指定した形の中間証明状態へ至る 形式証明の自動生成を目指して ～Transformerを用いた深層学習的アプローチ～

鵜狩 慧久[1,a)]　江本 健斗[1]

概要：プログラムへのバグの埋め込みがないことを Coq などの定理証明支援系を用いて形式的に証明し保証する手法がある．しかし，この手法には労力的な課題があり，自動化しようにも単純な探索では計算量的に難しい．深層学習的な手法を用いた証明自動生成手法の提案もあるが，その証明生成の成功率は 30% 程度であり，実用上の十分な利便性を提供できていない．この原因としては，モデルの目標を証明の完成に限定している点が考えられる．本研究は，中間証明状態を目標としてユーザが提示できるような機構をモデル構造やユーザインターフェースに追加し，モデルの目標達成率や利便性の向上を目指す．

キーワード：定理証明支援系，Coq，深層学習

Towards Automatic Generation of a Formal Proof to an Intermediate Goal State of a Specified Form –A Deep-learning Approach with Transformers–

Ukari Eku[1,a)]　Emoto Kento[1]

Abstract: Nowadays, formal methods using proof assistants such as Coq have been used to guarantee that programs are bug-free. However, this method suffers from the heavy burden of building formal proofs, and automatic proofs through simple search methods is difficult due to its computational comlexity. A deep-learning approach for automatic proof generation has been proposed, but its achievement rate is about 30% and this is not satisfactory. We think that the reason is that the model's goal is to finish the entire proof. This research proposes an approach based on an interface and a model for users to give an intermediate proof state as the model's goal, aiming at improving the usability and goal achievement of the model

Keywords: Proof Assistant, Coq, Deep-learning

1. はじめに

プログラムに潜むバグは，大きな社会的損失の原因となる．例えば，乗り物や産業ロボットの制御プログラムのバグは，命に関わる重大な事故の原因となる．また，金融システムやセキュリティソフトのバグは，それを用いる個人や団体に経済的な損失や法的責任問題をもたらす．

プログラムへのバグ埋め込みを防ぐ方法のひとつとして，Coq [1] などの定理証明支援系を用いる手法がある．プログラムが満たすべき性質（入力に対して想定外の出力をしないこと等）を論理式で表し，その論理式を定理証明支援系の支援によって見落としなく形式的に証明することで，プログラムの振る舞いを保証するという手法である．

しかし，定理証明支援系を用いる手法には一般的に労力的な課題がある．例えば，純粋な数学の定理である四色定

[1] 九州工業大学
Kyushu Institute of Technology, Iizuka, Fukuoka 820–8502, Japan
[a)] ukari@pl.ai.kyutech.ac.jp

理の Coq での証明には 4 年がかかった [3]．また，実用的なソフトウェアにバグがないことの証明としては，C 言語コンパイラの証明に 5 年がかかった [4]．産業分野での形式的手法の普及にはこの証明の労力削減が必要である．

定理証明支援系での証明の労力を削減する方法として，ソフトウェアによる証明の自動化が考えられる．単純な探索では計算量的な困難があるため，深層学習を用いた効率的な手法として証明自動化モデル ASTactic [10] が提案された．しかし，残念ながら ASTactic の証明生成の成功率は 30%程度であり，利便性が良いとはいえない．この原因として，ASTactic がモデルの目標を証明の完成に限定している点が考えられる．

本研究は，証明全体の完了ではなく，証明の中間状態（証明の途中の形）を目標として証明の自動生成を行うアプローチを提案する．すなわち，証明の中間状態をユーザが提示できるような機構をモデルやユーザインターフェースに追加し，モデルにはその中間状態までの証明の生成を行わせることで，モデルに与えるタスク難易度の調整を行えるようにして証明生成の成功率と利便性の向上を図る．

以降の本論文の構成は以下のとおりである．まず，第 2 節にて既存の事柄についての導入を行う．次に，第 3 節にて提案手法を導入する．そして，第 4 節にて提案手法を評価し，最後に，第 5 節にて本論文をまとめる．

2. 準備

本節では，本論文で用いる既存結果について導入する．

2.1 定理証明支援系 Coq

定理証明支援系 Coq [1] は，形式的証明の正しさを保証するシステムである．ユーザは，関数型言語 Gallina を用いて，プログラム本体やその性質に関する定理を記述する．そして，Coq による確認のもと，タクティックというコマンドを用いて定理の証明を対話的に進める．証明が完了すれば，その証明が正しいことが保証される．

簡単な定理の証明例として，加算を実装したプログラム `plus` が，$0 + n = n$ の等式を満たすことの証明を示す：

```
1 Theorem plus_O_n : ∀n : nat, plus O n = n.
2 Proof.
3   intros.
4   simpl.
5   reflexivity.
6 Qed.
```

Coq での定理や補題は，**Theorem** や **Lemma** コマンドの後に名前を書き（上の例では `plus_O_n`），: の後ろに命題を書く．そして，続く **Proof** コマンドから，Coq との対話を通して命題の証明を進め，証明が完了したら **Qed** コマンドで証明を閉じる．この際，Coq は証明すべき命題をゴールとして提示してくるので，ユーザは，そのゴールをより自明な命題に還元するようにタクティックを入力していく．ここでは詳細を省くが，上記の例では，`intros` と `simpl` のタクティックでゴールを自明な等式に還元し，最後に等号の反射性により証明を完了するために `reflexivity` タクティックを用いている．

2.2 深層学習と Transformer モデル

深層学習は，一定以上の層数・パラメータ数を持ったニューラルネットワーク [6] を通じて，データから高度な特徴を学習する機械学習の一手法である．深層学習では，生の入力データから，より抽象的なレベルの表現を段階的に抽出する．各層は，前の層からの出力を入力として受け取り，非線形変換を適用して次の層への入力を生成する．深層学習モデルの主要な要素には，多数の隠れ層と非線形活性化関数があり，これらによりモデルは複雑なデータ構造を捉えることができる．その具体的な応用は，画像認識・音声認識・自然言語処理など, 多岐にわたる [5], [8].

Transformer モデル [9] は，アテンション機構を主要な構成要素としたニューラルネットワークアーキテクチャである．位置符号化とアテンション機構により，Transformer は入力データ内の任意の位置間の関係を効率的に学習し，様々な距離の依存関係を捉えることができる．また，モデルの構造上，並列計算との相性がよく，大規模な学習に向いている．これらの特徴のため，既存の RNN や LSTM を置き換えるものとして様々な目的に応用されている．

2.3 ASTactic とその問題点

Coq での証明作業の深層学習に基づく自動化への試みとして，自動定理証明のための推論モデルである ASTactic が提案された [10]．ASTactic は，現在の証明状況（証明したいゴールや仮定）を AST（抽象構文木）の形で受け取り，そこから証明を 1 ステップ進めるためのタクティックを AST の形で出力する．AST で入出力を扱うことで，緻密な環境認識と柔軟な証明生成を実現している．この 1 ステップのタクティックの生成は複数個行われ，証明が完了するまで DFS でこれが繰り返される．

ASTactic の問題点は，それが証明の完了を目標としている点にある．ASTactic は，最大で 30%の精度で証明を完成させることができているものの，証明が完成できなかった場合には，ユーザは手動で証明を進める他に選択肢がない．そして，このユーザの手による証明の進行は，ASTactic が残りの証明を終えられる地点に至るまで延々と繰り返されることになる．つまり，ゴールの見える地点までの証明をすべてユーザが与えないと ASTactic はゴールに辿り着いてくれず，これでは利便性が高いとは言い難い．ゴールの見えない地点においても「この辺りに進め」という大雑把な道筋をユーザが与えられることが望ましい．

3. 提案手法

本研究は，前節で述べた既存手法の問題点に対し，ユーザが証明の大雑把な道筋を与えられるような機構を導入して問題の解決を図る．具体的には，目標とする証明の中間状態（ゴールの形）をユーザが提示できるような機構をモデルやユーザインターフェースに追加し，モデルにはその中間状態までの証明の生成を行わせる．

3.1 提案手法による証明進行の概要

図 1 に，既存手法と提案手法の証明進行の比較を示す．

既存手法では，証明の最初の段階で自動証明モデルを呼び出した場合（**Proof** 直後の `ASTactic`），モデルは証明の完成を目標とし，橙色で示す 4 つのタクティックすべての生成を期待される．必要な証明ステップが長くなると，モデルに要求されるタスクの難易度が高くなり，目標達成が難しくなる．

一方，提案手法では，ユーザは次に辿り着いて欲しい「大雑把なゴールの形（証明の中間状態）」を “_ + _ = _ + _” というクエリによって示す（::: がクエリの開始を意味し，‘_’ はワイルドカードを意味する）．提案モデルは，与えられたゴールの形に至ることを（証明全体の完了に比べて短期的な）目標として，橙色で示す 2 つのタクティックの生成を期待される．モデルに要求されるタスクの難易度は，ユーザがある程度コントロール可能であり，仮にモデルが証明に失敗しても，ユーザにはクエリの与え方をより易しいものに変えるなどの選択肢がある．ユーザは，この「大雑把なゴールの形」を繰り返し与えることで，モデルを誘導しつつ証明を完了させる．

なお，人手による定理の証明の構築時には，ノート上でも Coq 上でも，「次の大雑把なゴールの形」を思い浮かべながらそこに向かって具体的な証明を組み立てるということを繰り返す．よって，提案手法の「次に辿り着きたいゴールの大雑把な形をクエリとしてモデルに与え，そこに至る具体的な証明を生成させる」というやり取りは，自然な証明支援の実現になっている．

3.2 提案モデル

提案モデルは，深層学習モデルでの推論と重み付きランダムサーチを組み合わせて，ユーザの指定した中間証明状態に至るタクティック（とその引数）の列を生成する．深層学習モデルは，後段のランダムサーチが優先的に探索すべきタクティックを示す役割を担う．ランダムサーチアルゴリズムは，前段の深層学習モデルが示す優先度（確率）に基づいて幅を持った探索を行い，具体的なタクティック列の生成を行う．

表 1 Transformer のハイパーパラメータ

入力/出力の埋め込み次元数	128
レイヤー数	4
隠れ層の次元数	512
アテンションヘッドの数	4
ドロップアウト率	0.2

3.2.1 深層学習モデル

深層学習モデルは，現在のゴールの状態とユーザの示すクエリを入力として受け取り，ゴール状態をクエリに近づけるのに必要なタクティックとその引数を推論する．推論結果は後述する形式のベクトルで，後段のランダムサーチを誘導する各種の確率を保持する．

図 2 に深層学習モデルのネットワーク構造を示す．ASTactic は TreeLSTM を用いて AST の特徴ベクトル化を行っていたが，本研究は，実装の容易さや計算速度の速さなどから，Transformer モデルを中心にネットワークを構成する．

提案モデルの実行に必要な入力は，現在のゴール X_S と，ユーザが求めるゴールの形（クエリ）X_Q である．提案モデルでは，両者ともフラットな文字列としてモデルに入力される．

入力された X_S と X_Q は，それぞれ文字単位で対応する埋め込みベクトル列 X'_S，X'_Q に変換される．埋め込みベクトルは学習可能なパラメータであり，学習中に適切な表現を獲得する．埋め込みベクトルの次元数は，本論文の試験実装では 128 次元である．

ベクトル列に変換された X'_S，X'_Q は，それぞれ別の Transformer モデルに入力され，埋め込み V_S，V_Q に変換される．2 つの Transformer モデルは，全く同じ構造であるが，その重みは共有されていない．これらの Transformer のハイパーパラメータは表 1 の通りである．これらは，BERT モデル [2] に従って，入力ベクトル列を単一の埋め込みベクトルに変換する（図 3）．すなわち，先頭の入力に特別なトークン [CLS] を入力し，その後に続けて X'_S または X'_Q を入力する．最終層において，出力の先頭を取り出し，埋め込み V_S，V_Q とする．

2 つの Transformer から得られた V_S，V_Q は，結合されて V_{SQ} となり，これを全結合層に通して出力ベクトル列 $Y' = (Y'_1, Y'_2, \ldots, Y'_n)$ を得る．この出力の各ベクトル Y'_i は，現在のゴール X_S からクエリされたゴール状態 X_Q に至るための，i 番目のタクティックとその引数を表現したものである．その具体的な構造は図 4 に示すとおりである．ベクトルの先頭には，タクティックの種類を選択（優先度付け）する確率が配置され，その後に各タクティックのための引数の確率が並ぶ．このベクトルは，推論時に後段のランダムサーチが生成すべき各タクティックと引数の確率（優先度）を与え，ランダムサーチはその確率に基づき具体的なタクティックの生成を行う（推論時には最初の

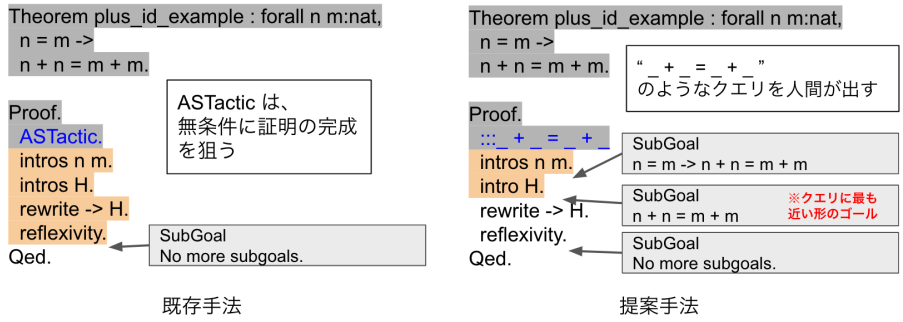

図 1 既存手法と提案手法の比較

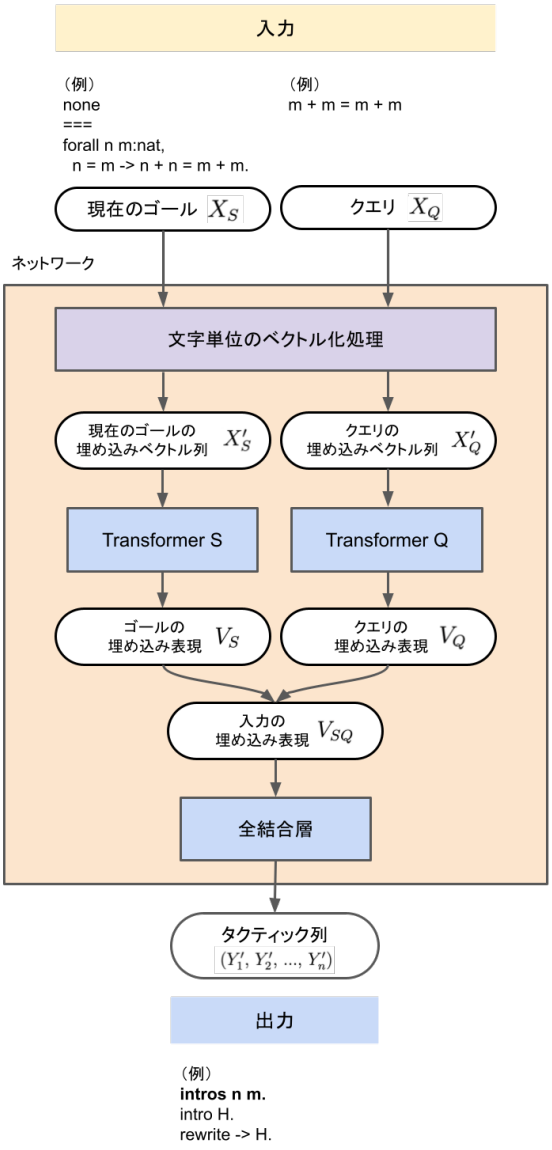

図 2 提案モデル

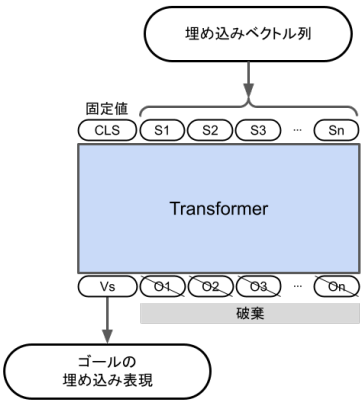

図 3 系列データの埋め込み

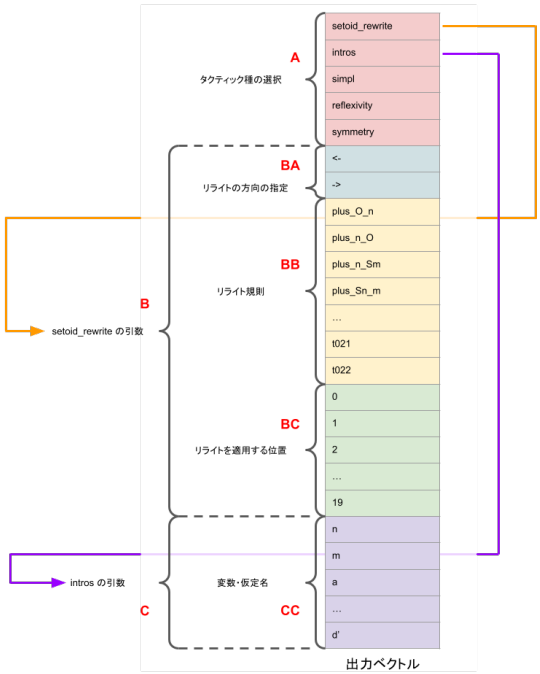

図 4 出力ベクトルの構造．同色の領域のそれぞれが確率ベクトル．

ベクトル Y_1 しか利用しない）．現状の提案モデルでは，利用できる定理や変数の種類はあらかじめ決められている必要があり，それによって出力の次元数が決まる（本論文の試験実装では 65 次元のベクトル）．教師データとしての出力ベクトルは，実際に使われたタクティックとその引数に対する onehot ベクトルを結合したものになる（他のタクティックの引数部分は零ベクトル）．

以上のように，提案のネットワークあ h，入力文字列 X_S，

Step 1: スタック St に，現在の Coq のゴール状態と空のタクティック列の組 $(X_S, [\,])$ を入れる．
Step 2: St から (X_S, T) を取り出す．取り出せないなら探索終了．
Step 3: X_S とクエリ X_Q との類似度が閾値を超えたとき，タクティック列 T を出力して探索を終了．
Step 4: T の長さが一定値以上なら，Step 2 へ戻る．
Step 5: 深層学習モデルに X_S，X_Q を与え，出力 Y'_1 を得る．
Step 6: Y'_1 から，その確率に基づいて複数個の具体的なタクティック（と引数）を生成する．
Step 7: 生成された各々のタクティック t に対し，状態 X_S の Coq で t が実行可能ならば，その実行後の状態 X'_Q を取得し，St に $(X'_Q, T +\!+ [t])$ を入れる．
Step 8: Step 2 へ戻る．

図 5 ランダムサーチアルゴリズム

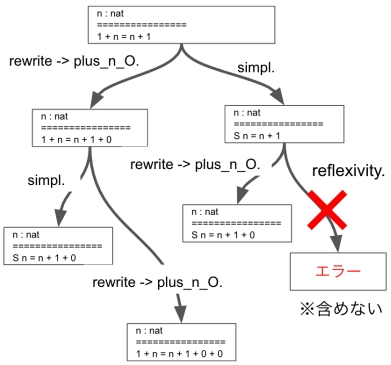

図 6 タクティック適用による状態遷移木

X_Q を受け取り，固定長の出力ベクトル列 $(Y'_1, Y'_2, \ldots, Y'_n)$ を出力する．推論時には，出力ベクトル列の最初のベクトル Y'_1 が，後段のランダムサーチが「次に使うべきタクティックとその引数」を構成する際の確率（優先度）として利用される．

3.2.2 ランダムサーチ

ランダムサーチは，前段の深層学習モデルが示す優先度（確率）に基づいて幅を持った探索を行い，具体的なタクティック列の生成を行う．この際，ひとつのタクティックが生成される度に，それを用いて遷移した Coq の新たなゴール状態と元のユーザのクエリに対して深層学習モデルによる「次のタクティック」の推論を行う．

具体的なアルゴリズムを図 5 に示す．全体としては素直な DFS であり，クエリに十分近い状態に至るまで，深層学習による推論に沿ったタクティック列の生成を行う．

Step 3 における類似度は，本論文の試験実装では X_S と X_Q をトークン列に分解した際のトークン一致の割合を用いた．この際，ワイルドカードは任意のトークンに対して一致するとした．また，閾値を 0.9 とした．

Step 4 で，生成するタクティック列の最大長（探索の深さ）に閾値を設けている．この閾値は，探索時間を観察しつつ実用的に利用できるように設定する．

Step 6 で深層学習モデルの出力 Y'_1 から具体的なタクティックをいくつ得るのか（すなわち，探索の幅）はアルゴリズムのパラメータとなる．初手となる探索の起点（root）は多くの候補を生成し，その後は少なめの候補を生成する．

なお，上記はスタックを用いてアルゴリズムを記述しているが，本論文の試験実装では再帰関数による実装を行った．また，Coq の状態変化には，ASTactic とともに提案された学習環境 CoqGym の API を用いた．

3.2.3 ハイブリッドランダムサーチ

前節のランダムサーチアルゴリズムの Step 6 において，Y'_1 からの具体的なタクティックの生成を決定的にひとつだけ行う，決定的アルゴリズムを考える．具体的には，確率が最大のタクティック（とその引数）を常に生成するアルゴリズムを考える．

ハイブリッドランダムサーチは，まず決定的アルゴリズムによるタクティック列の生成を試みて，それが失敗した際には元のランダムサーチアルゴリズムによる生成を行う．これにより，深層学習モデルの推論精度が高い場合の計算量の削減が期待できる．本論文の試験実装ではこのハイブリッドランダムサーチを用いた．

3.3 データセットの生成

提案モデルの学習に必要な情報は，その入力となる現在のゴール X_S とユーザの求めるゴールの形（クエリ）X_Q と，モデルが出力すべき「X_S から X_Q に至るためのタクティック列 T（の確率ベクトル表現 Y）」である．本研究では，このデータセットを以下の流れで準備した．

まず，CoqGym を介して Coq に適当な定理を入力する．この定理は既存の Coq 証明等から適当に用意する．

次に，あらかじめ用意したタクティックとその引数のリストをもとに，ランダムにタクティックとその引数を生成して Coq で実行し，実行に成功したものについてはその時のゴールと適用したタクティックを記録する．すなわち，図 6 に示すような遷移元の状態を親としたツリー構造を記録する．

最後に，出来上がった状態遷移のツリーからランダムにパスを抜き出し，一つの学習データとする（図 7）．パスの最初の状態のゴールが「現在のゴール X_S」，パスの最後の状態のゴールが「ユーザが求めるゴールの形（クエリ）X_Q」となる．ただし，X_Q に関しては，それに含まれる数値や変数をランダムにワイルドカードに置換する．そして，パスを構成する遷移に用いられたタクティックの列 T からその確率ベクトル表現 Y を作る（すなわち，onehot ベクトルを並べたものを構築する）．

4. 試験実装と評価

Python と PyTorch を用いて提案の深層学習モデルを構築し，学習及び評価を行った．

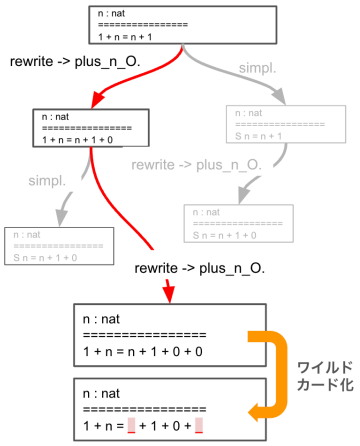

図 7 パスの抜き出しとクエリの生成

```
Axiom t000: ∀a b c:nat, (a+b)+c = a+(b+c).
Axiom t001: ∀a b:nat, a + b = b + a.
Axiom t003: ∀a:nat, 0 + a = a.
Axiom t100: ∀a:nat, 0 * a = 0.
Axiom t101: ∀a:nat, a * 0 = 0.
Axiom t102: ∀a:nat, a * 1 = a.
Axiom t103: ∀a:nat, 1 * a = a.
Axiom t104: ∀a b:nat, a * b = b * a.
Axiom t105: ∀a b c:nat, a*(b*c) = (a*b)*c.
Axiom t200: ∀a b c:nat, a*(b+c) = a*b + a*c.
Axiom t302: ∀a:nat, a - a = 0.
Axiom t303: ∀a:nat, a - 0 = a.
Axiom t400: ∀a b c:nat, a*(b-c) = a*b - a*c.
Axiom t900: 2 = 1 + 1.
Axiom t901: 3 = 1 + 1 + 1.
Axiom t910: 1 = S 0.
Axiom t911: 2 = S (S 0).
Axiom t912: 3 = S (S (S 0)).
Axiom t020: ∀a:nat, S a = a + 1.
Axiom t021: ∀a:nat, S (S a) = a + 2.
Axiom t022: ∀a:nat, S (S (S a)) = a + 3.
```

図 8 今回の提案モデルが使える定理

4.1 モデルの学習

図 8 の自然数の四則演算に関わる定理をモデルが使うことを前提に，前述の手法で生成した 108105 個の状態遷移ツリーから動的にパスを生成して学習に用いた．

モデルの学習は，1 台のデスクトップ PC を用いて行った．スペックは，OS：Ubuntu 22.04.1 LTS，CPU：AMD Ryzen 7 3700X，メモリ：32GB，GPU：GeForce RTX 2070 SUPER である．

学習率等のパラメータは予備実験を行い決定した．エポック数に制限は設けず，学習損失やテスト損失の推移をプロットして確認し（図 9），損失が安定して低下し変化が少なくなっていることや過学習の兆候が見られないことを確認して，学習を終了した．

4.2 定量的評価

深層学習モデルが，探索量の削減にどの程度寄与できているのかを明らかにするために，一様なランダムサーチによって証明を構築しようとするモデル（ランダムモデル）との比較を行った．具体的には，提案モデルに「出力ベクトルの値をランダムな数値に置き換える処理」を加えたものをランダムモデルとした．また，提案手法およびランダムモデルにおけるランダムサーチのパラメータは，探索の深さ制限を 5，10，15，20 とし，探索の幅制限（root）を 10000 とし，探索の幅制限（root 以外）を 1，2，3，4，5 とした．タイムアウトは全て 60 秒とした．

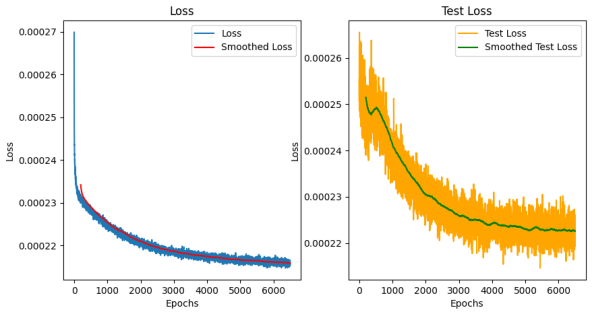

図 9 学習損失とテスト損失の推移

表 2 評価用のゴール変形

元のゴール状態 X_S	クエリ X_Q
Theorem a1: ∀n, n + 1 = 1 + n.	n + 1 = 1 + n
	n + 1 = n + 1
	n + 1 = S n
	S n = 1 + n
	S n = S n
	S n + 0 = S n + 0
Theorem a1: ∀n:nat, n * 2 = n + n.	n * (1 + 1)= n + n
	n * 1 + n * 1 = n + n
	n + n = n + n
	n + n + 0 = n + n + 0
Theorem a1: ∀n:nat, n * 3 = n + n + n.	n * (1 + 1 + 1)= n + n + n
	n * 1 + n * 1 + n * 1 = n + n + n
	n + n + n = n + n + n
	n + n + n + 0 = n + n + n + 0

提案モデルとランダムモデルに対し，表 2 に示す X_S と X_Q のペアの各々に対してタクティック列の構築を 10 回ずつ実行し，ゴール変形の成功率や探索数（タクティックの試行回数）を比較した．

ランダムサーチのパラメータ毎のそれぞれのモデルの，ゴール変形の成功率を図 10 に，探索数（タクティックの実行数）を図 11 に示す．全ての条件において，提案モデルがランダムモデルより高い成功率でゴール変形を達成していることがわかる．その成功率の比は，最大で 4.8 倍となった（探索の深さ制限: 20, 探索の幅制限: 1）．同様に，全ての条件において，提案モデルがランダムモデルより少ない探索数でゴール変形を達成していることがわかる．その探索数の比は，最小でランダムモデルの 15%の探索数となった（探索の深さ制限: 10, 探索の幅制限: 2）．

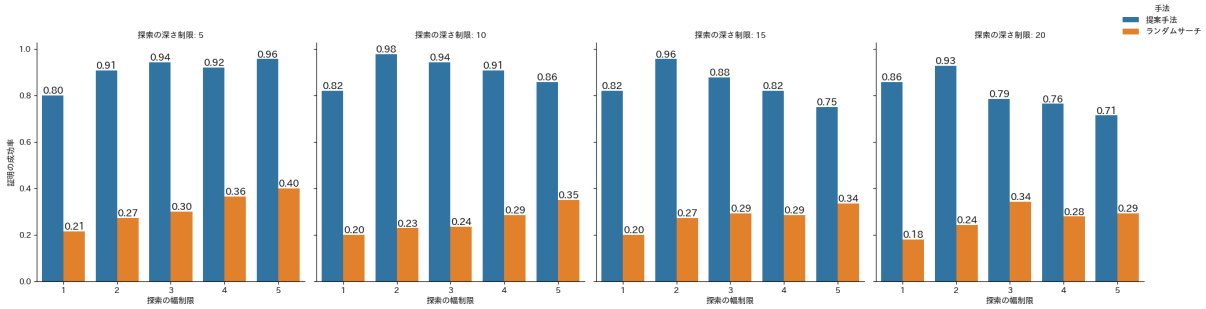

図 10　各条件におけるゴール変形の成功率

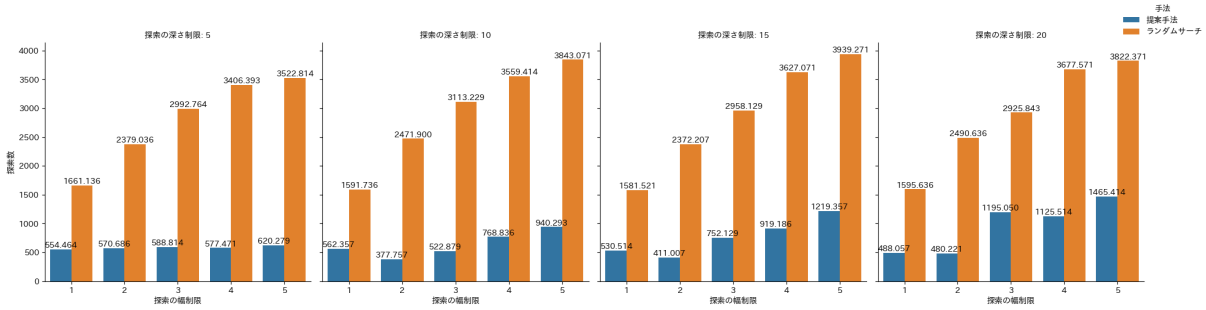

図 11　各条件における探索数

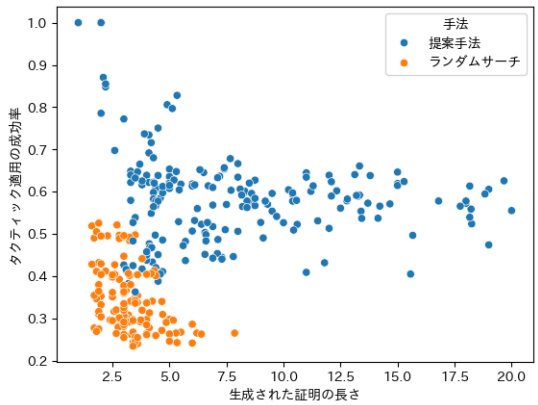

図 12　証明の長さとタクティックの適用成功率

ランダムモデルでは探索の幅制限が緩和されるにつれて証明の成功率が高まる傾向が見られるが，提案手法では逆の傾向を持っている（探索の深さ制限が 10 以上の場合において）．これは，ランダムモデルが生成したタクティックは適用に失敗することが多く，幅が広がるほどに深い探索地点に至れるようになるのに対し，提案モデルの生成するタクティックは成功しやすく，幅が広がると深い探索地点に留まってタイムアウトしがちになってしまう傾向があるからだと考えられる．実際，図 12 は証明の長さとタクティックの適用成功率の関係をプロットしたものであるが，提案モデルが比較的長い証明を生成している場合があり，その場合でも多くはランダムモデルよりタクティック適用の成功率が高いものになっていることが分かる．

いずれの結果からも，深層学習モデルによる探索の誘導が効果的に働いていることが確認できる．

表 3　アンケート項目と結果

質問	回答
質問 1: 現状の精度の提案モデルを活用することで、証明にかかる労力を削減できていると思いますか？	思う — 3
	思わない — 3
質問 2: 今後、提案モデルの精度を向上させることは有益だと思いますか？（自動証明モデルとしての方向性は正しいと思いますか？）	思う — 6
	高精度なものがある分には多少有益だが、より有用な代替がある — 0
	特に有益ではない — 0
	その他 — 0

4.3　定性的評価

定性的評価のために，提案モデルへのクエリを受け付ける IDE を作成し，Coq 利用経験者にその IDE 上で証明作業を行ってもらい，操作感などのアンケートを実施した．IDE は Web サービスとして実装し，評価者に各自の PC でアクセスしてもらう形で証明作業を実施してもらった．対象とした証明は，自然数の四則演算に関する簡単な等式とした．また，操作説明やアンケートも Web 上に公開したフォーム経由で配布し結果の収集を行なった．評価に先立ち，まず，評価者に提案モデルの役割や IDE の操作について説明する目的で，簡単なチュートリアルを行ってもらった．

アンケート項目とその回答結果については表 3 のとおりである．加えて，提案モデルの使用感等について，自由記述で意見を収集した．実際的な使用感についての参考情報として以下にまとめる．

4.3.1 手法について

「ワイルドカードの使い所が難しい」という意見があり，実際，今回の評価では評価者全員が利用していないことから，今回の評価で用いた程度の簡単な定理の証明ではワイルドカードの利用場面は少ない可能性がある．しかし，定理が複雑になる場面においてはワイルドカードによって目標とするゴールの形を大雑把に指定できることが利便性の向上につながると期待される．

また，「クエリに何を書けば良いのかわからない」という意見があり，クエリの考え方の共有に多少のコストがかかる可能性もある．

一方で，「中間ゴールの形を指定する方式自体は，実現すれば便利」や「狙ったものに寄せて証明を進めてくれるのはありがたい」という意見もあり，アンケートの質問 2 の結果と合わせて考えると，提案の手法については一定の支持が得られていると考えられる．

4.3.2 現状の精度について

「編集距離を小さくする方向に進むような探索の方が良さそう」や「同じクエリを複数回実行する必要がある」などの意見があった．アンケートの質問 1 の結果と合わせて考えると，現状の精度については改善が求められていると考えられる．

4.3.3 使用感について

「探索のランダム性で挙動が実行毎に変わってしまう点のコントロールができないか」という意見があった．乱数のシード値の固定や深層学習モデルのバージョン管理が今後の課題となり得る．

また，「適当に（思いつきで）ゴールを設定して，モデルが証明できるか試す遊びが面白い」という意見もあった．クエリの存在が Coq の利用に対する心理的障壁を下げる役割を持つ可能性が示唆されている．

4.3.4 その他

「（評価用 IDE について）ローカルの Coq 環境が必要ないのは良い」や「自分用の Coq の UI が欲しい」という意見があった．今回の提案モデルに限らず，「見せ方／扱わせ方 に関する研究」への需要を確認できたと考えている．

5. おわりに

本研究は，Coq における深層学習を用いた自動証明モデルに対し，証明の中間状態をユーザが提示できるような機構をモデルやユーザインターフェースに追加し，モデルに与えるタスク難易度の調整を行えるようにして証明生成の成功率と利便性の向上を図るアプローチを提案した．また，具体的なモデルや学習データの生成手法を提案し，その実装と評価を行った．

定量評価において提案モデルは，深層学習モデルを含まない純粋なランダムサーチモデルと比較して，ゴール変形の成功率は最大 4.8 倍高く，探索数（タクティックの試行回数）は最大 84 % 少なかった．また，ユーザによる定性評価では，現状のモデル精度には改善の余地が残されているものの，提案手法の方向性には一定の需要があることが確認された．

今後の課題として，モデル精度の改善，クエリの記述力の向上，利用できる定理の追加に再学習を要さない仕組みの組み込みが考えられる．例えば，木構造データ向けの Transformer [7] を用いて，ASTactic と同様に現在の証明状態を AST として与えることで現在の証明環境の認識を強化することが考えられる．また，クエリの記述力の向上には，単純なワイルドカードだけでなく，同じ部分式を表すメタ変数の導入が考えられる．

謝辞 本研究は JSPS 科研費 JP19K11903 の助成を受けたものです．

参考文献

[1] Bertot, Y. and Castéran, P.: *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*, Springer (2004).

[2] Devlin, J., Chang, M., Lee, K. and Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019*, pp. 4171–4186 (2019).

[3] Gonthier, G.: The Four Colour Theorem: Engineering of a Formal Proof, *Computer Mathematics, 8th Asian Symposium, ASCM 2007*, Lecture Notes in Computer Science, Vol. 5081, p. 333 (2007).

[4] Leroy, X.: Formal verification of a realistic compiler, *Communications of the ACM*, Vol. 52, No. 7, pp. 107–115 (2009).

[5] Redmon, J., Divvala, S. K., Girshick, R. B. and Farhadi, A.: You Only Look Once: Unified, Real-Time Object Detection, *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*, pp. 779–788 (2016).

[6] Rosenblatt, F.: The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, *Psychological Review*, Vol. 65, pp. 386–408 (1958).

[7] Shiv, V. L. and Quirk, C.: Novel positional encodings to enable tree-based transformers, *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, pp. 12058–12068 (2019).

[8] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W. and Kavukcuoglu, K.: WaveNet: A Generative Model for Raw Audio, *The 9th ISCA Speech Synthesis Workshop*, p. 125 (2016).

[9] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. and Polosukhin, I.: Attention is All you Need, *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pp. 5998–6008 (2017).

[10] Yang, K. and Deng, J.: Learning to Prove Theorems via Interacting with Proof Assistants, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, Vol. 97, pp. 6984–6994 (2019).