のように変わる。

```
1 subgoal

P, Q, R : Prop
=====
(P -> Q) /\ (Q -> R) -> P -> R
```

ゴールラインより上には証明に使用できる前提や変数が表示され、これらはコンテキストと呼ばれる。今、命題 P, Q, R がサブゴールからコンテキストに移されたが、この操作は“任意の命題 P, Q, R を取って固定する”ものと解釈できる。次の 4 行目 `intro H_PQR.` を実行すると、出力画面は以下ようになる。

```
1 subgoal

P, Q, R : Prop
H_PQR : (P -> Q) /\ (Q -> R)
=====
P -> R
```

この命令はサブゴールの十分条件をコンテキストに移す命令である^{*17}。つまり“($P \rightarrow Q$) \wedge ($Q \rightarrow R$) を仮定する”ことを意味する。続く `intro H_P.` も同様であり、出力画面は次の通りである。

```
1 subgoal

P, Q, R : Prop
H_PQR : (P -> Q) /\ (Q -> R)
H_P : P
=====
R
```

さて、今求められているのは“($P \rightarrow Q$) \wedge ($Q \rightarrow R$) と P が成り立つことを前提とした上で R を示す”

^{*15} ここでは“三段論法”を意味する英単語 “syllogism” を用いた。

^{*16} 型 `Prop` は“命題”を意味する英単語 “proposition” に由来する。なお、通常の数学では命題を全称記号で量化することはないと思われるが、Coq は高階述語論理を扱えるのでこのような記述が可能である。

^{*17} ここで `H_PQR` というのは仮定の名前であり、原則として自由に名付けることができる。

ということである。仮定 H_{PQR} はそのままでは使えないので \wedge の除去を行う。その命令が 6 行目の `destruct H_PQR as [H_PQ H_QR]`。である。これは仮定 H_{PQR} を分解して左側を H_{PQ} と、右側を H_{QR} と名付ける操作である。

```
1 subgoal

P, Q, R : Prop
H_PQ : P -> Q
H_QR : Q -> R
H_P : P
=====
R
```

続く 7 行目の `apply H_QR.` を実行すると、出力画面は次のようになる。

```
1 subgoal

P, Q, R : Prop
H_PQ : P -> Q
H_QR : Q -> R
H_P : P
=====
Q
```

サブゴールが R から Q に変わっている。この操作を解釈するならば、“今仮定 $Q \rightarrow R$ があるので、 R を示すには Q を示せばよい” ということになる。Coq を始めとする定理証明支援系では、このようにサブゴールを変形して証明を進める方式を採用している。我々が数学の証明を書くときは仮定から順番に演繹を行うのが普通なので、結論を変形していく操作には違和感を覚えるかもしれないが、慣れれば特に難しいものではない。次の 8 行目 `apply H_PQ.` も同様であり、サブゴールが Q から P に変わる。

```
1 subgoal

P, Q, R : Prop
H_PQ : P -> Q
H_QR : Q -> R
H_P : P
=====
P
```

ここまで来ればサブゴール P はコンテキストに含まれる P そのものである。コンテキストから直ちにサブゴールが得られる場合、9 行目のように `assumption.` と入力すればよい。すると出力結果は以下のようになる。

```
No more subgoals.
```

つまり “もう証明すべきサブゴールは残っていない” ということであり、あとは証明が完了したことを示す

Qed. を入力すれば証明モードを抜ける. このようにして証明された定理は Coq に登録され, 以後は定理 `syllogism` を証明の中で使用できるようになる. なお, Coq へ入力した `intro`, `destruct` 等の命令はタクティクと呼ばれる. 直訳すれば“戦術”となるが, サブゴールを解消するための戦術という意味でこのように呼ばれているものと思われる.

通常の数学での証明と同じく, 証明の方法は一通りではない. 今の例では例示のために証明のステップを意図的に細かく記載したが, この程度の推論であれば Coq は自動的に証明を組み上げることができる.

```

1 Theorem syllogism' : forall (P Q R : Prop), ((P -> Q) /\ (Q -> R)) -> (P -> R).
2 Proof.
3   intuition.
4   Qed.

```

このように Coq には若干ではあるが自動証明機能が実装されている. 自動証明が可能な条件はあるものの, 適切なタイミングで使えば形式化の労力を削減する強力な手段となる.

2.2 Actuary パッケージの概要

前節で紹介したような形式的証明を行って得られた補題・定理群をライブラリまたはパッケージと呼ぶ. 筆者は Coq を用いて生命保険数学の記号や基本的な公式を形式化し, 一連の結果を Actuary パッケージとして GitHub ^{*18}に公開した.

<https://github.com/Yosuke-Ito-345/Actuary>

本パッケージの Version 1.0 は 2021 年 2 月 14 日にリリースされたが, そのファイル構成は表 5 のようになっている ^{*19}. 形式化する生命保険数学の内容は日本のアクチュアリー試験の教科書 [8, 9] やアメリカの SOA

表 5 Actuary パッケージの概要

ファイル名	内容
Basics.v	生命保険数学の形式化に使用する基本的な数学の補題を集めたもの
Interest.v	確定年金現価等の金利に関する記号や公式をまとめたもの
LifeTable.v	生命表の定義や基本的な性質をまとめたもの
Premium.v	生命年金現価や保険料に関する記号や計算式および公式をまとめたもの
Reserve.v	責任準備金の記号や公式をまとめたもの
Examples.v	Actuary パッケージの応用例を紹介したもの

(Society of Actuaries) 試験の教科書 [15] および参考書 [19] を始め, インターネット上に公開されている講義ノート [17, 23] を参考にした.

■ **アクチュアリー記号** 生命保険数学で用いられるアクチュアリー記号も Coq に実装している. アクチュアリー記号は補助記号が多く振られる複雑な記法であり, それをテキスト形式で直観的にわかりやすく表現する

^{*18} GitHub とはソフトウェア開発のために利用されるインターネット上のプラットフォームであり, 定理証明支援系 Coq の開発も GitHub で行われている.

^{*19} Coq のファイルは拡張子を .v とする.

のはなかなか難しいが, 試行錯誤を経て表 6 のような形に落ち着いた.

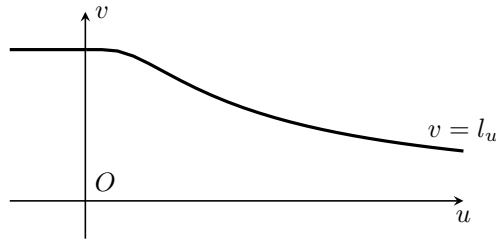
表 6 アクチュアリー記号 (抜粋)

意味	アクチュアリー記号	Coq の記法
生存数	l_x	$\backslash l_x$
死亡数	d_x	$\backslash d_x$
生存確率	${}_t p_x$	$\backslash p_ \{t \ \& \ x\}$
死亡確率	${}_f {}_t q_x$	$\backslash q_ \{f t \ \& \ x\}$
定期保険の現価	$A_{x:\overline{n}}^1$	$\backslash A_ \{x' 1:n\}$
生命年金現価	$\ddot{a}_{x:\overline{n}}$	$\backslash a' _ \{x:n\}$
定期保険の平準純保険料	$P_{x:\overline{n}}^1$	$\backslash P_ \{x' 1:n\}$
定期保険の責任準備金	${}_t V_{x:\overline{n}}^1$	$\backslash V_ \{t \ \& \ x' 1:n\}$

■**生命表の定義** 生命表 l_x は生命保険数学における種々の計算の基礎であるが, 数学的には実数値関数と考えるのが自然である. Coq へ実装する際は, 一般的な生存関数 (survival function) の定義も参考にして次のように定式化した.

定義. 実数上の実数値関数 l が生命関数であるとは, 次の 4 条件を全て満たすことを言う^{*20}:

1. $l_0 > 0$,
2. 任意の負の実数 u に対し $l_u = l_0$ が成り立つ,
3. $\lim_{u \rightarrow \infty} l_u = 0$,
4. l は広義単調減少である.



生命関数は 0 以上の実数に対して定義されればよいが, Coq では実数全体で関数を定義する方が扱い易いため, 便宜上負値も定義域に含めている. 実際の **Actuary** パッケージにおける生命関数の定義は次の通りである.

```

1  (* life table *)
2  Record life : Type := Life {
3    l_fun :> R -> R;
4    l_0_pos : l_fun 0 > 0;
5    l_neg_nil : forall u:R, u <= 0 -> l_fun u = l_fun 0;
6    l_infty_0 : is_lim l_fun p_infty 0;
7    l_decr : decreasing l_fun

```

^{*20} 一般に, 関数 f の x における値を f_x のように表す.

8

}.

文法の詳細な説明は省略するが、ここに現れる `Record` は一般のプログラミング言語で言うレコード型と同じ意味合いである。スクリプト中の `R` は実数型を表しており、3 行目の `R → R` は実数から実数への関数を意味する。4 行目から 7 行目は前述の定義に示した 4 条件を形式化したものである。

なお、この定義は最終年齢が ∞ になる場合も含んでいるし、微分可能性も仮定していない。その意味で非常に広い定義であるが、生命年金現価等を定義する上では有限和の方が扱い易いので次の性質を仮定する。

- 定義.**
1. 生命関数 l が有限であるとは、 $l_x = 0$ となる自然数 x が存在することを言う。
 2. 有限な生命関数 l に対し、 $l_x = 0$ となる最小の自然数 x を l の最終年齢と呼ぶ。

これは `Actuary` パッケージでは次のように形式化されている。

```
1 Definition ages_dead (l:life) : Ensemble nat := fun x:nat => \l[l]_x = 0.
2 Definition l_finite (l:life) := exists x:nat, (ages_dead l x).
3 Definition ult_age (l:life) (l_fin : l_finite l) := sval (leastN l_fin).
```

ここで `nat` は自然数を意味し、`Ensemble` は集合を意味する。上記の定義は次のような意味合いである。

- 1 行目: 生命関数 l に対し、集合 `ages_dead(l)` を $\{x \in \mathbb{N} \mid l_x = 0\}$ で定める。
- 2 行目: 生命関数 l に対し、命題 “ $x \in \text{ages_dead}(l)$ となる自然数 x が存在する。” を `l_finite(l)` と書く。(有限性の定義)
- 3 行目: 有限な生命関数 l に対し、集合 `ages_dead(l)` の最小値を `ult_age(l)` と書く。(最終年齢の定義)

生命関数の定義が形式化できれば、あとは通常の生命保険数学の通りに死亡数や生存確率等が定まり、それらを用いて生命年金現価や保険料等を定義することができる。

■諸公式の形式化 よく知られているように保険料や責任準備金等の間には様々な関係が成り立っている。代表的な公式としては次のようなものが形式化されている。

$$A_{x:\overline{n}} = 1 - d\ddot{a}_{x:\overline{n}}$$

```
1 Lemma ins_endow_ann_due : forall (x n : nat), x < ω ->
2   \A_{x:n} = 1 - \d * \a''_{x:n}.
```

次はいわゆる Fackler の再帰式である。

$${}_{t-1}V_{x:\overline{n}}^1 + P_{x:\overline{n}}^1 - vq_{x+t-1} = vp_{x+t-1} \cdot {}_tV_{x:\overline{n}}^1$$

```
1 Lemma res_term_rec : forall (t x n : nat), 0 < t <= n -> x+t < ω ->
2   \V_{(t-1) & x'1:n} + \P_{x'1:n} - \v * \q_{(x+t-1)} = \v * \p_{(x+t-1)} * \V_{t & x'1:n}.
```

これらは `Actuary` パッケージにおいて形式的な証明を与えられているため、論理的には一切誤りのないことが保証されている。実際の証明やその他の形式化された補題等については、GitHub にアップロードしている `Actuary` パッケージを直接ご覧いただきたい。

2.3 Actuary パッケージの応用例

実際に Actuary パッケージを用いてどのように証明を行うのか、具体的な応用例を以下に示す。これらは Actuary パッケージのファイル Examples.v から抜粋している。イメージが湧くように証明スクリプトも提示するが、詳細な説明は省略する。

■生命年金現価が予定利率に関し広義単調減少であること 予定利率 i, i' に関する生命年金現価をそれぞれ $\ddot{a}_{x:\overline{n}|}$, $\ddot{a}'_{x:\overline{n}|}$ とするとき, $i \leq i'$ ならば $\ddot{a}_{x:\overline{n}|} \geq \ddot{a}'_{x:\overline{n}|}$ が成り立つ。これは Actuary パッケージを使用して次のように証明することができる。

```

1 Lemma ann_due_decr_i : forall (i i' : R) (x n : nat), i > 0 -> i' > 0 -> x < ω ->
2   i <= i' -> \a''[i]_{x:n} >= \a''[i']_{x:n}.
3 Proof.
4   move => i i' x n Hipos Hi'pos Hx Hlei'.
5   assert (Hvpos : 0 < \v[i]) by (apply /v_pos /Hipos).
6   assert (Hv'pos : 0 < \v[i']) by (apply /v_pos /Hi'pos).
7   rewrite /life_ann_due.
8   apply Rsum_ge_compat => k /andP; case => /leP Hmk /ltP Hkn.
9   apply Rmult_ge_compat_r; [by apply (p_nonneg _ l_fin) |].
10  case: (zerop k) => [Hk0 | Hkpos].
11  - rewrite Hk0 !Rpower_0 //; lra.
12  - case: (Rle_lt_or_eq_dec i i') => // [Hlt | Heq].
13    + rewrite /Rpower.
14      apply /Rgt_ge /exp_increasing.
15      apply Rmult_lt_compat_l; [rewrite (_ : 0 = INR 0%N) //; apply lt_INR => // |].
16      apply ln_increasing => //.
17      rewrite /v_pres.
18      apply Rinv_1_lt_contravar; lra.
19      + rewrite Heq; lra.
20 Qed.

```

■将来法の責任準備金と過去法の責任準備金が一致すること Actuary パッケージでは責任準備金は将来法で定義されている。次の例は、養老保険の責任準備金について、過去法で計算しても結果が一致することを証明したものである。

```

1 Theorem res_eq_pros_retro : forall (t x n : nat), x+t < ω -> t <= n -> n > 0 ->
2   \V_{t & x:n} = \P_{x:n} * \s''_{x:t} - \A_{x'1:t} / \A_{x:t'1}.
3 Proof.
4   move => t x n Hxt Htn Hn.
5   assert (Hx : x < ω) by (apply lt_INR; move: Hxt; rewrite -plus_INR => /INR_lt; lia).
6   assert (Ha'' : \a''_{x : n} < 0) by
7     (apply /pos_neq0 / (Rge_gt_trans _ 1); [lra]; by apply (ann_due_ge1 _ i_pos _ l_fin)).
8   rewrite (_ : \s''_{x:t} = (\a''_{x:n} / \A_{x:t'1} - \a''_{(x+t):(n-t)})).
9   2:{ rewrite -(acc_due_plus_ann_due _ _ t) //;
10     rewrite /Rdiv (Rmult_assoc _ \A_{x:t'1}) Rinv_r;
11     [l apply /pos_neq0 / (ins_pure_endow_pos _ _ l_fin) => //]; lra. }

```



```

12  rewrite Rmult_plus_distr_l.
13  rewrite {1}/prem_endow_life.
14  rewrite /Rdiv -Rmult_assoc (Rmult_assoc \A_{_:}) Rinv_l // Rmult_1_r.
15  rewrite /Rminus Rplus_assoc (Rplus_comm _ (-)) -Rplus_assoc
16    -/(Rminus (_*) (_*)) -Rmult_minus_distr_r.
17  rewrite (_ : (\A_{ x : n} - \A_{ x '1: t}) */ \A_{ x : t'1} = \A_{(x+t):(n-t)}).
18  2:{ rewrite (ins_endow_pure_endow _ _ _ t) //.
19    rewrite Rplus_comm Ropp_r_simpl_l Rmult_comm -Rmult_assoc Rinv_l;
20    [| by apply /pos_neq0 /(ins_pure_endow_pos _ _ l_fin)]; lra. }
21  rewrite /res_endow_life; lra.
22  Qed.

```

2.4 今後の課題

Actuary パッケージはまだ開発初期段階であり、形式化した範囲も狭い。今後の発展の方向性としては以下のようなものが考えられる。

■**死力の形式化** 現時点では生命関数に微分可能性を仮定していないため、死力の概念が導入できていない。今後は死力を形式化した上で、連続払の生命年金現価等も扱えるようにしたい。微分や積分については既に Coq で形式化されたパッケージがあるので、そちらを活用することになるだろう。

■**連合生命** 連合生命については現時点で形式化されておらず、今後の課題である。連合生命では記号が更に複雑になるため、Coq に実装する際に記法をどう定めるかが問題になると思われる。特に生命表の個数を可変にする場合はなおさら困難が伴うであろう。

■**推論の自動化** 定理証明支援系による形式化は証明に一切の飛躍が許されないので多くの時間を要する。Actuary パッケージでもそれは同じで、人間には明らかと思えるような等式でも Coq で証明するのは容易でないことがある。生命保険数学でよく用いられる式変形等については自動的に推論してくれるようなタクティクが開発できれば、アクチュアリー数学の形式化が促進されるであろう。

3 今後の展望

最後に、Coq や Actuary パッケージの将来的な応用可能性について考察する。

3.1 産業界への応用

今まで定理証明支援系の学術的な側面を中心に述べてきたが、産業界への応用も十分に考えられる。

■**アクチュアリー業務への応用** Actuary パッケージを更に発展させることができれば、アクチュアリー業務への応用も視野に入ってくる。例えば、算出方法書に記載される数式の検証や、保険数理に関するプログラムの検証等を Coq で行うことが可能になるかもしれない。現状は Coq での検証に論理学等の基礎知識が要求されるが、ユーザインターフェースを洗練させたり、Coq の自動推論機能を強化させたりすることによって、必要な予備知識を少なくしていくことが望ましいだろう。

■Python への応用 定理証明支援系がソフトウェアの品質保証に活用できることは前述した通りである。一方で、形式的証明はシステムの開発工数を増加させる懸念もあることから、現実にはコストパフォーマンスを加味した運用が求められるようである。Python は機械学習でも使用される有名な言語であるが、2016 年に富士通研究所から Python を使用した開発に Coq を応用する興味深い研究が報告された [27]。この報告では、業務で実際に 9 ヶ月以上使用されてきた Python のある関数について、Coq でバグを発見できたことが述べられている。データサイエンスは近年発展が著しく、またアクチュアリー業務への応用も期待される分野であるが、定理証明支援系はこの方面でも力を発揮するかもしれない。

3.2 数学の形式化の未来

定理証明支援や自動定理証明の技術が進歩し、現代数学の形式化が更に進んだ未来ではどのようなことが起こるだろうか。将来、人工知能が人間の数学者を凌駕する日が来るだろうか。これは深い問いであり、それをテーマとして一冊の本ができるほどである [25]。

コンピュータによる証明が今後どこまで進歩するかはわからないが、少なくとも現在、我々がその目覚ましい発展の渦中にいることは間違いない。筆者も数学を専攻した者の端くれとして、アクチュアリーの立場からこの研究に少しでも貢献していきたい。

付録

定理証明支援系はその有用性にもかかわらず日本における知名度が低いように感じている。本論文を執筆した動機の一つは、数学の予備知識を持つアクチュアリーの方々へ Coq の存在を知ってもらうことであった。

Coq の定番の入門書としては Yves Bertot, Pierre Castéran による “Interactive Theorem Proving and Program Development” [3] がある。計算機科学者向けのテキストとしては Benjamin C. Pierce 等による “Software Foundations” というシリーズがあり、以下の Web サイトから閲覧可能である。

<https://softwarefoundations.cis.upenn.edu/>

このシリーズの一部は日本語にも翻訳されている。

<https://www.chiguri.info/sfja/>

また、日本語で書かれた初学者向けの記事として池淵未来のチュートリアルがある。

<https://www.iiij-ii.co.jp/activities/programming-coq/>

Actuary パッケージで利用している SSReflect/MathComp については Yves Bertot 等によるオンラインの入門書 “Mathematical Components” [4] が無料で閲覧可能である。日本語の入門書としては、萩原学, Reynald Affeldt による “Coq/SSReflect/MathComp による定理証明” [12] が 2018 年に出版された。

Coq のコミュニティは全世界に広がっており、Coq のメーリングリスト coq-club^{*21}や Zulip のチャットルーム^{*22}では毎日世界中の Coq ユーザの間で活発なやり取りがなされている。本論文が Coq コミュニティに

^{*21} coq-club@inria.fr

^{*22} <https://coq.zulipchat.com>

新たな参加者が加わるきっかけとなれば望外の喜びである.

謝辞

アクチュアリー数学の形式化にあたり, Coq を始めとする定理証明支援系の研究状況につき, 産業技術総合研究所の Reynald Affeldt 氏より貴重な助言をいただきました. 心より感謝申し上げます.

参考文献

- [1] Kenneth Appel and Wolfgang Haken. Every planar map is four colorable. *Bulletin of the American Mathematical Society*, Vol. 82, No. 5, pp. 711–712, 1976.
- [2] Jeremy Avigad, Johannes Hölzl, and Luke Serafin. A formally verified proof of the central limit theorem. *Journal of Automated Reasoning*, Vol. 59, pp. 389–423, 2017.
- [3] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development: Coq’Art: The Calculus of Inductive Constructions*. Springer-Verlag Berlin Heidelberg New York, 2004.
- [4] Yves Bertot, Georges Gonthier, Assia Mahboubi, and Enrico Tassi. Mathematical components. <https://zenodo.org/record/4457887#.YCjx12NUtpQ>, 2021.
- [5] Kevin Buzzard. Proving theorems with computers. *Notices of the American Mathematical Society*, Vol. 67, No. 11, pp. 1791–1799, 2020.
- [6] Thierry Coquand and Gérard Huet. The calculus of constructions. Technical Report RR-0530, INRIA, 1986.
- [7] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, pp. 378–388. Springer International Publishing, 2015.
- [8] 二見隆. 生命保険数学 (’92 改訂版) 上巻. 生命保険文化研究所, 1992.
- [9] 二見隆. 生命保険数学 (’92 改訂版) 下巻. 生命保険文化研究所, 1992.
- [10] Georges Gonthier. A computer-checked proof of the four colour theorem. <http://www2.tcs.tu-berlin.de/~abel/lehre/WS07-08/CAFR/4colproof.pdf>, 2005.
- [11] Sébastien Gouëzel and Vladimir Shchur. A corrected quantitative version of the Morse lemma. *Journal of Functional Analysis*, Vol. 277, No. 4, pp. 1258–1268, 2019.
- [12] 萩原学, アフェルトレナルド. Coq/SSReflect/MathComp による定理証明: フリーソフトではじめる数学の形式化. 森北出版, 2018.
- [13] Thomas C. Hales. Formal proof. *Notices of the American Mathematical Society*, Vol. 55, No. 11, pp. 1370–1380, 2008.
- [14] 情報処理推進機構. セキュリティ機能と保証レベル. <https://www.ipa.go.jp/security/jisec/forusers/abouteal.html>, 2020.
- [15] Chester Wallace Jordan, Jr. *Society of Actuaries’ Textbook on Life Contingencies*. The Society of Actuaries, second edition, 1967.
- [16] Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. *Journal of Automated Reasoning*, Vol. 53, pp. 173–213, 2014.

- [17] Vladislav Kargin. Life contingency models I. https://www2.math.binghamton.edu/lib/exe/fetch.php/people/kargin/publications/actuarial_1_ln_master.pdf, 2017.
- [18] Gerwin Klein, et al. seL4: Formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, pp. 207–220, 2009.
- [19] Johnny Li and Andrew Ng. *SOA Exam MLC Study Manual*. ACTEX Learning, fall 2017 edition, 2017. Also available as <https://www.actexamdriver.com/trials/Actex%20MLC%20Fall%202017%20Print%20FT%20sample.pdf>.
- [20] The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, No. 15 in CPP 2020, pp. 367–381. Association for Computing Machinery, 2020.
- [21] William Mccune. Solution of the Robbins problem. *Journal of Automated Reasoning*, Vol. 19, pp. 263–276, 1997.
- [22] Lawrence C. Paulson. Natural deduction as higher-order resolution. *Journal of Logic Programming*, Vol. 3, pp. 237–258, 1986.
- [23] Eric V. Slud. Actuarial mathematics and life-table statistics. <https://www.math.umd.edu/~slud/s470/BookChaps/01Book.pdf>, 2001.
- [24] The Coq Development Team. The coq reference manual: Release 8.13.1. <https://coq.inria.fr/distrib/current/refman/>, February 2021.
- [25] 照井一成. コンピュータは数学者になれるのか?: 数学基礎論から証明とプログラムの理論へ. 青土社, 2015.
- [26] Freek Wiedijk. Formal proof: Getting started. *Notices of the American Mathematical Society*, Vol. 55, No. 11, pp. 1408–1414, 2008.
- [27] 山本晃治, 宗像一樹. 証明駆動開発の現実的な開発プロジェクトへの適用に向けて. ソフトウェアエンジニアリングシンポジウム 2016 論文集, Vol. 2016, pp. 104–111, 2016.