Stress-Testing the Reasoning Competence of Language Models With Formal Proofs

Konstantine Arkoudas*
System 2 Labs
konstantine.arkoudas@gmail.com

Serafim Batzoglou
Seer, Inc.
serafim.batzoglou@gmail.com

Abstract

We present a broad empirical study of state-of-the-art LLMs and LRMs (Large Reasoning Models) on PROOFGRID, a new battery of challenging but tractable logical inference tasks that form a domain-independent test of constraint-based reasoning. The tasks include proof writing and proof checking across propositional and equational logic. We also introduce two novel tasks: *proof inpainting* and *proof gap-filling*. Solving these problems requires tracking the global structure of a mathematical argument, writing hierarchical subproofs, maintaining coherence across nested assumptions, performing complex case analyses, applying inference rules, reasoning about identity and term rewriting, and reasoning about proofs themselves. Our experiments reveal impressive performance by top-tier models but also systematic failure modes. Along with the benchmarks, we release a new data resource comprising over 10K formal deduction problems and corresponding proofs.

1 Introduction

Large language models (LLMs) have rapidly advanced, finding applications in increasingly complex problems well beyond their original scope of narrow NLP tasks such as summarization, information extraction, and composition of short texts like emails. The recent development of the inference-compute paradigm, commonly powered by reinforcement learning, has given rise to so-called *large reasoning models* (LRMs). These models perform test-time search to identify promising completions that are more likely to yield correct answers, at the expense of higher inference costs. Driven by this scaling of test-time compute, LRMs have demonstrated dramatic improvements in logical and mathematical reasoning, enabling some models to even surpass human performance on some very challenging benchmarks, such as FrontierMath (Glazer et al. 2024).

However, most test batteries for logical reasoning are focused on the correctness of the final answer, typically determined by an exact-match comparison with the ground truth. This ignores the reasoning behind the answers and is likely to inflate performance, because models might arrive at the right answers for the wrong reasons, by latching on to superficial patterns. This evaluation paradigm is particularly unsuitable for mathematics, where proofs are not just a means to an end but the very foundation of the discipline. Thus, to properly assess the mathematical reasoning abilities of LLMs and LRMs, it is essential to evaluate whether they can construct and verify proofs. PROOFGRID addresses this gap by requiring models to write, check, and reason about formal logic proofs.

An advantage of using pure logic to evaluate reasoning ability is that the problems are entirely abstract and thus require what psychologists call *fluid intelligence*, as opposed to *crystallized* intelligence (Sternberg, Robert 2020). Fluid intelligence refers to innate reasoning ability employed to solve novel problems, while crystallized intelligence reflects the declarative knowledge and problem-solving know-how that accumulates through explicit instruction. Current benchmarks often allow models to leverage their vast stores of crystallized knowledge, acquired through their pretraining, enabling them to memorize their way to correct answers by employing shortcuts that are sometimes valid but are other times derived from spurious patterns. By focusing on form rather than content, symbolic-logic problems gauge pure reasoning competence.

In addition, most existing benchmarks no longer pose a serious challenge and are approaching saturation, as shown by near-ceiling performances from even general-purpose LLMs. PROOFGRID incorporates complex and structurally rich reasoning challenges that expose the limitations of even

*Corresponding author

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 12376–12394
November 4-9, 2025

the most capable models. We hope that this will stimulate continued research and development in this field.

The initial version of the PROOFGRID dataset contains about 10K problems and proofs for propositional logic and 1K problems and proofs pure equational logic. This is a data resource that we plan to expand continuously with new material. In addition to the data, we release a set of benchmarks based on *task slices* of the overall dataset. These are specific tasks based on *fixed* smaller subsets of the full dataset, each typically containing 200–500 instances. Using this approach, we benchmarked 15 prominent LRMs and LLMs across multiple tasks: proof checking, proof discovery, proof infilling, and recovery of proof gaps. Proofs have formal syntax and semantics and are expressed in a natural-deduction style that mirrors human inferential practices The rest of this document is structured as follows. Section 2 discusses related work; Section 3 describes the datasets of PROOFGRID; Section 4 presents the results of our evaluations; Section 5 presents some findings from analyzing the results; and Section 6 concludes.

2 Related Work

There are many datasets for evaluating a diverse array of reasoning skills, and while we cannot review all of them here, we can categorize them into 4 main groups depending on the type of reasoning on which they focus: (a) *natural-language reasoning*; (b) *mathematical and scientific reasoning*; (c) *generalist/hybrid reasoning*; and (d) *formal reasoning*.

Benchmarks in the first group probe the ability of models to reason in natural language. This includes classical NLI benchmarks such as SNLI (Bowman et al. 2015) and MNLI (Williams et al. 2018), which study entailment judgments of the form $T \models h$, where T is a context (or "theory") and h is a hypothesis. Both T and h are natural-language texts found in the wild, and the task is to determine if h follows from T, is contradicted by it, or neither. These determinations are typically supported by informal or commonsense reasoning and general background knowledge, and require relatively shallow inferences.

We include in this category datasets in which T and h are expressed in "controlled natural language" (CNL), namely, natural language that is not organic but rather synthesized from logic-formula templates, where the formulas come from a well-circumscribed logic such as Datalog or a particular description logic. These datasets, which tend to emphasize multi-step inference chains and closed-world reasoning, include RuleTaker (Clark et al. 2021), ProofWriter (Tafjord et al. 2021), ProntoQA (Saparov and He 2023), LogicInference (Ontañón et al. 2022), LogicAsker (Wan et al. 2024), LogicNLI (Tian et al. 2021), and FLD (Morishita et al. 2023). Folio (Han et al. 2024) can arguably be placed in the same group.

Math benchmarks include GSM8K (Cobbe et al. 2021), MATH (Hendrycks et al. 2021b), OlympiadBench (He et al. 2024), Omni-MATH (Bofei et al. 2024), and the more recent (and much more demanding) FrontierMath (Glazer et al. 2024) and OlymMATH (Sun et al. 2025). Problems here are expressed in natural language and the expected answers are literals such as numbers or symbolic expressions like $x^2 - 5$. These benchmarks do not test mathematical reasoning per se, since they follow an outcome-based evaluation paradigm that judges models only on their final answers. This allows models to exploit memorization, heuristics, and pattern matching. Scientific-reasoning datasets, like ARC (Clark et al. 2018), Entailment Bank (Dalvi et al. 2021), and TheoremQA (Chen et al. 2023) focus on natural science rather than mathematics, but they follow essentially the same pattern: questions are posed in natural language and probe problem-solving in knowledge-rich domains; only the answers are evaluated, not the reasoning.

The third group comprises broad-coverage benchmarks like MMLU (Hendrycks et al. 2021a) and its successor MMLU Pro (Wang et al. 2024), HLE (Humanity's Last Exam) (Scale AI and Center for AI Safety 2025), BBH (Suzgun et al. 2023) and its successor BBEH ("Big Bench Extra Hard") (Google Research 2025), and SuperGPQA (Team et al. 2025). These provide a coarse picture of general model ability but blur distinctions between domain knowledge, language understanding, and reasoning. They also remain outcome-focused.

The last category includes formal-reasoning datasets that use symbolic inputs rather than natural language. A notable example here is MiniF2F (Zheng et al. 2022), which contains 488 mathematics problems expressed symbolically in Lean, Metamath, Isabelle, and HOL Light, and whose solutions require formal proofs in these systems. In principle, such benchmarks can verify not just the correctness of an answer but also the correctness of the reasoning behind the answer, since the

reasoning is expressed as a formal proof that can be checked by machine. However, such benchmarks tend to be tied to specialized proof systems, restricting their accessibility.

PROOFGRID falls into this last group, since its inputs are based on symbolic formulas and its outputs require structured formal proofs. It mitigates the issue of heavy dependence on esoteric details of complex formal systems by employing a minimalist natural-deduction formalism, which is both strikingly compact (the entire proof language is fully described in a short prompt) and aligned with ordinary reasoning practices. This should make PROOFGRID both rigorous and approachable.

PROOFGRID can also be compared with the CNL members of the first group (benchmarks for "natural-language reasoning"), since they are also based on formal logics. Relevant points of comparison include (i) the expressiveness of the underlying formalism; (ii) the range of reasoning mechanisms tested; (iii) the maximum depth of reasoning required; and (iv) the current degree of saturation. Regarding expressiveness, some of the CNL datasets, such as RuleTaker and ProofWriter, are based on stratified Datalog, which supports recursive Horn clauses and limited negation but disallows full disjunction, unrestricted negation, quantifier alternation, and function symbols. Likewise, ProntoQA is based on positive Horn Logic, which is reducible to Datalog over unary predicates. In theory, FLD supports full first-order logic, albeit without equality or function symbols. In practice, the fragment it exercises is far more limited, essentially confined to monadic predicates without quantifier alternations. This reflects a broader trend among benchmarks in the CNL category. Even though they are solidly framed in logic, most of them end up targeting very weak subsets of predicate logic, partly because of the sheer complexity of the infrastructure that is needed for a correct implementation of full-blown proofs in first-order logic. Similar remarks apply to LogicInference, LogicAsker, LogicNLI, and Folio. Consequently, these benchmarks end up testing mostly variants of simple forward chaining on conditional rules, which requires little more than universal specialization and modus ponens.

By contrast, the current version of PROOFGRID is based on full propositional logic (with unrestricted negation and disjunction) and equational logic. The former allows us to encode a vast range of reasoning-intensive problems requiring proofs, many of which cannot be represented in Datalog (e.g., problems involving parity and counting in general, graph connectivity, pigeonhole constraints, injectivity, mutual exclusion, and others). Equational logic allows us to test the ability of AI models to reason about function applications and identity, which is beyond the scope of the CNL benchmarks. Moreover, PROOFGRID supports a diverse range of reasoning idioms, including hypothetical reasoning (conditional proofs under provisional assumptions), which requires the tracking of nested assumption scopes, reasoning by contradiction and indirect proof, complex and highly nested case analyses, and so on. These are foundational tools in mathematical reasoning and crucial for evaluating whether a model understands how to structure and manage complex deductive arguments, not just chain together atomic rules.

The maximum depth of reasoning required by CNL benchmarks is smaller than 10 (the most demanding is FLD, with a maximum depth of 8). PROOFGRID problems feature reasoning chains with dozens of steps (in some cases over 100 steps). Finally, the CNL benchmarks are largely saturated (Zhou et al. 2025), with DeepSeek-R1 exceeding 80% overall accuracy on FLD, Folio, and ProntoQA, and even GPT-4o scoring over 98% on ProntoQA. By contrast, the mean accuracies of DeepSeek R1 and GPT-4o on PROOFGRID are 35% and 13%, respectively.

In summary, PROOFGRID differs from prior reasoning benchmarks in four key ways: it enforces proof-based evaluation rather than outcome-only scoring; it minimizes content-knowledge advantages by using abstract symbolic inputs; it employs a minimalist natural-deduction framework that still supports a wide range of common reasoning mechanisms; and it introduces challenging tasks that even the best current models cannot solve reliably.

3 Datasets

The inaugural version of PROOFGRID[1] has two parts, one for propositional logic and one for equational logic. The former has 2 datasets, PL_1 and PL_2, and 4 tasks: proof checking; proof writing; proof infilling; and proof gap-filling. The equational-logic part has one dataset, EQ_1, and 3 similar tasks (proof checking, proof writing, and proof gap-filling). We describe all of them in detail below.

All proofs in PROOFGRID are formal, and those

[1] https://github.com/System-2-Labs/ProofGrid/

in PL_1 and PL_2 are written in *natural deduction* style. Natural deduction makes direct and intuitive use of techniques like conditional proof, reasoning by contradiction, case analysis, and application of inference rules that reflect the way human mathematicians and scientists construct and verify arguments. To evaluate and write these proofs, a reasoner must be able to recognize local and global logical relationships, track assumption and lemma scope, and compose multiple steps into coherent chains of inference. Because natural deduction aligns closely with human inferential practice, both formally and cognitively, it offers an ideal testbed for probing the reasoning competence of AI models.

Another advantage of the particular natural-deduction system used in PROOFGRID is its simplicity and autonomy. Models do not need to have mastered the intricate syntax of complex interactive theorem-proving systems, either via pretraining or fine-tuning. Instead, our PL_1 tasks rely solely on short but rigorous and self-contained prompts describing the formal syntax and operational semantics of an exceptionally simple and stripped-down natural-deduction language, NDL, along with a few well-chosen ICL examples. Our EQ_1 proofs are even simpler (essentially sequences of term rewrites); their syntax and semantics are also fully described in short prompts and illustrated with examples.

PL_1 Data

PL_1 is a relatively simple dataset comprising short and minimally structured problems and proofs. LRMs do well both on proof-checking and on proof-writing in PL_1, but we show that proof masking and gap-filling pose challenges even in this basic setting.

PL_1 contains 1.4K problems, where each problem consists of a set of premises $p_1, \ldots, p_n$ and a conclusion p that is logically entailed by the premises. The p and p_i are formulas constructed inductively from propositional atoms A, B, C, ... A complex formula is either a negation $(\sim p)$; a conjunction $(p_1 \,\&\, p_2)$; a disjunction $(p_1 \mid p_2)$; a conditional $(p_1 \Rightarrow p_2)$; or a biconditional $(p_1 \Leftrightarrow p_2)$. The negation operator $\sim$ binds most tightly, followed by conjunctions, disjunctions, conditionals, and biconditionals. All binary constructors are right-associative.

Premises and goals were randomly generated as tree structures; see Appendix B for details.

Here is a sample proof-writing problem: derive $p = (B \mid D)$ from $p_1 = (A \Rightarrow B)$, $p_2 = (\sim A \Rightarrow C)$; and $p_3 = (C \Rightarrow D)$. Appendix B contains additional examples and various statistics on the generated problems.

Proofs for all problems were generated automatically by a theorem prover written in Athena (Athena Language Team 2025), a natural-deduction system for proof engineering that can be viewed as a proper superset of NDL. All proofs generated by this theorem prover were confined to the NDL core. Here is an example of a formal proof for the problem described above:

```
assert premise-1 := (A ==> B)
assert premise-2 := (~ A ==> C)
assert premise-3 := (C ==> D)

# We prove (B | D) by a case analysis on (A | ~ A),
# which holds by the law of the excluded middle.
{
  (A | ~ A) BY ex-middle on A;

  # Case 1: Show that A implies (B | D)
  assume A {
    # Applying modus ponens to premise-1 and the
    # assumption A gives B:
    B BY mp on premise-1, A;
    # So now (B | D) follows by
    # disjunction introduction:
    (B | D) by left-either on B, D;
  };

  # Case 2: Show that (~ A)  implies (B | D)
  assume (~ A) {
    C BY mp on premise-2, (~ A);
    D BY mp on premise-3, C;
    (B | D) BY right-either on B, D;
  };

  # The case analysis is now complete:
  (B | D) BY cases on (A | ~ A),
                      (A ==> B | D),
                      (~A ==> B | D)
}
```

As mentioned above, NDL is a minimal subset of Athena that contains only introduction and elimination rules, conditional subproofs, and a few other convenient derived inference rules, such as De Morgan and disjunctive syllogism.

The theorem prover generated correct proofs by construction. To obtain a balanced dataset with roughly equal numbers of correct and incorrect proofs, we (a) randomly corrupted a number of these proofs so as to make them incorrect, and we (b) included proofs written by o3 and Gemini-2.5-Pro, which tended to have more organic errors than those produced by the noising procedures described in (a).

PL_1 Task 1: Proof Checking

This task contains 300 instances. The objective is simple: Given a proof for a given problem, determine if it is correct. An incorrect verdict must be accompanied by additional information: the line

number containing the earliest error, the type of the error (syntax error, type error, or logic error); and a detailed description of the error. The full prompt can be found on the PROOFGRID repo.

Ground truth is determined by an instrumented version of the Athena proof checker, designed to make the checker compatible with the NDL description given in the prompts. This instrumentation is necessary because some of these 300 proofs were written by models and contain syntax forms that are interesting *generalizations* of NDL's syntax forms, but are not, strictly speaking, allowed by NDL proper. For instance, in NDL an inference rule R is always applied to a number of plain formulas $p_1, \ldots, p_n$. But models started taking the liberty of writing entire subproofs inline for various p_i. Athena actually allows this, but NDL does not, so instrumentation is needed to ensure alignment, so that when a model later is asked to check such a proof and complains that the proof violates NDL syntax, it receives full credit. As another example, case analyses are always binary in NDL: the ternary inference rule `cases` is applied to a binary disjunction $(p_1 \mid p_2)$ and two conditionals of the form $(p_1 \Rightarrow q)$ and $(p_2 \Rightarrow q)$. But some models started applying `cases` to more complex disjunctions of the form $(p_1 \mid p_2 \mid \cdots \mid p_n)$ for $n > 2$, along with n conditionals $(p_i \Rightarrow q)$ for $i = 1, \ldots, n$, for a total of $n + 1$ arguments. This is also allowed by Athena, but it is nowhere mentioned in the NDL definition in the proof-checking prompt, so if a model later checks such an LRM-generated proof and complains that `cases` is applied to too many arguments, it should be given credit, even if Athena accepts such liberal applications of `cases`.

Our evaluation methodology for this task is lenient. If the proof is correct (according to this modified checker) and the model reports it as correct, we give the model a full point. We also give the model a full point if the proof is incorrect and the model reports it as such, although a manual review of error details reveals that in some cases models report spurious errors.

PL_1 Task 2: Proof Writing

This task contains 400 instances. Given a deduction problem ("Derive p from $p_1, \ldots, p_n$"), the model must write a proof for it, including English comments explaining the reasoning strategy. Accuracy here is evaluated by checking the generated proofs with Athena. We are lenient with respect to minor syntax errors: we wrote scripts that patch up a wide range of common syntax errors made by the models (such as failing to balance parentheses) before we send the result to Athena for checking. We believe it is possible to likewise detect and repair a good deal of minor non-syntactic errors as well, in which case these accuracies would be expected to go up; this is left for future work.

PL_1 Task 3: Proof Infilling

The data for this task was obtained by masking 400 proofs from the PL_1 dataset. Given a proof, we randomly mask any number of the following four types of items: (a) conclusions (formulas immediately preceding the **`BY`** keyword); (b) inference rules (immediately preceding **`on`**); (c) assumptions (immediately following **`assume`**); and (d) one or more arguments to an inference rule application (immediately following **`on`**). Masks are unique identifiers: `MASK1`, `MASK2`, and so on.

The objective is to determine if there is a way to unmask the proof completely so that it correctly derives the target conclusion from the given premises; and if so, to provide a detailed assignment of appropriate values to the masks. This is essentially a constraint satisfaction problem where the constraints are given by the syntax rules and operational semantics of proofs. The key question addressed by this task is whether a model can infer missing symbolic structure consistent with the rules that govern natural-deduction reasoning, and in particular whether it can jointly restore syntax and semantics. High accuracy in this task would strongly suggest that the model possesses a robust internal model of logical reasoning that goes beyond surface-level pattern recognition.

Model responses are evaluated by unmasking the proof according to the specified mask values and then checking that the resulting proof correctly derives the expected conclusion. If a model claims that the problem is unsolvable, we consider that response as incorrect when the original (unmasked) proof succeeds and correct when the original proof fails. This evaluation is deliberately lenient, because masking an erroneous proof might salvage it by occluding the error. Thus, the accuracies reported in Section 4 for this task are upper bounds on actual performance. By contrast, when we restrict evaluation to *correct* proofs that have been masked, then an unsolvability claim is always incorrect and we count it as such. Accuracy restricted on the set of correct proofs is reported in Section 4 as "strict" accuracy. As expected, a model's strict

accuracy on proof infilling tends to be lower.

PL_1 Task 4: Proof Gap Filling

This dataset was obtained by randomly inserting holes (gaps) in a set of 200 correct proofs, written either by the Athena theorem prover or by a model. Unlike masks, which are locally focused on individual formulas and rules, gaps apply at the subproof level, i.e., a gap replaces a subproof, which could be either a single step (inference rule application), or a conditional subproof, or a *proof chain*: a sequence of consecutive subproofs. A proof usually receives several gaps, in different places. More information can be found in Appendix B.1.2. We ask the models to map every gap to a subproof in such a way that the result becomes a correct proof of the goal from the premises. A solution is counted as correct iff it achieves that. We report accuracy based on this notion of correctness. Since this task also involves the writing of proof fragments, we extend the same leniency toward minor syntax errors as in the proof-writing task.

PL_2

PL_2 consists of 300 much more structurally rich propositional-logic problems. They are grouped as follows: (1) 10 pebbling problems on DAG pyramids; (b) 5 counting-principle (partition) problems; (c) 15 problems for odd-numbered De Bruijn formulas; (d) 30 relativized pigeonhole-principle problems; (e) 50 Tseitin-formula problems; (f) 70 subset-cardinality problems; (g) 70 graph-coloring problems; and (h) 50 simple DAG pebbling problems encoding Horn-clause inference. We provide more details in Appendix B.2. The structure of all 300 problems is similar to that of PL_1: each problem consists of a number of premises and a target conclusion. In all 300 cases, the conclusion does follow from the premises. In this paper we only study proof writing in PL_2.

The task involves NDL_f, an even simpler version of NDL that replaces all inference rules with a single inference syntax form: p **FROM** $p_1, \ldots, p_n$. The semantics of this form are simple: conclude p on the basis of $p_1, \ldots, p_n$, where each p_i is either a premise or an active assumption or a result deduced earlier in the proof, as long as (1) p indeed follows from $p_1, \ldots, p_n$; and (2) $n \leq 5$.

This specification is operationalized as follows. First, we check to make sure that $n \leq 5$ and that each p_i is in the current assumption base (meaning it's a premise, or an active assumption or previously derived conclusion). If not, an error is reported. Otherwise we make a call to a SAT solver to see if p follows from $p_1, \ldots, p_n$. If it does, the step is accepted, otherwise an error is reported.

The restriction $n \leq 5$ is imposed to limit the granularity of the inferences that a model can make with **FROM**. If no limit is placed, a model could avoid honest toil by generating an one-line proof: *goal* **FROM** *premises*. Requiring $n \leq 5$ is a simple and effective way to preclude such pseudo-proofs.[2]

In this version of NDL, models are free to take much larger reasoning steps without bothering with tedious details. This ability to formulate helpful but straightforward lemmas without having to prove them in detail is crucial for PL_2, because these problems are considerably more complex; demanding fully detailed NDL proofs for them would be neither fair nor realistic. It is also a more accurate reflection of how mathematicians actually work.

While PL_2 problems are complex, they may be viewed as easy-hard problems. They are hard insofar as their general problem families are known to be challenging. But they are easy in that we make sure to only use problems whose size and difficulty are strictly bounded, in order to keep the tasks realistic and feasible. First, we ensure that the CNF encodings for the vast majority of problems (over 95% of them) contain no more than 100 clauses each (although we do not use CNF in the dataset, retaining full propositional-logic notation throughout). Second, we ensure that every goal has a resolution proof with fewer than 150 steps. Note that we generated most of these problems as unsatisfiability proofs for a list of formulas, but we then converted them to forward inference problems by taking the premises to be the tail of the list and the goal to be the negation of the head of the list.

We solved each problem with Vampire and analyzed the output resolution proofs to ensure that the overall number of inference steps was less than the maximum of 150 (we do not count CNF conversion steps). We then translated all resolution proofs produced by Vampire into NDL_f natural-deduction proofs, which became the raw material for the proof-checking task: determine if the given

[2] In principle, a model could circumvent this definition by gradually merging premises into a single conjunction, using no more than four at a time, and then deriving the goal in one large **FROM** step. We have not observed any such attempts. In any case, simple extensions of the $n \leq 5$ rule could block such contrived strategies as well (e.g., by also enforcing $m \leq 5$, where m is the total number of conjuncts across all arguments that represent premises).

NDL_f proof is correct. All 300 NDL_f proofs are correct by construction, so a model would attain perfect proof-checking accuracy if it deemed every proof to be correct. The proof-writing task was simply to write a proof in NDL_f that derives the goal from the premises.

Equational Logic

EQ_1 contains 1K purely equational proofs. Each example includes a set of equational axioms $\{E_1, \ldots, E_m\}$ and a derivation of an identity $s = t$ from those axioms. Each axiom is of the form

$$\forall\, x_1, \ldots, x_n \,.\, L = R$$

and is given a unique name. L and R are standard first-order terms, with variables and/or constants at the leaves and function symbols with arity greater than 0 at internal nodes, e.g., $g(a, f(X))$. Variable names start with upper-case letters and function symbols with lower-case letters. A proof is correct iff for every step $s_i = s_{i+1}$ **by** $E_{i_1}, \ldots, E_{i_k}$, the term s_{i+1} is correctly derived from s_i by the cited equations. All of the proofs were initially correct by construction. About half of them were then synthetically corrupted. Details and examples can be found in Appendix C. We study the following three tasks in EQ_1.

Proof Checking: This task contains 200 samples from EQ_1, about half of which are correct. Checking an equational proof means verifying that each step goes through owing to the cited identities. If the model determines that a proof is correct, it must not only pronounce it correct but also output a sequence of detailed structured explanations, one for each step of the proof. If the model says that the proof is incorrect, it needs to indicate the first step where the proof goes wrong and detailed about the error, including positional information. Credit is given only if the model gets all the details right.

Recovery of Elided Equations: This task, which also has 200 samples, omits the justifying equations from every step of the proof and asks the model to find them. Here we report two accuracy quantities, one in which a problem is considered correctly solved only if the model recovers all needed equations for *every* step of the proof; and one that records the number of steps (out of the total number of steps across all proofs) for which the model has recovered the right equations.

Gap Filling: This task, with 96 instances, inserts a gap of a randomly chosen size at a randomly chosen location in a given proof and asks the model to fill that gap with an appropriate sequence of steps. A problem here is solved correctly iff the model provides a sequence of steps for the gap that makes the proof valid.

4 Results

We use the following abbreviations:

```
    opus-4:  claude-opus-4-20250514
        r1:  deepseek-r1-0528
     ds-v3:  deepseek-v3
  sonnet-4:  claude-sonnet-4-20250514
magistral:  magistral-medium-2506
    grok-3:  grok-3-beta
     llama:  llama-3.1-405b (by Nous Research)
```

Accuracies for proof checking and writing in PL_1 are given in Tables 1 and 2, respectively.

Model	Acc	Model	Acc
gpt-5	**0.86**	o3	0.85
gemini-2.5-pro	0.84	opus-4	0.81
grok-4	0.78	o3-mini	0.76
o4-mini	0.75	r1	0.72
sonnet-4	0.64	llama	0.64
ds-v3	0.63	gpt-4.1	0.63
grok-3	0.60	magistral	0.56
gpt-4o	0.54		

Table 1: PL_1 proof-checking accuracies

Model	Acc	Model	Acc
grok-4	**0.63**	gpt-5	0.62
gemini-2.5-pro	0.59	o3	0.55
opus-4	0.46	r1	0.39
o4-mini	0.38	sonnet-4	0.35
grok-3	0.29	gpt-4.1	0.28
o3-mini	0.26	ds-v3	0.26
magistral	0.13	gpt-4o	0.11
llama	0.08		

Table 2: PL_1 proof-writing accuracies

Proof unmasking and gap filling results for PL_1 are shown in Table 3. PL_2 proof-writing results appear in Table 4. These numbers reflect syntax repairs (such as parenthesis balancing) that we performed automatically on all generated PL_2 proofs. Performance is considerably lower without these repairs (e.g., the raw accuracies of the top four models are as follows: 0.43 for Grok-4, 0.37 for GPT-5, 0.33 for Gemini-2.5-Pro, and 0.21 for o3). Finally, EQ_1 results are shown in Table 5.

5 Analysis

Models generally performed well on checking the simple proofs of PL_1, and even on generating them,

Model	Proof Infilling		Gap Filling
	Accuracy	Strict Accuracy	Accuracy
gpt-5	0.62	0.53	**0.85**
o3	0.65	0.55	0.59
gemini-2.5-pro	0.65	**0.62**	0.56
grok-4	**0.70**	0.58	0.58
o4-mini	0.52	0.40	0.28
o3-mini	0.34	0.30	0.18
r1	0.51	0.36	0.23
grok-3	0.16	0.16	0.32
opus-4	0.37	0.32	0.14
sonnet-4	0.23	0.23	0.13
ds-v3	0.24	0.13	0.18
gpt-4.1	0.15	0.09	0.18
llama	0.07	0.10	0.13
magistral	0.16	0.05	0.12
gpt-4o	0.26	0.04	0.11

Table 3: Performance on PL_1 proof infilling and gap filling.

Model	Acc	Model	Acc
grok-4	**0.51**	gpt-5	0.40
gemini-2.5-pro	0.36	o3	0.23
r1	0.14	opus-4	0.11
o4-mini	0.07	sonnet-4	0.06
ds-v3	0.05	grok-3	0.05
o3-mini	0.05	magistral	0.03
llama	0.03	gpt-4o	0.02
gpt-4o	0.01		

Table 4: PL_2 proof-writing accuracies

but they did worse on proof unmasking and gap filling. Performance drops sharply on PL_2, with only Grok-4 managing to reach 50% accuracy.

In general, we find that models continue to make elementary errors, and decoy problems reveal that they are often willing to accept false presuppositions. To illustrate, when asking LRMs to prove even-indexed instances of the De Bruijn formula (see Appendix B.2.5), most of them generate fabricated proofs. In addition, slight changes in problem presentation can have a significant impact on performance. A notable example is *conditionalizing* a problem (importing its premises into the theorem itself): while logically equivalent, this reformulation makes the conclusion longer and more complex, which degrades the performance of even the best models, especially on larger instances.

Model	PC	ER-G	ER-L	Gaps
gpt-5	**0.67**	**0.77**	**0.98**	**0.56**
grok-4	0.54	0.60	0.94	0.55
o3	0.52	0.67	0.96	0.33
gemini-2.5-pro	0.52	0.77	0.98	0.43
o4-mini	0.41	0.30	0.81	0.15
o3-mini	0.25	0.39	0.83	0.03
opus-4	0.25	0.33	0.90	0.16
r1	0.24	0.39	0.80	0.17
sonnet-4	0.22	0.29	0.85	0.07
grok-3	0.13	0.12	0.74	0.02
ds-v3	0.06	0.09	0.72	0.02
llama	0.05	0.01	0.16	0.01
gpt-4o	0.04	0.00	0.04	0.01
gpt-4.1	0.02	0.07	0.47	0.03
magistral	0.01	0.02	0.30	0.01

Table 5: Performance on EQ_1 tasks. **PC**: equational proof-checking accuracy; **ER-G** and **ER-L**: global/local ER accuracy.

On the other hand, a closer look at the detailed results reveals many instances of exceptionally strong performance on hard reasoning problems. Many models did well on DAG-pyramid pebbling problems, for example. These are fundamentally about dependencies, resource constraints, and sequential execution; several AI planning problems can be translated into SAT using pebbling-like encodings. The DAG-pyramid pebbling problems are particularly interesting because a naive approach to them would require exponentially long proofs. Only systems capable of structured reasoning with abstraction and lemma reuse can tackle these problems successfully as the height of the pyramid increases.

We asked all models to describe their proof strategies in natural language, and here is what o4-mini had to say about its proof for the pyramid of height 5: *We build the desired disjunction* $(A_{1_1} \vee A_{1_2})$ *by lifting the given OR-premises on the leaf atoms* A_{11} *and* A_{15} *up through each level of implications. At each stage we use nested case analyses (disjunction elimination) on the two OR-facts for that level, together with the appropriate conditional premises, to derive the OR-fact one level higher. Repeating this from* $A_7/A_8/A_9/A_{10}$ *up to* A_2/A_3 *and finally* A_1 *yields* $(A_{1_1} \vee A_{1_2})$. Even when the models get the proof wrong, the high-level strategy is often correct. For instance, even in some cases where o3 tended to avoid hard work by "proving" the conjecture with a single-line application of **FROM** to dozens of premises, it would still articulate a proper strategy.

This separation between strategic competence (e.g., outlining the overall proof structure and identifying relevant lemmas) and formal execution highlights the promise of an agentic approach to theorem proving. In such a framework, LLM-based agents act as planners that explore and propose viable proof strategies, while classical proof assistants or symbolic solvers serve as executors that

enforce rigor at the formal level. Iterative interaction between these components mirrors human mathematical practice, where conceptual insights typically precede fully formalized arguments, and points toward a scalable neurosymbolic paradigm that combines the generative strengths of language models with the reliability of symbolic verification.

6 Conclusions

This paper presented PROOFGRID, a novel benchmark designed to assess the logical reasoning abilities of LLMs and LRMs through tasks that demand rigorous proof construction, checking, and infilling across propositional and equational logic. Unlike prior benchmarks that focus solely on answer accuracy, PROOFGRID emphasizes the integrity of reasoning processes and offers a more accurate assessment of mathematical and logical competence. Our results demonstrate that while state-of-the-art LRMs exhibit very impressive capabilities on simpler inference problems, they continue to struggle with more challenging tasks, especially in reasoning over structurally rich problems.

Nonetheless, instances of high-quality reasoning, especially on problems requiring multi-step abstraction such as DAG pebbling, suggest that these models are beginning to develop powerful inferential strategies. The results on the novel tasks of proof unmasking and gap-filling are particularly impressive. To solve these problems, models must reason about proofs themselves, identifying missing steps, determining the scope of relevant assumptions and lemmas, and reconstructing fully valid derivations. Solving these tasks requires an awareness of proof structure that goes far beyond forward application of inference rules. While the masked proofs are relatively simple, it is nevertheless notable that leading LRMs perform quite well, suggesting that these systems are beginning to exhibit competence in reasoning about reasoning itself. PROOFGRID should provide a robust platform for tracking and accelerating these developments, underscoring the importance of proof-based evaluation as models move toward deeper integration into scientific and mathematical workflows.

The implications of progress on proof-based reasoning extend well beyond benchmarking. The formal methods community has long regarded fully verified proofs of program correctness as the gold standard for software reliability, but such proofs have been prohibitively difficult and resource-intensive, reserved mainly for mission-critical systems. If AI models advance to the point of reliably generating formal proofs alongside code, this could radically transform software development: models could not only produce implementations, but also provide correctness guarantees that can be independently verified by trusted proof checkers. Such a capability would represent a major step towards making LLM-generated software genuinely robust and trustworthy.

7 Limitations

While PROOFGRID offers a comprehensive evaluation of logical reasoning capabilities in LLMs and LRMs, a number of limitations should be acknowledged.

First, our benchmark primarily tests logical reasoning within formal languages, not natural-language reasoning. Although we did translate all PL_1 proofs to English for one set of experiments, this represents a small portion of our evaluation. Thus, PROOFGRID's results may not fully generalize to models' reasoning abilities in natural language contexts.

Second, our benchmark's evaluation metrics focus on accuracy, rather than efficiency. Different LRMs may achieve similar accuracy scores while varying significantly in test-time compute or other resource requirements. A more comprehensive evaluation would include these dimensions.

Finally, we note that our findings apply to current state-of-the-art models evaluated during our study period. Given the rapid advancement in AI capabilities, particularly in reasoning-focused models, the relative performance of various model families may change significantly in the not-too-distant future. Despite these limitations, PROOFGRID represents a significant advancement in the evaluation of logical reasoning in AI systems, offering insights into both the impressive progress and the persistent challenges in this crucial domain.

Acknowledgments

We thank Theo Evgeniou for the fruitful discussions that helped to spark this project and contributed to shaping its initial direction.

Author Contributions

The first author conceived the benchmark, designed the tasks, generated the datasets, and wrote the prompts and the paper. Both authors helped to implement the experimental infrastructure. The

second author oversaw and executed large-scale runs across multiple models, managed the pipeline, and contributed to debugging and validation.

References

Konstantine Arkoudas. 2005. Simplifying proofs in fitch-style natural deduction systems. *Journal of Automated Reasoning*, 34(3):239–294.

Konstantine Arkoudas and David Musser. 2017. *Fundamental Proof Methods in Computer Science*. MIT Press, Cambridge, MA.

Athena Language Team. 2025. Athena: A language for proof engineering. https://athena-lang.org.

Gao Bofei and 1 others. 2024. Omni-MATH: A Universal Olympiad Level Mathematic Benchmark For Large Language Models. *Preprint*, arXiv:2410.07985.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. 2023. TheoremQA: A theorem-driven question answering dataset. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7889–7901, Singapore. Association for Computational Linguistics.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2021. Transformers as soft reasoners over language. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI 2020.

Karl Cobbe, Vineet Kosaraju, and 1 others. 2021. Training verifiers to solve math word problems. In *Advances in Neural Information Processing Systems*.

Bhavana Dalvi, Peter Jansen, Oyvind Tafjord, Zhengnan Xie, Hannah Smith, Leighanna Pipatanangkura, and Peter Clark. 2021. Explaining answers with entailment trees. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7358–7370, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caroline Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, Olli Järviniemi, Matthew Barnett, Robert Sandler, Matej Vrzala, Jaime Sevilla, Qiuyu Ren, Elizabeth Pratt, Lionel Levine, Grant Barkley, and 5 others. 2024. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai. *Preprint*, arXiv:2411.04872.

Google Research. 2025. BIG-Bench Extra Hard. *Preprint*, arXiv:2502.19187.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, Lucy Sun, Alexander Wardle-Solano, Hannah Szabó, Ekaterina Zubova, Matthew Burtell, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, and 16 others. 2024. FOLIO: Natural language reasoning with first-order logic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 22017–22031, Miami, Florida, USA. Association for Computational Linguistics.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 2024. OlympiadBench: A Challenging Benchmark for Promoting AGI with Olympiad-Level Bilingual Multimodal Scientific Problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850. Association for Computational Linguistics.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the MATH dataset. *arXiv preprint arXiv:2103.03874*.

Massimo Lauria. 2021. Cnfgen formula generator. https://github.com/MassimoLauria/cnfgen.

Terufumi Morishita, Gaku Morio, Atsuki Yamaguchi, and Yasuhiro Sogawa. 2023. Learning Deductive Reasoning from Synthetic Corpus based on Formal Logic. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pages 25254–25274. PMLR.

Santiago Ontañón, Joshua Ainslie, Vaclav Cvicek, and Zachary Fisher. 2022. LogicInference: A New Dataset for Teaching Logical Inference to seq2seq Models. In *Proceedings of ICLR 2022 workshop on Objects, Structure and Causality*.

Abulhair Saparov and He He. 2023. Language Models Are Greedy Reasoners: A Systematic Formal Analysis of Chain-of-Thought. In *International Conference on Learning Representations*.

Scale AI and Center for AI Safety. 2025. Humanity's last exam. A benchmark designed to test AI systems at the frontier of human knowledge.

Carsten Sinz. 2005. Towards an optimal CNF encoding of Boolean cardinality constraints. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming*, page 827–831, Berlin, Heidelberg. Springer-Verlag.

Sternberg, Robert, editor. 2020. *The Cambridge Handbook of Intelligence*, 2nd edition. Cambridge University Press.

Haoxiang Sun, Yingqian Min, Zhipeng Chen, Wayne Xin Zhao, Zheng Liu, Zhongyuan Wang, Lei Fang, and Ji-Rong Wen. 2025. Challenging the Boundaries of Reasoning: An Olympiad-Level Math Benchmark for Large Language Models. *Preprint*, arXiv:2503.21380.

Mirac Suzgun and 1 others. 2023. Challenging big-bench tasks and chain-of-thought. In *International Conference on Machine Learning*.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. 2021. ProofWriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3621–3634. Association for Computational Linguistics.

M-A-P Team, Xinrun Du, Yifan Yao, Kaijing Ma, Bingli Wang, Tianyu Zheng, Kang Zhu, Minghao Liu, Yiming Liang, Xiaolong Jin, Zhenlin Wei, Chujie Zheng, Kaixin Deng, Shian Jia, Sichao Jiang, Yiyan Liao, Rui Li, Qinrui Li, Sirun Li, and 77 others. 2025. Supergpqa: Scaling llm evaluation across 285 graduate disciplines. *Preprint*, arXiv:2502.14739.

Jidong Tian, Yitian Li, Wenqing Chen, Liqiang Xiao, Hao He, and Yaohui Jin. 2021. Diagnosing the first-order logical reasoning ability through LogicNLI. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3738–3747. Association for Computational Linguistics.

Yuxuan Wan, Wenxuan Wang, Yiliu Yang, Youliang Yuan, Jen-tse Huang, Pinjia He, Wenxiang Jiao, and Michael Lyu. 2024. LogicAsker: Evaluating and improving the logical reasoning ability of large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 2124–2155, Miami, Florida, USA. Association for Computational Linguistics.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. 2024. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Preprint*, arXiv:2406.01574.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. 2022. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *Preprint*, arXiv:2109.00110.

Yujun Zhou, Jiayi Ye, Zipeng Ling, Yufei Han, Yue Huang, Haomin Zhuang, Zhenwen Liang, Kehan Guo, Taicheng Guo, Xiangqi Wang, and Xiangliang Zhang. 2025. Dissecting logical reasoning in llms: A fine-grained evaluation and supervision study. *Preprint*, arXiv:2506.04810.

A NDL Specification

As mentioned in Section 3, NDL can be viewed as a drastically stripped-down subset of Athena. Its semantics are specified in terms of *assumption bases*. An assumption base is simply a finite set of formulas β. In particular, after specifying the abstract syntax of proofs D, as we do below, the denotational semantics of any proof D can be specified as a function $\mathcal{I}$ that takes D and an assumption base β and produces either the *conclusion* obtained by evaluating D in the context of β or an error. Thus, the denotational semantics of NDL proofs coincide with the expected behavior of proof checking.

In the case of zero-order (propositional) logic, a deduction (proof) D in NDL is either:

- An application of an inference rule R to $n \geq 0$ arguments: R **`on`** $p_1, \ldots, p_n$. Most of the inference rules in NDL are either introduction or elimination rules, but there are also some convenient *derived* rules (meaning rules that can be derived from the introduction and elimination rules), such as De Morgan and disjunctive syllogism. A complete list can be found online in the PROOFGRID prompts.

- A *conditional proof* of the form **`assume`** p D. Here the formula p is a *hypothesis* and the subproof D is the *body* of the conditional proof. These proofs are used to produce conditional conclusions of the form $(p \Rightarrow q)$.

- A *proof sequence* $\{D_1; D_2; \cdots ; D_n\}$. This is a lemma formation mechanism: first we evaluate proof D_1 and obtain its conclusion p_1, which now becomes available as a lemma for the subsequent proofs; we then go on to D_2 and so on. The final conclusion is that obtained by D_n.

As mentioned above, to *evaluate* a proof D in NDL in some assumption base β means to check D. Evaluation produces either the conclusion of D or some error which indicates that D is incorrect. Below we sketch this evaluation algorithm via structural recursion on the input proof D.

Primitive proofs: If D is a primitive proof

$$R \text{ } \texttt{on} \text{ } p_1, \ldots, p_n,$$

we first check to see if $p_1, \ldots, p_n$ are in β and then proceed by a case analysis of R. For instance, if R is `both` (conjunction introduction) and $n = 2$, and p_1, p_2 are both in β, we produce the conclusion $(p_1 \wedge p_2)$. It's an error if a rule is given a different number or type of arguments than what it expects. For instance, if `both` is applied to only one argument p_1, the evaluation algorithm will produce the result *error*.[3] Further, with the exception of the rules `ex-middle`, `left-either`, `right-either`, `from-false`, and `by-contradiction`, all of the arguments given to a rule R must be in the current assumption base, β; it is a logical error if that is not the case. For the binary rule `left-either`, only the first argument needs to be in β, while for `right-either` and `by-contradiction` only the second argument needs to be in β.

Conditional Proofs: To evaluate a conditional proof **`assume`** p D in an assumption base β, we evaluate the body D in the augmented assumption base $\beta \cup \{p\}$. If and when that produces a conclusion q, we return the result $(p \Rightarrow q)$, otherwise we return *error*.

Proof Sequences: To evaluate $\{D_1; D_2; \cdots ; D_n\}$ in β, we start by evaluating D_1 in β. If that gives *error*, we also return *error*. Otherwise we return the result of recursively evaluating $\{D_2; \cdots ; D_n\}$ in $\beta \cup \{p_1\}$.

The conclusion of a subproof D can be given a name I and then subsequent proofs can refer to that conclusion by the name I, and likewise for hypotheses, e.g.:

```
assume hyp := (A <==> B) {
   p1 := left-iff on hyp;
   p2 := right-iff on hyp;
   both on p1, p2
}

# Evaluating (checking) this proof will produce the
# conclusion ((A <==> B) ==> ((A ==> B) & (B ==> A))
    ).
```

The **`BY`** keyword is also frequently used to broadcast the conclusion expected by a rule application: p **`BY`** R **`on`** $p_1, \ldots, p_n$. If the application of R to $p_1, \ldots, p_n$ (in some assumption base β) does produce the conclusion p, then the entire step succeeds and its result becomes p. But if the application of

[3] To keep the exposition simple, we assume that there is a single distinguished token *error* that represents all evaluation failures. In practice, of course, the reported error messages vary in accordance with the context.

R either results in *error* or in a conclusion q other than the advertised p, then the result of the entire `BY` step becomes *error*.

B Additional Information on PL_1 and PL_2

B.1 PL_1

For PL_1, premises and goals for all problems were generated randomly as ASTs (abstract syntax trees) by choosing a logical operator as the root (or an atom for a leaf) and then recursively generating subtrees up to a maximum depth. To minimize degenerate problems, we imposed the following constraints on any premises $p_1, \ldots, p_n$ and goal p:

1. all $n+1$ formulas must be distinct;
2. $\{p_1, \ldots, p_n\}$ must logically imply p;
3. every premise p_i must be necessary, meaning that if we remove p_i, the goal p is no longer entailed by the remaining premises;
4. the goal does not contain any atoms that do not occur in the premises; and
5. the premises by themselves are logically consistent (as is the negation of the goal).

We used a SAT solver to enforce 2, 3, and 5.

We limit duplicate entries by normalizing a problem's representation as follows: sort the list of all premises, conjoin the resulting elements, form the conditional between that conjunction and the conclusion, and then universally quantify over all atoms in that conditional. We then identify two problems iff these universally quantified normal forms are alpha-equivalent (identical up to renaming of bound variables). Thus, for instance, the argument with $p_1 = (A \Rightarrow B)$, $p_2 = (B \Rightarrow C)$, $p_3 = (\sim C)$, and conclusion $(\sim A)$ is represented by the QBF (Quantified Boolean Formula):

$$\forall A, B, C \,.\, (((A \Rightarrow B) \,\&\, (B \Rightarrow C) \,\&\, (\sim C)) \Rightarrow (\sim A))$$

and is thus considered identical to the problem with $p_1 = (C \Rightarrow D)$, $p_2 = (\sim E)$, $p_3 = (D \Rightarrow E)$, and conclusion $(\sim C)$.

The Athena theorem prover that generated most of the proofs in the dataset uses a combination of backward and forward heuristics, ordered according to the strategy in Section 4.1.14 of the textbook *Fundamental Proof Methods in Computer Science* (Arkoudas and Musser 2017). The theorem prover is written as an Athena *method*, which is automatically instrumented so that a successful run produces a low-level *certificate*: a fully detailed proof comprising all applications of primitive inference rules that were made during the run, as well as all conditional (sub)proofs and (sub)proofs by contradiction, properly scoped. This certificate is typically a very long proof (many thousands of lines), because it includes all inferences made even during paths of the search tree that were ultimately dead ends. These proofs are then simplified by an aggressive proof-optimization algorithm (Arkoudas 2005) that removes detours and other inefficiencies from formal proofs, radically simplifying their structure.

The deliberate corruption of a proportion of these proofs used interventions such as altering the logical form of an argument to an inference rule (e.g., applying conjunction elimination to a disjunction), or the number of arguments, removing a step from a chain of subproofs, and so on. As mentioned in the body of the paper, we also included in the dataset proofs written by models (along with relevant provenance data).

B.1.1 Proof Masking

As an example of this task, below is a masked version of the proof given in Section 3.

```
assert premise-1 := (A ==> B)
assert premise-2 := (~ A ==> C)
assert premise-3 := (C ==> D)
# Goal: (B | D)
{
  (A | ~ A) BY ex-middle on MASK1;
  assume A {
    MASK2 BY MASK3 on premise-1, MASK4;
    (MASK5 | D) by left-either on MASK6, MASK7;
  };
  assume MASK8 {
    MASK9 BY mp on premise-2, MASK10;
    D BY mp on MASK11, MASK12;
    MASK13 BY right-either on B, D;
  };
  MASK14 BY cases on (A | ~ A), MASK15, MASK16
}
```

Having seen the original proof, the correct assignment is easy to decipher: `MASK1`:`A`; `MASK2`:`B`; `MASK3`:`mp`; `MASK4`:`A`; and so on. But without the benefit of having seen the proof before, this is not an easy task. We have to infer that `MASK1` must be `A` from the specification of the excluded-middle rule and the fact that we see the conclusion `(A | ~ A)` to the left of `BY`. On the last line, because the goal is `(B | ~ D)` and the pivot disjunction is `(A | ~ A)`, we can infer from the specification of case analysis that `MASK15` and `MASK16` must be conditionals of the form `(A ==> (B | D))` and `(~ A ==> (B | D))`, respectively. When a proof is heavily obscured, the corresponding problem generally becomes more

complex. In our dataset, we randomly masked anywhere from 30% to 90% of all candidates (of all 4 types listed above) in a given proof.

Masking problems are usually underdetermined, i.e., there may be multiple mask assignments that yield a correct proof. Accordingly, it is not expected that a mask assignment produced by a model recovers the original proof. For instance, the original proof might contain a chain of two consecutive applications of left and right conjunction elimination: `··· left-and on (A & B); right-and on (A & B); ···`. But if we mask both rules, e.g., by replacing `left-and` by `MASK-3` and `right-and` by `MASK-4`, then `MASK3: left-and; MASK4: right-and` and `MASK3: right-and; MASK4: left-and` are both valid solutions, because the two inference steps can be swapped without affecting the correctness of the subsequent reasoning.

B.1.2 Proof Gap Filling

The following is a simple example of a gap-filling problem:

```
1  assert premise-1 := (C | (A & E))
2  assert premise-2 := (C ==> D)
3  assert premise-3 := (A <==> (E ==> D))
4  # Goal: (E ==> D)
5
6  {
7    left-iff on premise-3;
8    GAP-1;
9    assume E {
10     case1 := assume C {
11                mp on premise-2, C
12              };
13     case2 := assume (A & E) {
14                left-and on (A & E);
15                mp on (A ==> (E ==> D)), A;
16                GAP-2
17              };
18     GAP-3
19   }
20 }
```

Bearing the goal `(A ==> D)` in mind, it is clear from the structure of the overall proof skeleton, and particularly from the structure of the top-level conditional proof (the **assume** that opens on line 9 and closes on line 19), that this conditional proof establishes `(E ==> D)`. The body proceeds by a case analysis of `premise-1`, the disjunction `(C | (A & E))`. The `case1` subproof establishes that `C` implies `D`, thus `case2` must establish that `(A & E)` implies `D`. Therefore, `GAP-2` must derive `D` from the assumption base at the end of line 15, which includes the conditional `(E ==> D)` (derived on line 15) and the `case-2` hypothesis `(A & E)` (as `GAP-2` is inside the scope of that hypothesis). So this gap is easy to fill: First detach `E` from the assumption `(A & E)` and then use modus ponens on `(E ==> D)` and `E` to infer `D`. `GAP-3` is a simple application of `cases` to `premise-1`, `(C ==> D)` and `((A & E) ==> D)`.

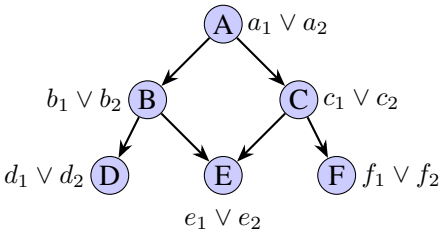
`GAP-1` does not make any contribution to the proof. It is there only because the LLM that generated the initial proof included a redundant step at that point—the application of `right-iff` on `premise-3`. Any replacement for that gap is given full credit as long as the entire proof obtained by filling the gaps as specified by the model correctly derives the goal from the premises.

B.2 PL_2

In this section we describe the problem families in PL_2.

B.2.1 DAG Pebbling Problems

Consider a DAG pyramid, e.g., of height 2, as shown below. Nodes A–E represent tasks. Each task can be completed in two ways: a_1 and a_2 for A, b_1 and b_2 for B, and so on. We indicate this by attaching these disjunctions to every node. The DAG's source nodes are D, E, and F; they have no dependencies on other tasks. Arrows indicate task dependencies: B depends on D and E, C depends on E and F (cannot be completed before they are), and so on. A is the *target node*, the task that can only be completed (or *pebbled*) last.



A pebbling problem on a DAG pyramid of this sort is defined by 3 classes of formulas. The *source formulas* assert that every source node can be pebbled, i.e., we have $d_1 \vee d_2$, $e_1 \vee e_2$, and $f_1 \vee f_2$. The *precedence formulas* encode dependencies: For every non-source node, we assert 4 conditional precedence formulas encoding 4 distinct completion possibilities. For instance, the 4 precedence conditionals for B are: $d_i \wedge e_j \Rightarrow b_1 \vee b_2$, for all $i, j \in \{1, 2\}$. This says that B can be pebbled if either D has been pebbled by d_1 and E by e_1, or D by d_1 and E by e_2, and so on. (Precedence formulas are typically represented by clauses but we find conditionals more intuitive.) The last class of formulas consists of the negations of all target disjunctions, in our case simply $\neg(a_1 \vee a_2)$ since we only have 1 target (A). The task now is to show that

the set of all formulas is unsatisfiable. However, in our forward-inference formulation we simply prove that target A must be pebbled: $a_1 \vee a_2$. This follows logically from the source and precedence formulas.

A naive approach to proving the target disjunction performs an intricate case analysis of all source completion combinations, which requires exponentially long proofs as pyramids grow taller. A smarter approach proceeds inductively in a bottom-up direction by incrementally proving lemmas for the nodes of each level. The source nodes provide the induction basis, and the precedence formulas act as inductive hypotheses of sorts. Continuing this way finally derives the target disjunction. We include 10 problems for DAG pyramids up to height 10, and our ground-truth NDL proofs are written programmatically (using the outlined algorithm).

The 50 "simple" DAG pebbling problems don't have disjunctions associated with nodes and the DAGs are not pyramids. Rather, they have long linear dependency chains and can thus gauge how well a system can reason about long inference chains captured by Horn clauses (conditionals with atomic conclusions and conjunctive antecedents), either in a forward or in backwards-chaining fashion.

B.2.2 Graph Coloring Problems

For the 70 graph-coloring problems we restrict attention to 3 colors. We use the `networkx` library to construct random undirected graphs with relatively few nodes (typically ≤ 10) and edge probability randomly chosen from the uniform distribution [0.6,0.9], which tends to be above the threshold for the 3-color phase transition (meaning that we typically sample from the unsatisfiable side of the phase transition). Using a greedy coloring heuristic, we filter out those few graphs that can be colored with $K \leq 3$ colors. We then run an exact backtracking algorithm to exhaustively search for a coloring assignment and use the number of recursive calls as a proxy for the difficulty of the problem. We finally retain only those problems that satisfy the additional constraint that their unsatisfiability (showing it's impossible to color the graph with 3 colors) can be proved with < 150 resolution steps.

B.2.3 Relativized Pigeonhole Principle Problems

We have m pigeons, t *resting places*, and n pigeonholes. We introduce two families of Boolean variables:

$$p_{i,k} \quad (1 \leq i \leq m,\ 1 \leq k \leq t) \quad \text{and}$$
$$q_{k,j} \quad (1 \leq k \leq t,\ 1 \leq j \leq n).$$

Informally,

- $p_{i,k}$ means "pigeon i comes to rest at place k."
- $q_{k,j}$ means "the (unique) pigeon at resting place k eventually flies into hole j."

We then conjoin formulas enforcing the following constraints:

1. *Coverage of pigeons by resting places:*
$$\big(p_{i,1} \vee p_{i,2} \vee \cdots \vee p_{i,t}\big) \quad (i = 1, \ldots, m),$$
so every pigeon chooses some resting place.

2. *Mutual exclusion at resting places:*
$$p_{i,k} \Rightarrow \neg p_{j,k} \quad \forall\, 1 \leq i < j \leq m,\ 1 \leq k \leq t,$$
so no two pigeons share the same resting place.

3. *Coverage of occupied places by holes:*
$$p_{i,k} \Rightarrow \big(q_{k,1} \vee q_{k,2} \vee \cdots \vee q_{k,n}\big)$$
$$\forall\, 1 \leq i \leq m,\ 1 \leq k \leq t,$$
so if a pigeon rests at k, that place is assigned to some hole.

4. *Mutual exclusion at holes:*
$$q_{k,j} \Rightarrow \neg q_{k',j}$$
$$\forall\, 1 \leq k < k' \leq t,\ 1 \leq j \leq n,$$
so no hole receives more than one pigeon.

A simple combinatorial argument shows that the conjunction of these clauses is satisfiable *if and only if*

$$m \ \leq \ t \ \leq \ n.$$

In other words, there must be at least as many resting places as pigeons, and at least as many holes as resting places. Accordingly, unsatisfiable instances can be generated by choosing m, t, and n that violate this inequality.

As an example, take $m = 2$ pigeons, $t = 2$ resting places, and $n = 1$ hole. Then

$$\{p_{1,1}, p_{1,2}, p_{2,1}, p_{2,2},\ q_{1,1}, q_{2,1}\}$$

are our variables, and the above four sets of constraints become

$$p_{1,1} \vee p_{1,2}, \quad p_{2,1} \vee p_{2,2},$$
$$p_{1,k} \Rightarrow \neg p_{2,k} \quad (k = 1, 2),$$
$$p_{i,k} \Rightarrow q_{k,1} \quad (i = 1, 2,\ k = 1, 2),$$
$$q_{1,1} \Rightarrow \neg q_{2,1}.$$

Here $m = 2 \leq t = 2$ but $t = 2 > n = 1$, so these formulas are unsatisfiable, as there is no way to route two pigeons—via two resting places—into a single hole.

B.2.4 Subset Cardinality Problems

Let $B = (L, R, E)$ be a bipartite graph with left part L, right part R, and edge set $E \subseteq L \times R$, where the vertices in L and R are represented as integers. We introduce one Boolean variable $x_{i,j}$ for each edge $(i, j) \in E$, where $x_{i,j}$ means we "select" the edge (i, j). For any vertex $v \in L \cup R$, we write $N(v)$ for the set of v's neighbors:

$$N(v) \;=\; \{\, u : (u, v) \in E \text{ or } (v, u) \in E\}$$

and we define $d(v) = |N(v)|$. We build formulas enforcing two sets of cardinality constraints:

- **Left-side constraint.** For each $i \in L$, at least half of its incident edges must be selected:

$$\sum_{j \in N(i)} x_{i,j} \;\geq\; \left\lceil \tfrac{d(i)}{2} \right\rceil.$$

- **Right-side constraint.** For each $j \in R$, at most half of its incident edges may be selected:

$$\sum_{i \in N(j)} x_{i,j} \;\leq\; \left\lfloor \tfrac{d(j)}{2} \right\rfloor.$$

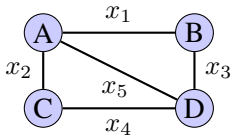
Each of these inequalities (or equalities) is then translated into conjunctive normal form by a standard cardinality-constraint encoding, e.g., a sorting-network encoding or a sequential counter (Sinz 2005). We used the `cnfgen` library (Massimo Lauria 2021) to generate 70 unsatisfiable formulas of this type in DIMACS format, and we then reversed the CNF transformation in most cases and converted the formulas into the uniform NDL/Athena notation used throughout PROOFGRID.

B.2.5 De Bruijn Formula Problems

For any integer $n > 0$, the n^{th} De Bruijn formula, which we denote by $\mathcal{D}_n$, is defined as the conditional $p \Rightarrow q$, where q is the conjunction $A_1 \wedge \cdots \wedge A_n$ and p is the conjunction of the n conditionals $(A_1 \Leftrightarrow A_2) \Rightarrow q$, $(A_2 \Leftrightarrow A_3) \Rightarrow q$, …, $(A_n \Leftrightarrow A_1) \Rightarrow q$. It is easy to show that $\mathcal{D}_n$ is a tautology (and hence provable) iff n is odd. The formula does *not* hold for any even n. We automatically generated NDL proofs for the first 15 odd-indexed instances of $\mathcal{D}_n$. These proofs grow quite large with increasing n. They are a hard test for proof checking not just due to their sheer length, but also because they have multiple nested assumptions that require very careful and precise tracking of assumption scope.

B.2.6 Tseitin Formula Problems

Consider the following undirected graph G, where each node (A, B, C, D) is also viewed as a propositional atom, and where additional propositional atoms x_i are attached to the edges:



The core Tseitin formula for G is the conjunction of the following *definitions*, one for each node: $A \Leftrightarrow x_1 \oplus x_2 \oplus x_5$, $B \Leftrightarrow x_1 \oplus x_3$, $C \Leftrightarrow x_2 \oplus x_4$, $D \Leftrightarrow x_3 \oplus x_4 \oplus x_5$, where $\oplus$ denotes the exclusive or: $x_i \oplus x_j \Leftrightarrow (x_i \wedge \sim x_j) \vee (x_j \wedge \sim x_i)$. An easy induction on n, with $n = 2$ as the base case, shows that $p_1 \oplus p_2 \oplus \cdots \oplus p_n$ holds iff exactly an odd number of p_i hold (we omit parentheses since $\oplus$ is commutative and associative). Thus, these definitions assert that each node atom holds iff an odd number of edges incident to that node hold. So far all we have is a set of definitions, one for each node, that are perfectly jointly consistent. But now suppose we go on to assert that an odd number k of node atoms hold, say by specifying A, $\sim B$, $\sim C$, $\sim D$ (so that only $k = 1$ atom holds, A in this case). By what we just said about the exclusive or, it would follow that $A \oplus B \oplus C \oplus D$ holds. However, by the given definitions, we have:

$$\begin{aligned} A \oplus B \oplus C \oplus D \quad &\Leftrightarrow \quad (x_1 \oplus x_2 \oplus x_5) \oplus (x_1 \oplus x_3) \\ &\qquad \oplus (x_2 \oplus x_4) \oplus (x_3 \oplus x_4 \oplus x_5) \\ &\Leftrightarrow \quad (x_1 \oplus x_1) \oplus (x_2 \oplus x_2) \\ &\qquad \oplus (x_3 \oplus x_3) \\ &\qquad \oplus (x_4 \oplus x_4) \oplus (x_5 \oplus x_5) \\ &\Leftrightarrow \quad \texttt{false}. \end{aligned}$$

The first step follows just by rearranging the variables on the right-hand side of the first equivalence, while the second step follows because $p \oplus p \Leftrightarrow \texttt{false}$ for any formula p. Thus we have a contradiction: $A \oplus B \oplus C \oplus D$ holds because we have stipulated that only A holds, but at the same time we have shown that $A \oplus B \oplus C \oplus D$ cannot hold because it leads to `false`. The key intuition is that because each edge variable appears twice on the right-hand side of the first equivalence (once for each node to which the edge is incident), we can regroup all variables as shown above to derive

`false`.

While the conventional formulation of a Tseitin problem is as a jointly unsatisfiable set of formulas (typically clauses, but again, we work with full propositional logic), we can easily pose it as a forward inference problem: Given the n definitions for the n node atoms of a graph G, prove the negation of any conjunction of an odd number of those atoms.

B.2.7 Counting Principle Problems

The 5 problems in this group encode statements of the form *There is a way to partition a set of M elements into subsets of size p each*. Clearly, this statement is satisfiable iff p divides M. Set $n = M/p$. For each element $e \in \{1, 2, \ldots, M\}$ and each part (or "block") $b \in \{1, 2, \ldots, n\}$, introduce a Boolean variable $x_{e,b}$ that holds iff element e is assigned to block b. We conjoin the following sets of constraints:

1. *Exactly one block per element.* Every element e must appear in exactly one block:

$$\big(x_{e,1} \vee x_{e,2} \vee \cdots \vee x_{e,n}\big) \text{ and } x_{e,b_1} \Rightarrow \neg x_{e,b_2} \quad \forall\ b_1 \neq b_2, \quad e = 1, \ldots, M.$$

2. *Exactly p elements per block.* Each block b must contain exactly p elements:

$$\sum_{e=1}^{M} x_{e,b} \;=\; p, \quad b = 1, \ldots, n.$$

In SAT, such an equality is typically encoded by a pair of cardinality constraints $\geq p$ and $\leq p$, each represented by a standard cadinality encoding.

Simple counting shows that these constraints admit a satisfying assignment precisely when p divides M.

C Equational Proofs

In the example term $g(a, f(X))$ mentioned in Section 3, f and g have arities 1 and 2, respectively, and a is a constant (symbol of arity 0). Variables start with upper-case letters; unary function symbols are $f, f_1, \ldots$; binary symbols are $g, g_1, g_2, \ldots$; ternary symbols are $h, h_1, \ldots$; and quaternary function symbols (arity 4) are $r, r1, \ldots$. The proof is a sequence of rewrite steps $s_1 = s_2 = \cdots = s_n$ starting from the original term $s = s_1$ and leading to $s_n = t$. Each step $s_i = s_{i+1}$ is justified by citing one or more of the given axioms.

The following is a simple example of a set of (unconditional) axioms and an equational proof based on these axioms:

```
- Axioms:
E1: h(c,V1170,c) = h(V1170,a,f4(f1(c)))
E2: h(V1173,V1174,V1173) = g2(V1173,V1174)
E3: h(a,V1181,f4(f1(c))) = g1(f2(f4(V1181)),f4(e))
E4: f2(f4(V1184)) = g(f3(V1184),f2(e))
E5: g(V1187,f2(e)) = g4(f1(f5(V1187)),d)
E6: f4(V1188) = V1188

- Proof:

s = g(h(c,a,c),h(a,c,a))
    g(h(a,a,f4(f1(c))),h(a,c,a))          by E1
    g(h(a,a,f4(f1(c))),g2(a,c))           by E2
    g(g1(f2(f4(a)),f4(e)),g2(a,c))        by E3
    g(g1(g(f3(a),f2(e)),f4(e)),g2(a,c))   by E4
    g(g1(g4(f1(f5(f3(a))),d),e),g2(a,c))  by E5, E6
```

This is a rather short proof. The average number of steps in an EQ_1 proof is 18; the maximum is 35.

What does it mean to say that such a proof is correct? It means that for every proof step $s_i = s_{i+1}$ **by** $E_{i_1}, \ldots, E_{i_k}$, the term s_i has $k > 0$ subterms $u_1, \ldots, u_k$ called *redexes*, in mutually disjoint positions, where each redex u_i *matches* the left-hand side of exactly one of the k listed equations, call it E_{u_i}, under some *substitution* θ_i. A substitution θ is a function from variables to terms; we say that a term s' matches a term s under θ iff $\widehat{\theta}(s) = s'$, where $\widehat{\theta}$ is the unique homomorphic extension of θ to the set of all terms. The term s_{i+1} is then obtained from s_i by replacing every redex u_i with the corresponding right-hand side of E_{u_i}, called the *contractum*, i.e., by applying the substitution θ_i to the right-hand side of E_{u_i}. For example, if E_1 is $f(X) = g(X, f(X))$, then $h(a, f(b), c) = h(a, g(b, f(b)), c)$ by E_1, because the subterm $f(b)$ matches the left-hand side of E_1 under $\theta = \{X \mapsto b\}$ and we can obtain $h(a, g(b, f(b)), c)$ by replacing the redex $f(b)$ with the corresponding right-hand side of E_1—the contractum $g(b, f(b))$.

All problems were randomly generated, first by generating a chain of terms $s_1, \ldots, s_n$ where each s_{i+1} is a minor perturbation of s_i and then inducing a set of universally quantified equations enabling the rewriting of s_i into s_{i+1}. The proofs in this dataset are tedious and involve fairly large terms but are routine and do not require much ingenuity.

For the equational proof-checking task, each dictionary (record) in EQ_1 comprises the following items:

- `start` and `end` represent the starting and destination terms;
- `equationalAxioms` is a dictionary mapping the names of equational axioms to the axioms;

- `numberOfProofSteps` gives the number of steps in the proof;
- `correctProof` is boolean signifying whether the proof is actually correct;
- `incorrectStep` is the position of the first step in the proof that contains an error, if such a step exists, and -1 otherwise;
- `proof` is a list of *steps*, where every step is a structured record (dictionary) with the following keys:
  - `step`: the position of the step (counting starts with 0);
  - `term`: the term corresponding to that step;
  - `redexList`: the list of redexes that rewrite this step's term into the term of the next step; each redex in `redexList` is a dictionary with the following keys:
    * `equation` gives the matching equation for that particular redex;
    * `equationName` is the unique name of that equation;
    * `redex` gives the term representation of the redex;
    * `redexPosition` gives the unique position of the redex inside the overall term as a list of positive integers representing the Dewey decimal position of the redex;
    * `theta` gives the matching substitution.

Below is a full example of the above representation of an equational proof:

```
- start: g(r(c,e,d,c),r(b,e,c,e))
  end: g(g3(f1(f4(b)),d),f2(f5(g1(e,b))))
  equationalAxioms:
    E1: f4(V186772) = V186772
    E2: g4(V186784,V186785) = f3(V186784)
    E3: g4(V186773,V186774) = g2(f4(f3(V186773)),
   V186774)
    E4: h5(V186755,c,e) = f1(f1(g1(V186755,e)))
    E5: f(V186794) = g3(V186794,d)
    E6: g2(f4(f3(V186783)),b) = f(g3(V186783,a))
    E7: f1(f1(V186767)) = f4(g4(V186767,f4(e)))
    E8: r(c,e,V186752,c) = g4(f3(V186752),b)
    E9: r(V186733,e,V186734,V186735) = h5(V186733,
   V186734,V186735)
    E10: f3(V186766) = f4(V186766)
    E11: f3(V186793) = f2(f5(V186793))
    E12: g1(V186758,V186759) = g1(V186759,V186758)
    E13: g3(V186799,V186800) = f1(V186799)
  numberOfProofSteps: 9
  correctProof: false
  incorrectStep: 9
  proof:
    - step: 0
      term: g(r(c,e,d,c),r(b,e,c,e))
      redexList: []
    - step: 1
      term: g(r(c,e,d,c),h5(b,c,e))
      redexList:
        - equation: r(V186733,e,V186734,V186735) =
   h5(V186733,V186734,V186735)
          equationName: E9
          redex: r(b,e,c,e)
          redexPosition:
            - 2
          theta:
            V186735: e
            V186734: c
            V186733: b
          contractum: h5(b,c,e)
    - step: 2
      term: g(g4(f3(d),b),h5(b,c,e))
      redexList:
        - equation: r(c,e,V186752,c) = g4(f3(
  V186752),b)
          equationName: E8
          redex: r(c,e,d,c)
          redexPosition:
            - 1
          theta:
            V186752: d
          contractum: g4(f3(d),b)
    - step: 3
      term: g(g4(f3(d),b),f1(f1(g1(b,e))))
      redexList:
        - equation: h5(V186755,c,e) = f1(f1(g1(
  V186755,e)))
          equationName: E4
          redex: h5(b,c,e)
          redexPosition:
            - 2
          theta:
            V186755: b
          contractum: f1(f1(g1(b,e)))
    - step: 4
      term: g(g4(f3(d),b),f1(f1(g1(e,b))))
      redexList:
        - equation: g1(V186758,V186759) = g1(
  V186759,V186758)
          equationName: E12
          redex: g1(b,e)
          redexPosition:
            - 2
            - 1
            - 1
          theta:
            V186759: e
            V186758: b
          contractum: g1(e,b)
    - step: 5
      term: g(g4(f4(d),b),f4(g4(g1(e,b),f4(e))))
      redexList:
        - equation: f3(V186766) = f4(V186766)
          equationName: E10
          redex: f3(d)
          redexPosition:
            - 1
            - 1
          theta:
            V186766: d
          contractum: f4(d)
        - equation: f1(f1(V186767)) = f4(g4(
  V186767,f4(e)))
          equationName: E7
          redex: f1(f1(g1(e,b)))
          redexPosition:
            - 2
          theta:
            V186767: g1(e,b)
          contractum: f4(g4(g1(e,b),f4(e)))
    - step: 6
      term: g(g2(f4(f3(f4(d))),b),g4(g1(e,b),f4(e)
  ))
      redexList:
        - equation: f4(V186772) = V186772
          equationName: E1
          redex: f4(g4(g1(e,b),f4(e)))
          redexPosition:
            - 2
          theta:
            V186772: g4(g1(e,b),f4(e))
          contractum: g4(g1(e,b),f4(e))
        - equation: g4(V186773,V186774) = g2(f4(f3
```

```
(V186773)),V186774)
        equationName: E3
        redex: g4(f4(d),b)
        redexPosition:
          - 1
        theta:
          V186774: b
          V186773: f4(d)
        contractum: g2(f4(f3(f4(d))),b)
    - step: 7
      term: g(f(g3(f4(d),a)),f3(g1(e,b)))
      redexList:
        - equation: g2(f4(f3(V186783)),b) = f(g3(
V186783,a))
        equationName: E6
        redex: g2(f4(f3(f4(d))),b)
        redexPosition:
          - 1
        theta:
          V186783: f4(d)
        contractum: f(g3(f4(d),a))
        - equation: g4(V186784,V186785) = f3(
V186784)
        equationName: E2
        redex: g4(g1(e,b),f4(e))
        redexPosition:
          - 2
        theta:
          V186785: f4(e)
          V186784: g1(e,b)
        contractum: f3(g1(e,b))
    - step: 8
      term: g(g3(g3(f4(d),a),d),f2(f5(g1(e,b))))
      redexList:
        - equation: f3(V186793) = f2(f5(V186793))
          equationName: E11
          redex: f3(g1(e,b))
          redexPosition:
            - 2
          theta:
            V186793: g1(e,b)
          contractum: f2(f5(g1(e,b)))
        - equation: f(V186794) = g3(V186794,d)
          equationName: E5
          redex: f(g3(f4(d),a))
          redexPosition:
            - 1
          theta:
            V186794: g3(f4(d),a)
          contractum: g3(g3(f4(d),a),d)
    - step: 9
      term: g(g3(f1(f4(b)),d),f2(f5(g1(e,b))))
      redexList:
        - equation: g3(V186799,V186800) = f1(
V186799)
        equationName: E13
        redex: g3(f4(d),a)
        redexPosition:
          - 1
          - 1
        theta:
          V186800: a
          V186799: f4(d)
        contractum: f1(f4(b))
        corrupted: true
        correctContractum: f1(f4(d))
        corruptionMethod: contractumCorruption
      corruptedRedex: 1
```


For the proof-checking task, as mentioned in Section 3, if a model judges a proof to be correct then it must output an explanation of that correctness for *each* step of the proof. An explanation for a step $s_i = s_{i+1}$ must list every redex for that step, along with the position of the redex and the corresponding equation and contractum. The position of a redex is encoded as a list of positive integers representing the Dewey decimal position of the redex in the term. If the model asserts that a proof is incorrect, it must specify the first step where the proof goes wrong and how exactly it does so. A proof is correctly checked iff the model correctly evaluates every step in the proof. The metric we report in Section 4 is accuracy—the fraction of proofs that are correctly checked.