Autoformalization に向けた自然言語証明構造の形式化

服部 清志　松崎 拓也　藤原 誠
東京理科大学　理学部第一部　応用数学科
1420080@ed.tus.ac.jp matuzaki@rs.tus.ac.jp makotofujiwara@rs.tus.ac.jp

概要

数学証明の自動形式化は数学論文の内容検証やシステムの安全性保証のために有用な技術である。しかし、現在主流である大規模言語モデル (LLM) を用いた手法の精度は高校・学部レベルの数学を内容とする形式化のベンチマークに対しても 50%程度に留まっている。本研究では証明の議論構造の解析を中心とした、自動形式化を実現するための新たなプロセスを示し、各段階の実現のための提案を行う。また、大学初年級レベルの解析学の教科書に掲載されている証明の形式化を行い、矛盾の導出に関わる表現など、自動形式化における課題を分析する。

1　はじめに

一方、大規模言語モデル (LLM) の発展により、定理証明の自動形式化 (Autoformalization) の研究が活発化している [2, 4, 5, 7]。自動形式化とは、自然言語で書かれた数学証明を形式言語による証明に自動翻訳することであり、定理証明の自動検証や複雑なソフトウェアの安全性検証等で有用な技術である。現在は、自然言語による定理証明を LLM で証明支援系の文法に従って形式化し、証明の細部を自動演繹によって埋める、という手法が主に試みられている。しかし、数学オリンピックや高校・学部レベルの数学に基づく形式化のベンチマークである MiniF2F [9] におけるその正答率は 50%程度に留まっている [6, 8]。

本研究は、自動形式化を実現するための手法として、自然言語証明の議論構造の解析を経由して Lean の形式証明に変換するという方法を提案する。また、その第一ステップである証明の議論構造の解析における課題点を、日本語で書かれた大学初年級レベルの定理証明を Lean で形式化した結果の分析に基づき考察する。さらに、その課題の一つである変数スコープの解決アルゴリズムを提案する。

本論文の以下の構成は次の通り。2 節では関連研究を紹介する。3 節では本研究で使用した証明支援系 Lean の説明を行う。4 節では本研究で提案する自動形式化の実現手法と議論構造の表現及び変数スコープ解決アルゴリズムの紹介を行う。5 節では定理証明の形式化結果の分析及び考察について述べる。6 節では残る課題について議論する。

2　先行研究

自動形式化に関する近年の研究のうち、本研究と特に関連するものをいくつか紹介する。Jiang ら [2] は、自然言語証明を証明内の一部の推論が欠けた形式証明に変換した後に、証明器を使い形式証明を完成させるという手法を提案した。我々の提案方法もこの流れを踏襲するが、本研究では証明で述べられている演繹の各操作を中間構造である議論構造表現に変換するステップを設ける。中間構造を適切に設計することで、Jiang らのように自然言語証明から直接形式証明に近い形に変換するよりもタスクが限定され、現状数が少ない自然言語証明と形式証明が対となったデータからより効率的に学習できると考えられる。

Zheng ら [8] らは LLM によって形式証明を生成し、その際に発生するエラーを自動証明器と証明支援系との相互作用によって修正することによって正しい形式証明を作成する、という手法を提案した。我々が提案する変数スコープエラー解決アルゴリズムも生成された形式証明に発生するエラーを修正する効果を持つ、という点では一致しているが、Zheng らの手法が LLM に起因するエラーの修正を目的としていることに対し、本研究は自然言語証明における慣習的な表現や曖昧性が原因で発生するエラーの解決を目的としている、という点で異なる。

3　証明支援系 Lean

証明支援系とは、数学の定義 / 定理 / 証明を記述できる形式言語を提供し、かつユーザーとの対話（形式言語でのやりとり）を通じて数学証明の自動

検証を行うシステムを指す。

Lean [1] はこの証明支援系に属する、依存型付きラムダ計算に基づくプログラミング言語であり、「タクティク」と呼ばれるコマンドを用いた対話的証明支援機能を持つ。また、ユーザーコミュニティによって作成されているライブラリである“Mathlib”を活用することで、既に形式化された学部レベルの数学定理を証明せずに使用できる。本研究では2021年にリリースされたLean4およびMathlib4を用いて形式化の検証および分析を行った。

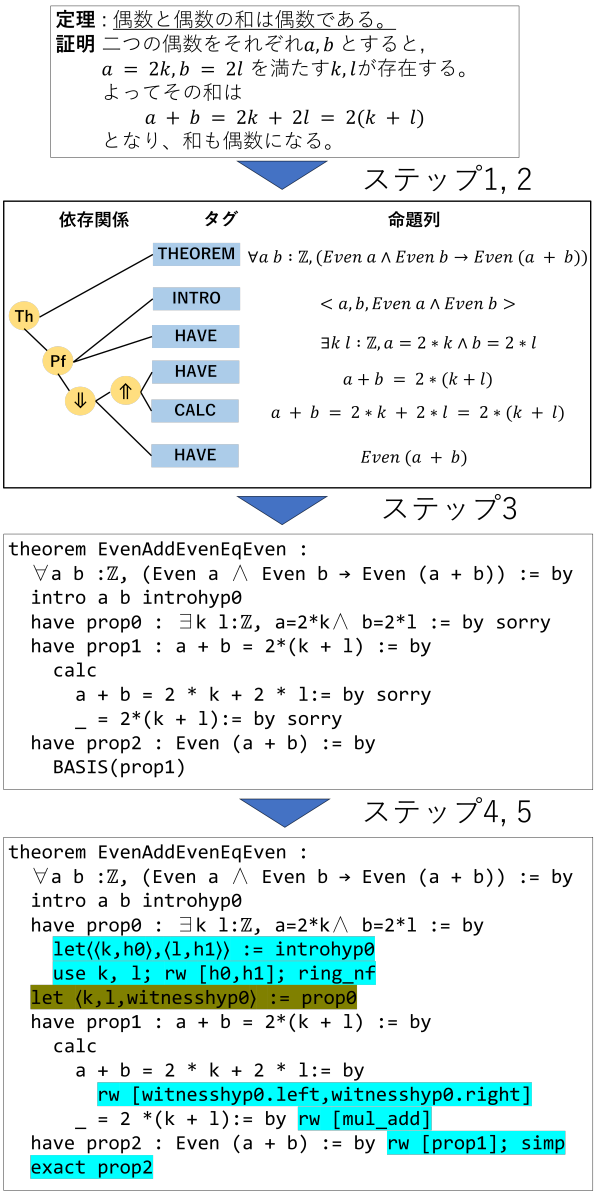

図 1　自動形式化の実行手順

4　提案手法

4.1　自動形式化の実現手順

本研究では以下の5つのステップで実現される自動形式化を考える。ここで、入力は自然言語証明、出力はLeanの形式言語で記述された証明である。

1. 自然言語で書かれた証明から命題の列を抽出し、各々を形式表現に変換する。
2. 各命題を証明中での役割でタグ付けし、命題間の依存構造を解析する（**議論構造解析**）。
3. 議論構造解析の結果（**議論構造**）をLean4の文法で書かれた**証明構造**に変換する。
4. 証明構造上で同一の対象を指示する変数を確定させるとともにLeanの文法上未定義となる変数を導入コマンドの追加等により解決する。
5. 各サブゴールに対して自動演繹を適用し、証明構造内の空白部分を埋める。

図1は上記実行手順を、「偶数と偶数の和は偶数である」の証明に適用した例である。ステップ1で証明から6つの命題を抽出し形式化したのち、ステップ2で各命題に対しHAVE（サブゴールの宣言）、INTRO（変数や条件の導入）など証明内の演繹の各操作を形式化したタグを付与し、命題間の依存関係が表現された議論構造に変換している。ステップ3では、議論構造を証明内の一部推論が欠けた形式証明に変換する。なお、欠落した推論部分には特殊なコマンド“sorry”を置く。ステップ4では サブゴール $a + b = 2 * (k + l)$ における未定義変数 k, l を let タクティクを挿入することで解決（図中黄色マーカー部分）し、ステップ5で各サブゴールを自動演繹によって証明することで形式証明を完成させている（図中水色マーカー部分）。

現時点では上記ステップ3, 4のみが具体的なアルゴリズムとして実現できている。本論文では、主としてステップ2, 3, 4について議論する。

4.2　議論構造表現の設計

本節以降では、「自然言語証明における議論構造」を「議論構造」と略記する。議論構造は定理と証明の対を表現するThを根、⇑, ⇓, Pfのいずれかを中間ノード、タグ付けされた命題を葉とする木構造で表される。命題に対して付与される全てのタグは変換先のLean4のタクティクに対応する形で設計されて

いる。背理法の導入を表す BY_CONTRA タグを除き、各タグは一つ以上の引数を取る。タグの一覧は Appendix A に示す。

中間ノード Pf はその子ノード達が Pf ノードの親ノードの論拠になっていることを示す。

⇓ ノードは先行する命題 Φ が後続の命題 Ψ の論拠になっていることを表し、Lean で書かれた証明構造の “have . . . : Ψ := by BASIS(Φ)” に変換される (図 1 の上から三段目の囲み部分、最下行)。BASIS(. . .) は上記のステップ 5 で実行される自動演繹におけるヒントとなる。

⇑ ノードは後続する命題 Ψ が先行する命題 Φ の論拠になっていることを表し、Lean で書かれた証明構造の “have . . . : Φ := by Ψ” に変換される。(図 1 の上から三段目の囲み部分、5-8 行目)。

4.3 変数スコープエラー解決アルゴリズム

自然言語証明中に登場する各変数がもつスコープの広さを自然言語の表層的、局所的な情報から正確に把握することは難しく、これは自動形式化における主要な課題となっている。

形式証明上で各変数が正しいスコープを持つことを保証する仕組みとして、以下では変数スコープエラー解決アルゴリズムを提案する。例えば、次の文では波線を付した m が自由変数として出現している。

$$\text{「}n \in \mathbb{N} \text{ があって}, \underset{\sim}{m} \geq n \text{ ならば } |x_{\underset{\sim}{m}} - a| < \epsilon/2\text{」}$$

このとき、この m の解釈として以下の場合が考えられる。

1. 先行する文脈で特定の条件を満たす m の存在が証明されており、その witness と同一のものを指している
2. この命題が別の命題を示すための論拠であり、示すべき命題が m についての全称命題である
3. 上記のいずれでもない

上記の命題をそのまま形式化した、

$$\exists n : \mathbb{N}, (m \geq n \Rightarrow |x_m - a| < \epsilon/2)$$

を Lean 上で サブゴールとして設定すると、変数 m は 「未知の識別子」 エラーとなる。このような場合、提案アルゴリズムは示すべき命題や既に証明済みの命題から該当の変数を導入できるか探索し、導入コマンドの追加もしくは変数の全称量化によって対応する。上記 1. の場合、 “let” タクティクを使用した witness の導入コマンドの追加、2. の場合は “intro” タクティクを使用した導入コマンドの追加、3. の場合は全称量化子の追加が対応する操作となる。図 2 に疑似アルゴリズムを示す。

Require: 命題 Φ の中で、Lean の形式証明上で未定義となる変数を X とする
if 命題 Φ が別の命題 Ψ を示すためのサブゴールであり、Ψ が $\forall X, \cdots$ の形である **then**
    サブゴール Φ の前に X を導入する命令 (intro X) を追加
    return
props ← X が出現する証明済みの命題
if props の個数が 0 **then**
    $\Phi \leftarrow \forall X, \Phi$
    return
for Ψ in props **do**
    if Ψ が $\exists X, \cdots$ という形である **then**
        Φ の直前に Ψ の witness として X を導入 (let $\langle X, \ldots \rangle$:= Ψ)
        return
$\Phi \leftarrow \forall X, \Phi$

図 2 変数スコープエラー解決アルゴリズム

4.4 サブゴールに対する自動演繹

本研究では、生成された各サブゴールの持つ空白部分、すなわち未証明の部分を自動演繹により証明し、Lean の形式言語で記述された完全な証明を出力することを目標とする。また、本研究では自動演繹を適用する各サブゴールに対し、自然言語の証明から読み取ることのできる論拠をヒントとして付与する。これは自動演繹を行う上での一助となることを想定している。なお、この情報が自動演繹の成功率をどの程度向上させるのかは未知であるため、今後実際に自動演繹を実装してどの程度実効性があるか検証する必要がある。

5 検証および分析

本節では自然言語証明から議論構造表現への変換部分の検証方法及び分析結果について述べる。

5.1 検証方法

宮島静雄著「微分積分学 I」[10] の第 1 章（実数の連続性と数列の収束）の 1 節から 2 節および 2 章（一変数連続関数）の 1 節から 2 節の証明 26 個のうち 24 個の証明に対し、命題の抽出、タグの付与及び形式表現への変換、依存構造の解析 (図 1 上から

二つ目の囲みのデータ形式の作成) を手動で行った。その後、プログラムを用いて解析結果を証明構造に自動的に変換し、手動で変数スコープ解決アルゴリズムの適用を行った。また、それら 24 個の証明について Lean における形式化を別途行い、上記の手順で自動生成した証明構造との比較分析を行った。なお、検証を行わなかった 2 つの証明のうち 1 つは、証明内で他の命題の証明内に登場した変数の条件を参照しており、形式証明として表現できないため省いた。また、もう 1 つの証明についてはスキル不足により形式表現が作成できなかったため除いた。

5.2 分析結果

検証を行った 24 個の証明全てに対し、自然言語証明から議論構造解析まで (§4.1 のステップ 1 および 2) が理想的に行われたとすれば、Lean で書かれた証明構造を正しく出力できることが確認された。さらに、それらに対して図 2 のアルゴリズムを適用することで、変数のスコープ制約違反が解消されることが確認された。なお、変数スコープの制約違反を含んでいた証明構造は 24 個の証明のうち 22 個であり、平均して 3.1 個の制約違反を含んでいた。このことから、スコープ解決アルゴリズムの意義は大きいといえる。

6 課題点

本節では検証を行う中で明らかになった、自動形式化の実行手順（§4.1）における課題点を述べる。

6.1 議論構造解析

議論構造解析には、命題の抽出、形式表現への変換、命題間の関係の分析の三要素が必要である。

命題の抽出を行うためには命題を構成する単語列の範囲を把握し抽出する必要があり、形式的にはスパン抽出タスクに近い。しかし、命題を宣言している文内でその命題の論拠についても言及されている場合 (図 1 内に出現する $a+b=2*k+2*l=2*(k+l)$ がその一例) や、命題の宣言と仮定の導入を同時に行っている場合など、スパンと命題が完全には一致しないケースもある。

自然言語で書かれた命題を形式表現へと変換する課題は、いわゆる Semantic Parsing の一種だが、数学テキストに対する研究は高校数学を対象としたもの [3] などわずかしか存在しない。

議論構造の解析は談話構造解析（文章内の文間関係の解析）の一種と考えられるが、数学証明に対しての研究は我々の知る限り存在しない。例えば、数学証明において頻出する論理展開の 1 つに、何かしらの仮定を置いて推論を行い矛盾を導く、というものがある。この仮定の置き方には、Lean における “by_contra” タクティクに対応する背理法による証明と、Lean においてはサブゴールの宣言 “have $\neg P$” とそれに続く仮定の導入 “intro P” に対応する否定命題の証明の 2 通りがあり、第 5 節の分析で扱った証明においては、否定命題を仮定して矛盾を示す場合はすべて背理法、そうでない場合は否定命題の証明に対応していた。しかし、自然言語で書かれた数学証明において上記の 2 種類の推論の違いが意識されていないことは多く、どちらの推論が行われているかを判定するには前後の文脈を正確に読み取る仕組みが必要になると思われる。

6.2 タグの設計

現在、議論構造を表現するために使用しているタグは本研究で検証を行った証明において出現した操作を表現するために必要な最小限のものである。そのため、数学的帰納法や「〜と同様に」「〜であるようにとればよい」といった、数学証明に頻出する表現に対応するタグは定義していない。今後検証例を増やし必要なタグを追加していくとともに、タグの増加がある一定の数で収束することを併せて示す必要がある。

7 おわりに

本研究では、数学証明の自動形式化を実現する手法として、自然言語証明の議論構造の解析を経由して形式証明へ変換するという方法を提案した。それに併せ、議論構造の表現の設計及び変数スコープエラー解決アルゴリズムの提案を行った。また、自然言語で書かれた数学証明から議論構造表現への変換について検証し、理想的な変換が行われれば正しい出力が得られることを確かめ、さらに幾つかの課題点を示した。今後は議論構造の解析や命題の形式化表現への変換モデルの実装、自動演繹の実装に取り組んでいきたい。

参考文献

[1] Lean FRO. Lean programming language and theorem prover, 2013. https://lean-lang.org/.

[2] Albert Q Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. arXiv preprint arXiv:2210.12283, 2022.

[3] Takuya Matsuzaki, Hidenao Iwane, Hirokazu Anai, and Noriko Arai. The complexity of math problems – linguistic, or computational? In Ruslan Mitkov and Jong C. Park, editors, Proceedings of the Sixth International Joint Conference on Natural Language Processing, pp. 73–81, Nagoya, Japan, October 2013. Asian Federation of Natural Language Processing.

[4] Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning. arXiv preprint arXiv:2202.01344, 2022.

[5] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. arXiv preprint arXiv:2009.03393, 2020.

[6] Huajian Xin, Haiming Wang, Chuanyang Zheng, Lin Li, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, et al. Lego-prover: Neural theorem proving with growing libraries. arXiv preprint arXiv:2310.00656, 2023.

[7] Xueliang Zhao, Wenda Li, and Lingpeng Kong. Decomposing the enigma: Subgoal-based demonstration learning for formal theorem proving. arXiv preprint arXiv:2305.16366, 2023.

[8] Chuanyang Zheng, Haiming Wang, Enze Xie, Zhengying Liu, Jiankai Sun, Huajian Xin, Jianhao Shen, Zhenguo Li, and Yu Li. Lyra: Orchestrating dual correction in automated theorem proving. arXiv preprint arXiv:2309.15806, 2023.

[9] Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. CoRR, Vol. abs/2109.00110, , 2021.

[10] 宮島静雄. 微分積分学 I - 1 変数の微分積分 -. 共立出版, 2010.

A　タグセット

議論構造表現 (§4.2) で使用されているタグセットと役割を以下にまとめる。

- **THEOREM** ··· 証明する命題の宣言および証明の開始を表す。議論構造の木構造における根ノードに対応する。
- **PROOF** ··· 親ノードで宣言された命題の証明に対応する命題列の開始を表す。
- **INTRO** ···　仮定の導入を表す。
- **LET** ··· 新たな変数に対して特定の数式もしくは述語を割り当てる（命名の宣言）。
- **HAVE** ··· サブゴールとなる命題の宣言を行う。
- **WITNESS** ··· 既に証明された存在命題から変数を witness として仮定に導入する。
- **BY_CONTRA** ··· 命題を偽であると仮定し、矛盾を導くことによって命題が真であることを証明すること (背理法の適用) を宣言する。
- **SUFFICES** ··· 特定の命題を証明したものとして仮定に導入する。
- **CONTRADICTION** ··· 現時点で証明された命題の中に相容れないものがあることを宣言する。矛盾を示す議論構造の終点を表す。
- **CALC** ··· 複数の比較演算子 $(=, <, >, \leq, \geq)$ で連結された数式を逐次的に比較することによって証明することを示す。例えば、$A = B < C$ であれば、$A = B$ を証明したのちに $B < C$ を証明することを示す。
- **BASIS** ···　論拠を示す。主に既に証明されている命題や定理の適用を明示するために用いる。
- **EMPTY** ··· 証明する命題に明示された論拠がないことを表す。

B　Lean のタクティク

図 1 中で使用されている Lean のタクティクのうち、タグで表現されないタクティク (use, rw, ring_nf, simp, exact) について説明する。

- **use** ··· ゴールもしくはサブゴールの先頭にある存在量化子に対して witness を与える。
- **rw** ··· [ ] で囲まれた命題で現在のゴールもしくは既に証明した命題の該当部分を書き換える。
- **ring_nf** ··· 可換環（加法と可換な乗法が定義された集合）の式を正規化 (式内の足し算掛け算を可能な限り実行し、式を簡単に) する。
- **simp** ··· 参照可能な証明済みの定理を使って現在のゴールをできる限り単純なものに書き換える。
- **exact** ··· 引数に与えられた仮定 (図 1 中では prop2) がゴールと同じものであれば証明を完了する。

C　タクティク適用による Lean 上での状態変化の例

以下の図 3 は 図 1 最下段の囲み部分 6 行目の三つの操作による状態の変化をまとめたものである。

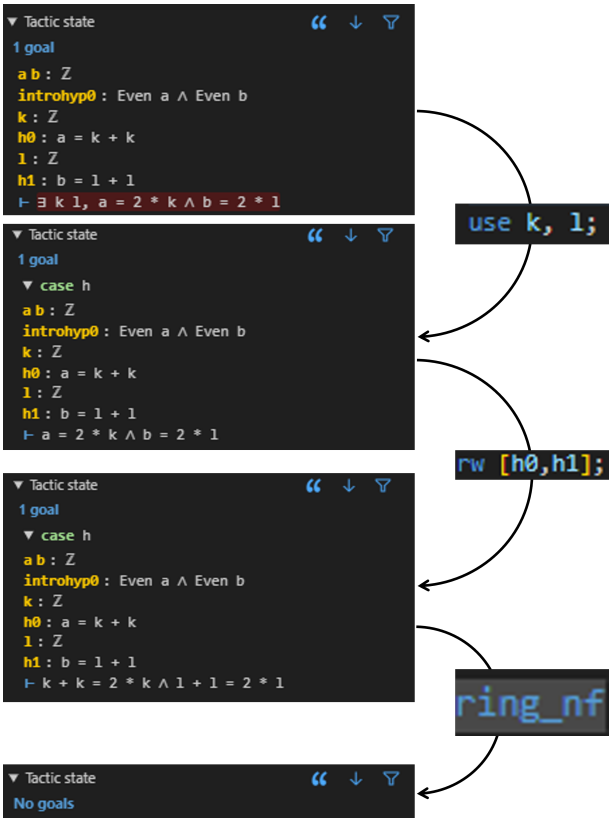

図 3　最初の操作である **use k l;** によってゴールの $\exists k, l$ が仮定にある k, l を witness として書き換えられる。
次に **rw [h0, h1]** によって、ゴールの a, b がそれぞれ $k + k, l + l$ で書き換えられる。
最後に、**ring_nf** をによって、ゴールにある足し算及び掛け算が可能な限り実行され、式が簡略化される。その結果、論理積で結ばれた二つの式がどちらも恒等式となり、自動的に証明済みとなる。(“no goals” という表記は証明が完了したことを表す。)