ProofNet: A Benchmark for Autoformalizing and Formally Proving Undergraduate-Level Mathematics Problems

Zhangir Azerbayev
Yale University*
`zhangir.azerbayev@yale.edu`

Bartosz Piotrowski
University of Warsaw*
`bartoszpiotrowski@post.pl`

Jeremy Avigad
Carnegie Mellon University
`avigad@cmu.edu`

Abstract

We introduce ProofNet, a benchmark for autoformalization and formal proving of undergraduate-level mathematics. The ProofNet benchmarks consists of 297 theorem statements expressed in both natural language and the Lean 3 theorem prover, 100 of which are also accompanied by natural language proofs. The problems are primarily drawn from popular undergraduate pure mathematics textbooks, and cover topics such as real and complex analysis, linear algebra, abstract algebra, and topology. We intend for ProofNet to be a challenging benchmark that will drive progress in autoformalization and automatic theorem proving. We report baseline results on the autoformalization of statements using few-shot learning with large language models.

1 Introduction

The creation of an automatic mathematician, that is, a system capable of autonomously posing conjectures and proving theorems, is a longstanding challenge in mathematics and artificial intelligence [Gelernter, 1959]. In recent years, generative neural language modelling has emerged as a promising approach to automating aspects of mathematics [Rabe and Szegedy, 2021].

One approach to applying language models to mathematics has been to treat mathematical reasoning in natural language as a sequence learning task [Welleck et al., 2021, 2022, Lewkowycz et al., 2022]. A key advantage of mathematical reasoning in natural language is the abundance of natural language mathematics data on the internet [Lewkowycz et al., 2022].

An alternative approach has been to use language models to guide formal proof-search in an interactive theorem prover (ITP) [Polu and Sutskever, 2020, Polu et al., 2022, Jiang et al., 2022]. A salient advantage of this method is that the ITP acts as a verifier for the language model's reasoning, enabling the natural implementation of bootstrapping techniques such as expert iteration [Silver et al., 2017, Polu et al., 2022].

Autoformalization seeks to build a bridge between informal and formal mathematical reasoning [Wang et al., 2018, Szegedy, 2020, Wu et al., 2022a], with the potential to one day be able to leverage vast corpora of natural language mathematics data while still grounding a system's reasoning in formal logic. However, lack of large parallel corpora of informal and formal mathematics means that autoformalization suffers from a lack of standard benchmarks to guide progress in the field.

To remedy this gap, we propose ProofNet,[2] a benchmark consisting of parallel natural language and formal mathematics that can be used to evaluate autoformalization and theorem proving. The ProofNet benchmark consists of 297 parallel natural language and formal theorem statements, 100 of

*Work completed while at Carnegie Mellon University
[2]Code and full dataset are available at `https://github.com/zhangir-azerbayev/ProofNet/`.

which are also accompanied by informal proofs, sourced from the exercises of popular undergraduate-level pure mathematics textbooks. Formal statements are expressed in the Lean 3 theorem prover [de Moura et al., 2015], and depend on Lean's mathlib [mathlib Community, 2020].

Moreover, language-model-based theorem provers and autoformalization systems have typically been evaluated on benchmarks consisting of competition and Olympiad-style problems [Zheng et al., 2022, Wu et al., 2022a]. While such problems require complex reasoning, their solutions only depend on a relatively small set of elementary facts about integers, real numbers, counting, and geometry. In contrast, modern research mathematics requires the mastery of a massive body of theory made up of thousands of definitions, lemmas, and theorems. The Lean 3 formalization of Perfectoid Spaces, an important definition in research-level arithmetic geometry, depends on over 3000 distinct theorems and definitions [Buzzard et al., 2020]. How to effectively reason over such a large repository of knowledge is an important unsolved problem in applying language models to mathematics [Irving et al., 2016, Wu et al., 2022b, Tworkowski et al., 2022] .

ProofNet falls short of requiring mastery of all of modern mathematics, but poses the still ambitious goal of reasoning over the core of an undergraduate mathematics, including basic analysis, algebra, and topology. We hope that this benchmark will spur the development of language models that are able to reason effectively over large knowledge bases.

2 The ProofNet Benchmark

Dataset collection Problems in the ProofNet benchmark are drawn from exercises in popular undergraduate mathematics textbooks. For a complete list of sources, see Appendix A.

Not all textbook exercises lend themselves naturally to formalization. In particular, we only consider for inclusion in ProofNet problems meeting the following criteria:

- *Self-containment*. Problems should only depend on the results commonly taught in an undergraduate curriculum. In particular, this rules out problems that are split into multiple sequentially dependent parts, or those using nonstandard notations.
- *Ease of formalization.* Not all kinds of mathematical problems can be naturally formalized, such as word problems, and such problems are excluded. We do not include exercises that require computing an unknown quantity. We do not include problems that depend on parts of Lean's mathlib that are relatively less mature, such as Euclidean geometry or combinatorics.
- *Low risk of train-test overlap.* Because language models are often pre-trained on large corpora mined from the internet that include mathlib, we refrain from including statements that are in mathlib or are likely to be added to mathlib in the future. In practice, this means we avoid the abstract "theory-building" style of theorems that constitute mathlib, and instead choose problems that involve applying general results to specific cases. For more insight into the stylistic differences between mathlib and ProofNet problems, see Appendix B.

Beyond the above criteria, problems were selected for broad coverage of the undergraduate curriculum and to range in difficulty from straightforward applications of the definitions to those requiring ingenuous creativity. Problems statements are transcribed into $\LaTeX$ and formalized by human annotators proficient in Lean. Natural language (informal) proofs are adapted from online solutions manuals, or in a few cases, written by the annotators.

Supported Tasks As ProofNet includes parallel natural language statements, natural language proofs, and formal statements, the dataset supports the evaluation of the following distinct tasks:

- *Formal theorem proving*. Given a formal statement of a theorem, produce a formal proof.
- *Informal theorem proving*. Given an informal statement, produce an informal proof.
- *Formalization and informalization of statements*. Given an informal (formal) statement, produce a corresponding formal (informal) statement.
- *Formalization of proofs*. Given an informal theorem statement, its informal proof, and its formal statement, produce a formal proof.

Table 1: Results of few-shot learning with LLMs on formalization and informalization of ProofNet statements. In addition to reporting autoformalization accuracy, we also report *typecheck rate*, which is the proportion of a model's samples that are accepted by Lean's type system. For the informalization task, we also report *compile rate*, i.e what proportion of the model's samples produce LaTeX that compiles. The most common reason why informal generations fail to compile is that they contain Unicode characters frequently used in Lean's mathlib but not accepted by the pdflatex compiler.

	Formalization		*Informalization*	
Model	Typecheck rate	Accuracy	Compile rate	Accuracy
GPT-J 6B	2.3%	0%	87.0%	9.8%
Code-davinci-002	20.2%	10.8%	100%	61.0%

3 Experiments

3.1 Methodology

In this work, we evaluate the capabilities of pre-trained language models on autoformalizing and informalizing theorem statements. Because of the technical challenges involved in building interactive proving environments in Lean [Han et al., 2021, Polu et al., 2022], we leave the investigation of autoformalizing proofs and formally proving ProofNet problems to future work.

Few-shot learning and large language models Few-shot learning is a simple and powerful method for adapting language models to sequence-to-sequence tasks [Brown et al., 2020]. We use few-shot learning with a manually constructed prompt as our baseline for autoformalizing and informalizing theorem statements. We perform inference using the OpenAI API's *Code-davinci-002* endpoint [Chen et al., 2021] and the GPT-J 6B model [Gao et al., 2021]. More details about prompting are given in Appendix C.

Because there are often many idiomatic ways of formalizing a given natural language theorem, all autoformalizations are evaluated for correctness by a human expert. Informalizations are also evaluated by a human.

3.2 Results and Discussion

In Table 1, we present our experimental results for few-shot formalization and informalization of ProofNet theorem statements. Although conceptually simple and easy to implement, our *Code-davinci-002* few-shot learning baseline achieves nontrivial performance, correctly formalizing 10.8% of theorems. GPT-J achieves near-trivial formalization performance, likely owing to its smaller parameter count.

Informalization accuracy is much higher than formalization accuracy for both models, supporting the intuitive claim that informalization is an easier task than formalization. This result also suggests that pre-trained language models have a strong grasp of the semantics of formal mathematics, and primarily struggle with generating lexically correct and type correct Lean code.

We further observe that among *Code-davinci-002*'s generations that type-check, roughly half are correct formalizations. This is consistent with our hypothesis that *Code-davinci-002* has a strong grasp of the semantics of mathematics, since the model performs well in the cases where it is able to generate valid Lean code.

We ground our remaining analysis in two case studies: one that shows a successful formalization produced by *Code-davinci-002*, and another that illustrates some of the model's most common failure cases.

Case study 1 The following is a correct and idiomatic Lean mathlib formalization of exercise 10.7.4 in Michael Artin's *Algebra* produced by *Code-davinci-002*:

NL: Let R be a ring, with M an ideal of R. Suppose that every element of R which is not in M is a unit of R. Prove that M is a maximal ideal and that moreover it is the only maximal ideal of R.

Code-davinci-002 output:
```
theorem exercise_10_7_10 {R : Type*} [ring R] (M : ideal R)
 (hM : ∀ x : R, x ∉ M → is_unit x) :
 is_maximal M ∧ ∀ (N : ideal R), is_maximal N → N = M
```

In this example, *Code-davinci-002* demonstrates strong knowledge of mathlib and some reasoning capability. For example, the model correctly identifies that ideals are expressed using a typeclass `ideal R`, but maximal ideals are expressed using a predicate `is_maximal`. Moreover, the model correctly recognizes that an instance of the `ideal` type can be implicitly coerced to a `set R` in order to apply the set-theoretic notation $\notin$. Finally, the model correctly infers that the best way to express "only maximal ideal" is the statement `∀ (N : ideal R), is_maximal N → N = M`, which demonstrates some logical reasoning ability.

Case study 2 The following is an incorrect formalization of exercise 16.6 from Sheldon Axler's *Linear Algebra* produced by *Code-davinci-002*.

NL: Suppose U is a subspace of V. Prove that $U^\perp = \{0\}$ if and only if $U = V$

Code-davinci-002 output:
```
theorem exercise_16_6 {K : Type*} [field K]
 {V : Type*} [add_comm_group V] [vector_space K V] {U : subspace V} :
 U.orthogonal = {0} ↔ U = ⊤
```

Here, *Code-davinci-002* makes three straightforward lexical mistakes: in mathlib, vector spaces are denoted by `module K V` rather than `vector_space K V`, subspaces of a vector space by `submodule K V` rather than `subspace V`, and the trivial subspace is denoted by $\bot$ rather than `{0}`. However, the model also makes a much more significant logical error. In order for the orthogonal complement $U^\perp$ of a subspace to make sense, the space must be endowed with a *inner product space* structure rather than merely a vector space structure, which is expressed in Lean as `inner_product_space K V`. Furthermore, inner product spaces are only defined over the real and complex fields, so one must also declare `[is_R_or_C K]`. Reliably inferring these kinds of implicit hypotheses is a major challenge for autoformalization systems. For the reference formalization of this exercise included in ProofNet, see Appendix D.

4 Related Work

Language modelling for proofs Language models found success in theorem proving both in the natural language setting [Lewkowycz et al., 2022, Welleck et al., 2021], and within many major ITPs such as Metamath [Polu and Sutskever, 2020], Isabelle [Jiang et al., 2022], and Lean [Han et al., 2021, Polu et al., 2022]. Popular benchmarks for evaluating language model-based provers are Hendrycks et al. [2021] and Welleck et al. [2021] for natural language, and Zheng et al. [2022] for ITPs.

Interactive Theorem Proving Work in formal theorem proving and autoformalization depends on libraries of formalized mathematics. This work directly depends on Lean's mathlib, but indirectly benefits form lessons learned from other proofs systems such as Coq [Bertot and Castéran, 2004], Mizar [Grabowski et al., 2010], and Isabelle [Nipkow et al., 2002].

5 Conclusion

We introduced ProofNet, a benchmarking consisting of parallel natural language theorem statements, natural language proofs, and formal theorem statements in Lean 3. We have shown that pre-trained large language models achieve non-trivial but far from consistent performance on the autoformalization of ProofNet statements. In the future, we hope to expand the number of exercises with informal proofs and increase the total number of exercises in the benchmark.

Acknowledgments

The authors would like to thank the Hoskinson Center for Formal Mathematics at Carnegie Mellon University for its generous funding and for providing a stimulating work environment. Piotrowski was supported also by the grant of National Science Center, Poland, no. 2018/29/N/ST6/02903, and by the Kosciuszko Fundation.

References

Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL `https://arxiv.org/abs/2005.14165`.

Kevin Buzzard, Johan Commelin, and Patrick Massot. Formalising perfectoid spaces. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 299–312. ACM, 2020. doi: 10.1145/3372885.3373830. URL `https://doi.org/10.1145/3372885.3373830`.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL `https://arxiv.org/abs/2107.03374`.

Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, *Conference on Automated Deduction (CADE) 2015*, pages 378–388. Springer, 2015.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2021. URL `https://arxiv.org/abs/2101.00027`.

Herbert L. Gelernter. Realization of a geometry theorem proving machine. In *IFIP Congress*, 1959.

Adam Grabowski, Artur Kornilowicz, and Adam Naumowicz. Mizar in a nutshell. *J. Formalized Reasoning*, 3(2):153–245, 2010.

Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W. Ayers, and Stanislas Polu. Proof artifact co-training for theorem proving with language models, 2021. URL `https://arxiv.org/abs/2102.06203`.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL `https://arxiv.org/abs/2103.03874`.

Geoffrey Irving, Christian Szegedy, Alexander A Alemi, Niklas Een, Francois Chollet, and Josef Urban. Deepmath - deep sequence models for premise selection. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing*

Systems, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/f197002b9a0853eca5e046d9ca4663d5-Paper.pdf.

Albert Q. Jiang, Wenda Li, Szymon Tworkowski, Konrad Czechowski, Tomasz Odrzygóźdź, Piotr Miłoś, Yuhuai Wu, and Mateja Jamnik. Thor: Wielding hammers to integrate language models and automated theorem provers, 2022. URL https://arxiv.org/abs/2205.10893.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models, 2022. URL https://arxiv.org/abs/2206.14858.

The mathlib Community. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM, jan 2020. doi: 10.1145/3372885.3373824. URL https://doi.org/10.1145%2F3372885.3373824.

Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. Springer, 2002.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving, 2020. URL https://arxiv.org/abs/2009.03393.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning, 2022. URL https://arxiv.org/abs/2202.01344.

Markus N. Rabe and Christian Szegedy. Towards the automatic mathematician. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction – CADE 28*, pages 25–37, Cham, 2021. Springer International Publishing. ISBN 978-3-030-79876-5.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, L. Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *ArXiv*, abs/1712.01815, 2017.

Christian Szegedy. A promising path towards autoformalization and general artificial intelligence. In Christoph Benzmüller and Bruce Miller, editors, *Intelligent Computer Mathematics*, pages 3–20, Cham, 2020. Springer International Publishing. ISBN 978-3-030-53518-6.

Szymon Tworkowski, Maciej Mikuła, Tomasz Odrzygóźdź, Konrad Czechowski, Szymon Antoniak, Albert Jiang, Christian Szegedy, Łukasz Kuciński, Piotr Miłoś, and Yuhuai Wu. Formal premise selection with language models, 2022. URL http://aitp-conference.org/2022/abstract/AITP_2022_paper_32.pdf.

Qingxiang Wang, Cezary Kaliszyk, and Josef Urban. First experiments with neural translation of informal to formal mathematics. In Florian Rabe, William M. Farmer, Grant O. Passmore, and Abdou Youssef, editors, *Intelligent Computer Mathematics*, pages 255–270, Cham, 2018. Springer International Publishing. ISBN 978-3-319-96812-4.

Sean Welleck, Jiacheng Liu, Ronan Le Bras, Hannaneh Hajishirzi, Yejin Choi, and Kyunghyun Cho. Naturalproofs: Mathematical theorem proving in natural language, 2021. URL https://arxiv.org/abs/2104.01112.

Sean Welleck, Jiacheng Liu, Ximing Lu, Hannaneh Hajishirzi, and Yejin Choi. Naturalprover: Grounded mathematical proof generation with language models, 2022. URL https://arxiv.org/abs/2205.12910.

Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models, 2022a. URL https://arxiv.org/abs/2205.12615.

Yuhuai Wu, Markus Norman Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. In *International Conference on Learning Representations*, 2022b. URL https://openreview.net/forum?id=TrjbxzRcnf-.

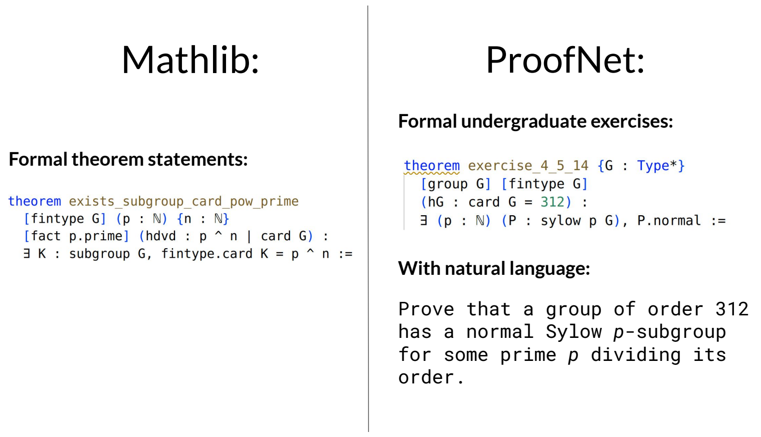
Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. minif2f: a cross-system benchmark for formal olympiad-level mathematics. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL `https://openreview.net/forum?id=9ZPegFuFTFv`.

A Problem Sources

The following is a complete list of sources ProofNet draws from:

- Analysis: Walter Rudin's *Principles of Mathematical Analysis*, Charles C. Pugh's *Real Mathematical Analysis*, Elias M. Stein and Rami Shakarchi's *Complex Analysis*.
- Linear Algebra: Sheldon Axler's *Linear Algebra Done Right*.
- Abstract Algebra: David S. Dummit and Richard M. Foote's *Abstract Algebra*, I.N. Herstein's *Abstract Algebra*, and Michael Artin's *Algebra*.
- Topology: James Munkres' *Topology*.

B Stylistic differences between mathlib and ProofNet



Mathlib:

Formal theorem statements:

```
theorem exists_subgroup_card_pow_prime
  [fintype G] (p : ℕ) {n : ℕ}
  [fact p.prime] (hdvd : p ^ n | card G) :
  ∃ K : subgroup G, fintype.card K = p ^ n :=
```

ProofNet:

Formal undergraduate exercises:

```
theorem exercise_4_5_14 {G : Type*}
  [group G] [fintype G]
  (hG : card G = 312) :
  ∃ (p : ℕ) (P : sylow p G), P.normal :=
```

With natural language:

Prove that a group of order 312 has a normal Sylow *p*-subgroup for some prime *p* dividing its order.

A sample theorem statement from mathlib, show on the left, and a sample theorem statement from ProofNet, shown on the right. Mathlib emphasizes including the most abstract and general formulations of mathematical results, whereas ProofNet predominantly tests the ability of models to apply those results in concrete instances.

C Few-shot prompting

For our prompting *Code-davinci-002* for the autoformalization task, we use the following 12-shot prompt.

```
Natural language version:
"Let $P$ be a $p$-subgroup of $G$. Then $P$ is contained in a Sylow $p$-subgroup of
$G$."
Translate the natural language version to a Lean mathlib version:
theorem exists_le_sylow {p : ℕ} {G : Type*} [group G] {P : subgroup G} (hP :
is_p_group p P) : ∃ (Q : sylow p G), P ≤ Q :=

Natural language version:
"Let $E$ and $F$ be complex normed spaces and let $f:E\to F$. If $f$ is
differentiable and bounded, then $f$ is constant."
Translate the natural language version to a Lean mathlib version:
theorem exists_eq_const_of_bounded {E : Type u} [normed_group E] [normed_space ℂ E]
{F : Type v} [normed_group F] [normed_space ℂ F] {f : E → F} (hf : differentiable ℂ
f) (hb : metric.bounded (set.range f)) : ∃ (c : F), f = function.const E c :=

Natural language version:
"Let $X$ be a topological space; let $A$ be a subset of $X$. Suppose that for each
$x\in A$ there is an open set $U$ containing $x$ such that $U\subset A$. Then $A$
is open in $X$."
Translate the natural language version to a Lean mathlib version:
theorem subset_of_open_subset_is_open (X : Type*) [topological_space X] (A : set X)
(hA : ∀ x ∈ A, ∃ U : set X, is_open U ∧ x ∈ U ∧ U ⊆ A): is_open A :=

Natural language version:
"Two multiplicative functions $f, g:\mathbb{N}\to R$ are equal if and only if
$f(p^i)=f(g^i)$ for all primes $p$."
Translate the natural language version to a Lean mathlib version:
theorem is_multiplicative.eq_iff_eq_on_prime_powers {R : Type*}
[comm_monoid_with_zero R] (f : nat.arithmetic_function R) (hf :
f.is_multiplicative) (g : nat.arithmetic_function R) (hg : g.is_multiplicative) : f
= g ↔ ∀ (p i : ℕ), nat.prime p → f (p ^ i) = g (p ^ i) :=

Natural language version:
"If $z_1, \\dots, z_n$ are complex, then $|z_1 + z_2 + \\dots + z_n|\\leq |z_1| +
|z_2| + \\dots + |z_n|$."
Translate the natural language version to a Lean mathlib version:
theorem abs_sum_leq_sum_abs (n : ℕ) (f : ℕ → ℂ) : abs (∑ i in finset.range n, f i)
≤ ∑ i in finset.range n, abs (f i) :=

Natural language version:
"If x and y are in $\\mathbb{R}^n$, then $|x+y|^2 + |x-y|^2 = 2|x|^2 + 2|y|^2$."
Translate the natural language version to a Lean mathlib version:
theorem sum_add_square_sub_square_eq_sum_square (n : ℕ) (x y : euclidean_space ℝ
(fin n)) : □x + y□^2 + □x - y□^2 = 2*□x□^2 + 2*□y□^2 :=

Natural language version:
"If $x$ is an element of infinite order in $G$, prove that the elements $x^n$,
$n\\in\\mathbb{Z}$ are all distinct."
Translate the natural language version to a Lean mathlib version:
theorem distinct_powers_of_infinite_order_element (G : Type*) [group G] (x : G)
(hx_inf : ∀ n : ℕ, x ^ n ≠ 1) : ∀ m n : ℤ, m ≠ n → x ^ m ≠ x ^ n :=

Natural language version "A set of vectors $\{v_i\}_{i\in I}$ orthogonal with
respect to some bilinear form $B: V\times V\to K$ is linearly independent if for
all $i \in I, B(v_i, v_i)\neq 0$."
Translate the natural language version to a Lean mathlib version:
theorem linear_independent_of_is_Ortho {V K : Type*} [field K] [add_comm_group V]
[module K V] {n : Type*} {B : bilin_form K V} {v : n → V} (hv₁ : B.is_Ortho v) (hv₂
: ∀ (i : n), ¬B.is_ortho (v i) (v i)) : linear_independent K v :=

Natural language version:
"Suppose that $V$ is an $n$-dimensional vector space.  Then for some set of vectors
$\{v_i\}_{i=1}^k$, if $k>n$ then there exist scalars $f_1, \dots, f_k$ such that
$\sum_{i=1}^k f_kv_k = 0$."
Translate the natural language version to a Lean mathlib version:
theorem exists_nontrivial_relation_sum_zero_of_dim_succ_lt_card {K V : Type*}
[division_ring K] [add_comm_group V] [module K V] [finite_dimensional K V] {t :
finset V} (h : finite_dimensional.finrank K V + 1 < t.card) : ∃ (f : V → K), t.sum
(λ (e : V), f e • e) = 0 ∧ t.sum (λ (e : V), f e) = 0 ∧ ∃ (x : V) (H : x ∈ t), f x
≠ 0 :=

Natural language version:
"A group is commutative if the quotient by the center is cyclic."
Translate the natural language version to a Lean mathlib version:
theorem comm_group_of_cycle_center_quotient {G H : Type*} [group G] [group H]
[is_cyclic H] (f : G →* H) (hf : f.ker ≤ center G) : comm_group G :=

Natural language version:
```

```
"If $H$ is a $p$-subgroup of $G$, then the index of $H$ inside its normalizer is
congruent modulo $p$ to the index of $H$."
Translate the natural language version to a Lean mathlib version:
theorem card_quotient_normalizer_modeq_card_quotient {G : Type*} [group G] [fintype
G] {p n : ℕ} [hp : fact (nat.prime p)] {H : subgroup G} (hH : card H = p ^ n) :
card (H.normalizer ▯ subgroup.comap H.normalizer.subtype H) ≡ card (G ▯ H) [MOD p]
:=

Natural language version:
"Suppose $X, Y, Z$ are metric spaces, and $Y$ is compact.  Let $f$ map $X$ into
$Y$, let $g$ be a continuous one-to-one mapping of $Y$ into $Z$, and put
$h(x)=g(f(x))$ for $x \in X$. Prove that $f$ is uniformly continuous if $h$ is
uniformly continuous."
Translate the natural language version to a Lean mathlib version:
theorem uniform_continuous_of_continuous_injective_uniform_continuous_comp {X Y Z :
Type*} [metric_space X] [metric_space Y] [metric_space Z] (hY : compact_space Y) (f
: X → Y) (g : Y → Z) (hgc : continuous g) (hgi : function.injective g) (h :
uniform_continuous (g ∘ f)) : uniform_continuous f :=
```

Because GPT-J has a shorter context than *Davinci-codex-002* (2048 tokens compared to 4096), we only use the last six examples when prompting GPT-J. When prompting models for informalization, we use the same examples as for formalization, but swapped around according to the following format:

```
Lean mathlib version:
theorem exists_le_sylow {p : ℕ} {G : Type*} [group G] {P : subgroup G}
  (hP : is_p_group p P) :
  ∃ (Q : sylow p G), P ≤ Q :=
Translate the Lean mathlib version to a natural language version "Let $P$ be a
$p$-subgroup of $G$. Then $P$ is contained in a Sylow $p$-subgroup of $G$."
```

D Case study 2 reference

The following is ProofNet's reference formal statement for case study 2

```
theorem exercise_16_6 {K : Type*} [is_R_or_C K] [inner_product_space K V]
 (U : submodule K V) :
 U.orthogonal = ⊥ ↔ U = ⊤
```