出力制約と表記正規化による LLM 形式証明のタクティク選択評価

門谷 拓能　神野 倫行　坂井 優介　渡辺 太郎
奈良先端科学技術大学院大学
{monya.hirotaka.mg5, jinno.tomoyuki.jx3}@naist.ac.jp
{sakai.yusuke.sr9, taro}@is.naist.jp

概要

LLM による Lean 形式証明生成では，最終正否判定は Lean の機械検証で定まる．しかし実用上は，**正しい証明の中から「参照タクティク列（問題特化タクティク）を同定できるか」**が重要であり，この同定には参照タクティク列との文字列一致（canonical）が有用な指標となる．一方で生出力には，説明文・コードフェンス・改行・別名などの表記揺れが混入しやすく，「実際には同等のタクティク列」であっても文字列一致が崩れて，タクティク選択が不安定になる．本稿はこの課題に対し，出力制約（Gate）と表記正規化（style-canon）によりタクティク列を安定な正規形へ写像し，canonical に基づくタクティク選択を頑健化する三段評価パイプラインを提案する．また評価データとして formal_bench v0.4 を作成した．実験を行った結果，出力制約が弱い条件では Lean 成功率が 1.00 でも canonical がほぼ 0 となり選択が破綻し得る一方，提案パイプラインにより canonical の一致率が回復し，問題特化タクティクの同定と比較が安定することを示す．

1　はじめに

LLM を数式処理や形式証明へ応用する研究が進む一方，Lean 形式証明生成では，正しい証明が複数のタクティク列で実現できるという性質がある．このとき実運用・大規模評価では，単に「正しいか（Lean が受理するか）」だけでなく，**参照タクティク列（問題特化タクティク）を安定に生成・同定できるか**が重要になる．例えば同じゴールを閉じても，探索的・冗長なタクティク列は実行時間やログ量を悪化させ得るため，参照タクティク列との文字列一致（canonical）により「狙ったタクティクが出ているか」を測りたい．しかしタクティク列には空白・改行・別名・括弧の有無など，同値だが異なる表記が複数存在し，生出力のままの文字列一致は表記差に敏感で，**タクティク選択（同定）が破綻する**．

なお最終正否判定は Lean 機械検証で十分であるが，本稿の目的は「正しい証明の中から問題特化タクティクを同定・比較する」ことであり，そのため canonical を主要指標として扱う．そこで本研究では，Gate（出力制約）と style-canon（表記正規化）を前段に置き，最終判定を Lean 実行に統一する三段評価パイプラインにより，表記揺れ由来の偽陰性を縮減する（図 1）．評価について，我々は formal_bench v0.4（以降，本ベンチ）[1] を作成し，本ベンチを用いて GPT-4o-mini と Claude 3.5 Sonnet の出力に対して，canonical 採点と Lean 検証を同一条件下で比較した．また補助的に，`ring_nf` の利用傾向など生成出力の軽量解析も行う．

本稿の研究課題は，RQ1：出力制約が弱い条件で canonical に基づくタクティク同定がどの程度破綻し，どの表記要因（改行・空白・別名等）が主因となるかの把握，RQ2：Gate と style-canon により canonical の一致率をどこまで回復でき，問題特化タクティクの選択（同定）と検証スループットをどこまで安定化できるかの検証，の 2 点である．

2　背景と関連研究

LLM による Lean 形式証明生成では，最終的な正否は証明器が受理するかで定まり，評価は success@1 や pass@k に基づく通過率で報告される [2, 3, 4, 5, 6]．一方，本研究のように「問題に特化したタクティク列を同定・比較する」目的では，参照タクティク列との文字列一致（canonical）が有用な評価軸となる．しかしタクティク列には空白・改行・別名など同値表記が多く，生出力のままの文字列一致は不安定になりやすい．

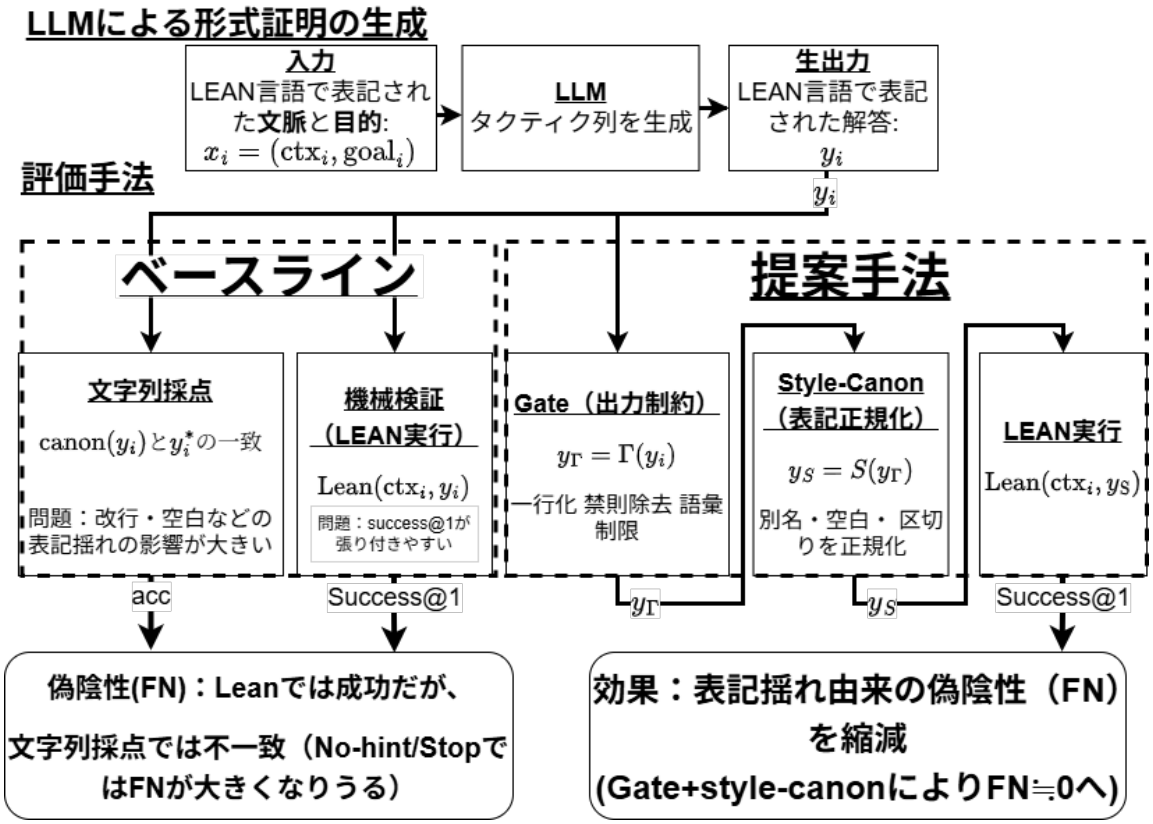

図 1: 文字列一致（canonical）と Lean 機械検証の不一致（偽陰性）と，提案する三段評価（Gate → style-canon → Lean 実行）の関係．図中の Γ, S, y_T, y_S は本文の $\mathcal{G}, \mathcal{S}, \mathcal{G}(y_i), \tilde{y}_i$ に対応する．

Lean 4 の普及 [7] と，LeanDojo による検証基盤 [8]，MiniF2F/ProofNet 等のベンチ整備 [9, 10] により評価は容易になった一方で，表記揺れに起因する偽陰性を**評価パイプライン仕様**として縮減する枠組みは限定的である．本稿は Gate（出力制約）と style-canon（表記正規化）を独立層として設計し，最終判定を Lean 実行に統一することで，文字列採点の偽陰性を抑えた再現性の高い評価手順を提示する．

3 提案手法

各問 i について，モデル出力 y_i に後処理を施し，$\tilde{y}_i = \mathcal{S}(\mathcal{G}(y_i))$ を Lean 4 で検証する．各問題 i は Lean の文脈 c_i（import や仮定）とゴール g_i からなり，モデルは一行のタクティク列 y_i を生成する．本研究では $\tilde{y}_i = \mathcal{S}(\mathcal{G}(y_i))$ を検証テンプレートに差し込み，Lean 4 がエラーなくゴールを閉じれば正解とする．

Gate（出力制約） $\mathcal{G}$ は，出力を「一行のタクティク列」へ整形する．具体的には，説明文・コードフェンス等の除去，複数行の連結と空白圧縮，区切り ; による連結形式への変換を行い，許可語彙集合 T（付録 A.2）以外の混入を抑止する．

style-canon（表記正規化） $\mathcal{S}$ は，$\mathcal{G}$ 後の一行タクティク列に対し，同値な表記差（別名・空白（改行を含む）・区切り等）を最小表現へ正規化する．証明内容は変更せず，受理安定性のみを改善する．また停止性・冪等性 $\mathcal{S}(\mathcal{S}(y)) = \mathcal{S}(y)$ を満たすよう設計する．Gate 併用下では FN が飽和し得るが，style-canon は，**canonical に基づくタクティク同定（参照列との一致判定）**を安定に行うための正規形化として機能する．

Lean 実行 正否は Lean 4 の受理可否で定める．ただし本稿の主眼は，正しい証明の中から参照タクティク列（問題特化タクティク）を同定できるかであり，その評価には canonical を用いる．

4 データセット：formal_bench v0.4

本節では，本研究で用いるベンチマーク formal_bench v0.4（JSONL，600 項目）の構造と作成手順を述べる．本ベンチは既存ベンチの問題（命題・証明スクリプト）を流用せず，著者が Lean 4 上で命題を設計し，収録形式（JSON による配布・命題単位の収録）といった体裁のみを公開ベンチ MiniF2F [9]，ProofNet [10] を参考にして独自に構築した．

収録形式（JSONL） データファイル `benchmarks/bench_v0_4.jsonl` は 600 行からなり，各行は {`id`, `split`, `lean_prop`, `domain`, `style`} を持つ．`lean_prop` は Lean 4 構文の文字列である．`id` は `hard_{family}_{k}` 形式で付与する（`k` は非負整数）．`family` は `ring_eq`, `linarith`, `nlin` などで，`_` を含み得るため，`id` の解釈は「末尾の `_{k}` を除いた部分」を `family` とする．`split` は評価用分割名であり，v0.4 は単一 split（固定：`ood_topic`）として提供する．`domain`/`style` は分析・集計用の補助タグであり，値の定義と対応は付録 A.5 に示す．

収録方針（family 設計） v0.4 では評価を「命題の難易度」ではなく「問題特化タクティクの同定（canonical 一致）の安定性」へ集中させるため，著者が Lean 4 上で代表的な 3 種のゴール（family）を設計し，各 family を 200 項目ずつ収録した（計 600 項目）．各 family では `lean_prop` を固定し，`id` のみを変えた 200 項目（`k=0..199`）を与えることで，成功率（Lean 受理可否）そのものではなく，文字列採点が表記差で崩れる状況を安定に観測できるようにしている．各 family では参照タクティク列 $y_i^\star$ を固定し，canonical 一致により「狙ったタクティクが出ているか」を安定に測る．

作成手順 各 family の代表命題を Lean 4 で設計し，型検査通過を確認した上で，ID と補助タグを付与して 200 件 ×3 family の JSONL を生成した．

参照タクティク列 $y_i^\star$（文字列採点用） 文字列採点（canonical accuracy）のため，各問に参照タクティク列 $y_i^\star$ を別途用意し，固定の形式正規化 canon(·) によって比較基準へ正規化した上で一致判定に用いる．これは最終判定（Lean 受理可否）とは独立に，**「Lean ではコンパイルされるが文字列一致では落ちる」**偽陰性（過少評価）を測る目的で導入した．なお実験で用いる生成プロンプトは再現性のため英語で固定し（付録 A.1），参照タクティク列 $y_i^\star$ の具体（family ごとの固定列）は付録表 5 に示す．

5 実験設定

比較可能性と再現性のため，図 1 のように，前処理・検証条件・指標定義を統一した．

ベンチマーク 評価には，著者が作成・公開する Lean 4 形式証明ベンチマーク formal_bench v0.4（$N = 600$）を用いる [1]．データ構造・作成手順の詳細は 4 節に示す．各問 i の入力 $x_i = (c_i, g_i)$ は，`lean_prop` を固定の検証テンプレート（`import` 等の文脈 c_i）に埋め込んで構成する．また，文字列採点に用いる参照タクティク列 $y_i^\star$ は 4 節で定義したものを用いる．

モデル群 対象モデルは GPT-4o-mini [11] と Claude 3.5 Sonnet [12] とする．既存の生成済み出力（JSON）を用いる場合も，同一の後処理・検証系（$\mathcal{G}, \mathcal{S}$，および Lean 実行）に載せ，条件間の比較が同一基準となるよう揃える．

条件（アブレーション） 後処理の有無を 4 条件で比較する（y_i: 出力，$\mathcal{G}$: Gate，$\mathcal{S}$: style-canon）．

- **No-hint/Stop**: $\tilde{y}_i \approx y_i$（生出力を停止条件で打切り）．
- **No-gate, style-canon**: $\tilde{y}_i = \mathcal{S}(y_i)$．
- **Gate-only**: $\tilde{y}_i = \mathcal{G}(y_i)$．
- **Gate+style-canon（提案）**: $\tilde{y}_i = \mathcal{S}(\mathcal{G}(y_i))$．

本稿では，提案条件における最終的な検証入力を

$$\tilde{y}_i = \mathcal{S}(\mathcal{G}(y_i))$$

と定義する．

検証環境と並列化 各問について，$\tilde{y}_i$ を検証用テンプレートの `by` 部に差し込み，`.lean` を生成して Lean 4 を実行する．600 問は 100 問 ×6 に分割して並列検証し，ログから集計値を得る．実行性能は items/s（検証件数/秒）で報告する．

指標 最終判定は Lean 4 の受理可否とし，$N = 600$ 問に対する `success@1` を算出する：

$$\texttt{success@1} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\big[\text{Lean}(c_i, \tilde{y}_i) \text{ が } g_i \text{ を閉じる}\big].$$

軽量な文字列採点として，canonical accuracy（acc）を次で定義する：

$$\text{acc} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\big[\text{canon}(\tilde{y}_i) = y_i^\star\big],$$

ここで canon(·) は全条件に共通の最小整形（説明文・コードフェンス除去，改行/空白圧縮，`;` 連結の整形）とし，Gate（許可語彙 T による制約，$\mathcal{G}$）とは区別する．$y_i^\star$ も同一の canon(·) で正規化済みとし，acc は「参照タクティク列と一致するか」を表す．

さらに，**「Lean ではコンパイルが通るが文字列採点では落ちる」**過少評価（偽陰性）を直接測るため，偽陰性率（false negative rate; FN）を次で定義する：

$$\texttt{FN} = \frac{\sum_{i=1}^{N} \mathbf{1}\big[\text{Lean}(c_i, \tilde{y}_i) = 1 \ \wedge\ \text{canon}(\tilde{y}_i) \neq y_i^\star\big]}{\sum_{i=1}^{N} \mathbf{1}\big[\text{Lean}(c_i, \tilde{y}_i) = 1\big]}.$$

$\text{Lean}(c_i, \tilde{y}_i) = 1$ が全問で成立する条件では，$\texttt{FN} = 1 - \texttt{acc}$ と解釈できる．なお評価では複数条件で

success@1 が 1.000 に張り付くため，本稿では **問題特化タクティクの同定**を表す acc を主要指標として報告し，FN は「同値表記により acc が崩れる割合」を示す診断指標として用いる．

補助指標として，ring_nf [13] 使用比率（提出列に ring_nf が出現した割合）と，タクティク長分布（; の個数+1）を集計し（付録表 3, 4），タクティク選好の差や規格化の影響を分析する．

6 実験結果

本節では，提案パイプライン（Gate ＋ style-canon）により canonical に基づくタクティク同定（acc）がどの程度安定化するかを示す．本ベンチでは success@1 が条件間で張り付きやすいため，本稿では acc を中心に，FN を「同値表記が同定を崩す割合」を表す診断指標として併記する．続いて，提案パイプライン下での生成挙動を補助的に把握するため，Lean の再実行を伴わず生成済み JSON から軽量に算出できる統計量として，(i) ring_nf 依存率と (ii) タクティク長分布を解析する．

6.1 主結果：評価ミスマッチの定量化と縮減

Lean 成功率とスループット 表 1 に示すように,Gate+style-canon 条件では，GPT-4o-mini と Claude 3.5 Sonnet のいずれも success@1=1.000（600/600）を達成した．

偽陰性（過少評価）の実態と縮減 表 1 より,No-hint/Stop 条件では，両モデルが success@1=1.000 を維持したにもかかわらず，acc は GPT-4o-mini で 0.000，Claude 3.5 Sonnet で 0.037 と低く，FN はそれぞれ 1.000，0.963 と極めて大きかった．すなわち，表記揺れ（改行・空白・別名等）が文字列一致を崩し，過少評価（偽陰性）が体系的に発生している．No-gate, style-canon では acc が 0.972/0.992（FN 0.028/0.008）まで回復し，不一致の大半が表記揺れとして吸収可能であることが分かる．さらに Gate 系（Gate-only ／ Gate+style-canon）では両モデルで acc=1.000，FN=0.000 となり，本ベンチにおける偽陰性は解消された．ただし表 1 の通り，本稿の設定では Gate の寄与が大きく，style-canon の効果は Gate 併用下では飽和して観測されにくい．その一方で，No-gate, style-canon が No-hint/Stop の偽陰性を大きく縮減していることから，偽陰性の主要因が表記揺れであり，style-canon が Gate を用いない設定でも有効なことが分かる．

表 1: 参照タクティク列の同定（acc）と診断指標（FN），および success@1 の比較．

内部名：GPT=gpt4o_mini, Claude=claude3.5_sonnet.

モデル	条件	acc	FN	success@1
GPT	No-hint/Stop	0.000	1.000	1.000
GPT	No-gate, style-canon	0.972	0.028	1.000
GPT	Gate-only	1.000	0.000	1.000
GPT	**Gate+style-canon**	**1.000**	**0.000**	**1.000**
Claude	No-hint/Stop	0.037	0.963	1.000
Claude	No-gate, style-canon	0.992	0.008	1.000
Claude	Gate-only	1.000	0.000	1.000
Claude	**Gate+style-canon**	**1.000**	**0.000**	**1.000**

補助解析（出力タクティクの傾向） 補助指標として，ring_nf 依存率とタクティク長分布を算出した（付録表 3, 4）．モデル間でタクティク選好に差はあるが，提出物の規格化により検証は安定する．

7 おわりに

本稿は，LLM による Lean 形式証明生成において，正しい証明の中から問題特化タクティク列を同定・選択するために canonical を主要評価軸として位置づけ，その不安定要因である表記揺れを Gate と style-canon により正規形化する評価パイプラインを提示した．

RQ1 として，No-hint/Stop のように出力制約が弱い条件では，canonical accuracy が極端に低い一方で Lean 成功率が 1.00 に達し得ることを確認した．これは，表記揺れ（冗長文，改行，空白，別名等）が文字列一致を崩して偽陰性を生む一方，Lean は同値なタクティク列を受理し得るためである．

RQ2 として，Gate（出力制約）と style-canon（表記正規化）を前段に置き，Lean 実行を最終判定とする三段パイプラインにより，表記揺れに起因する不一致（偽陰性）を縮減し，評価を安定化できることを示した．また，ring_nf の使用傾向にモデル間差があっても，提出物の規格化を介して Lean 成否と検証スループットが安定することを確認した．

一方で，本枠組みは許可タクティク集合 T の表現力に成功率が依存し，正規化により提出物の形式自由度が縮減されるため，単純な proxy 指標が難度差を反映しにくい．今後は，(i) nogate 対照による制約寄与の切り分け，(ii) blocked 実験による失敗類型の追跡，(iii) 目標型の拡張による難度レンジの拡大，および他証明器（Coq ／ Isabelle 等）への移植可能性の検証を進める．

参考文献

[1] Hirotaka Monya. formal_bench: Lean 4 benchmark (v0.4, 600 problems). GitHub repository, 2025. https://github.com/catlover-bot/formal_bench (accessed 2026-01-08). Dataset: `benchmarks/bench_v0_4.jsonl` (600 items).

[2] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. **arXiv preprint**, 2020.

[3] Guillaume Lample, Marie-Anne Lachaux, Thibaut Lavril, Xavier Martinet, Amaury Hayat, Gabriel Ebner, Aurélien Rodriguez, and Timothée Lacroix. Hypertree proof search for neural theorem proving. **arXiv preprint**, 2022.

[4] Peiyang Song, Kaiyu Yang, and Anima Anandkumar. Lean copilot: Large language models as copilots for theorem proving in lean. **arXiv preprint**, 2024. v3 (2025) available.

[5] Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. **arXiv**, 2025.

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Jared Kaplan, Ilya Sutskever, Wojciech Zaremba, et al. Evaluating large language models trained on code. **arXiv preprint arXiv:2107.03374**, 2021.

[7] Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In **Automated Deduction – CADE 28**, Vol. 12699 of **Lecture Notes in Computer Science**, pp. 625–635. Springer, 2021.

[8] Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. In **37th Conference on Neural Information Processing Systems (NeurIPS 2023), Track on Datasets and Benchmarks**, 2023. arXiv:2306.15626.

[9] Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. **arXiv preprint**, 2021.

[10] Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics. **arXiv**, 2023.

[11] OpenAI. Gpt-4o mini: Advancing cost-efficient intelligence, 2024. Accessed 2025-11-15.

[12] Anthropic. Claude 3.5 sonnet. https://www.anthropic.com/news/claude-3-5-sonnet, June 2024. Announcements. Published Jun 21, 2024 (updated Aug 28, 2025). Accessed 2025-11-15.

[13] Mathlib.tactic.ring.ringnf (mathlib4 docs). https://leanprover-community.github.io/mathlib4_docs/Mathlib/Tactic/Ring/RingNF.html. Accessed 2025-10.

A　補遺：プロンプト全文・追加ログ・補助表

A.1　生成プロンプト全文

Gate 用プロンプト

```
You are a Lean 4 tactic generator.
Output exactly one line of Lean tactic command(s)
that solves the goal.
Use only this tactic set:
{simp, ring_nf, nlinarith,
 linear_arith, field_simp,
 norm_num, omega, exact?}.
No commentary, no explanations,
no code fences, no prefixes.
Return a single line like:
simp; ring_nf
```

No-hint/Stop 系プロンプト

```
Solve the goal in Lean 4
with a tactic sequence.
Only output the tactic(s).
Do not include any comments
or extra text.
```

A.2　許可語彙集合 T（Gate のホワイトリスト）

Gate はモデル出力を「一行タクティク列」に整形する際，許可語彙集合 T のみを通過させる．本稿で用いた T を表 2 に示す．

表 2: Gate の許可語彙集合 T（ホワイトリスト）

区分	許可タクティク（文字列）
簡約・書換え	`simp`
代数正規化	`ring_nf`
算術系	`nlinarith`, `linear_arith`, `omega`
有理式処理	`field_simp`
数値評価	`norm_num`
補助	`exact?`

なお，Gate は ; により連結された戦術列（例：`simp; ring_nf`）を許可し，T 以外の語（解説文，コードフェンス，見出し，未知タクティク等）は除去または切り落としの対象とする．

A.3　補助表：ring_nf 依存率とタクティク長分布

表 3: Gate + style-canon における `ring_nf` 依存率（600 題中）

モデル	依存サンプル数	総数	比率
GPT-4o-mini (`gpt4o_mini_gate`)	200	600	0.333
Claude 3.5 Sonnet (`claude_3.5sonnet_gate`)	0	600	0.000

表 4: Gate + style-canon におけるタクティク長分布（件数，600 題中）

モデル	長さ=1	長さ=2–3	長さ=4–5	長さ ≥6
Claude 3.5 Sonnet	600	0	0	0
GPT-4o-mini	600	0	0	0

A.4　補助表：family ごとの参照（最適）タクティク列

表 5: formal_bench v0.4 における family ごとの参照タクティク列 $y^\star$（問題特化タクティク）

family	**$y^\star$（1 行タクティク列；正規化後の表記）**
`ring_eq`	`ring_nf`
`linarith`	`linear_arith`
`nlin`	`nlinarith`

domain/style/split と #items の対応は表 6 に示す．

A.5　formal_bench v0.4 の補助タグ定義（domain/style）

`domain/style` は集計用の補助タグであり，分割を表さない（v0.4 の `split` は固定で `ood_topic`）．`id` は `hard_{family}_{k}` 形式で，`family` は _ を含み得る．

表 6: v0.4 における family と補助タグ（domain/style）の対応（split は固定）

family	domain	style	split	#items
`ring_eq`	`algebra`	`eq`	`ood_topic`	200
`linarith`	`order`	`lin`	`ood_topic`	200
`nlin`	`ineq`	`nlin`	`ood_topic`	200