Exploring Mathematical Conjecturing with Large Language Models

Moa Johansson[1], Nicholas Smallbone[1]

[1]*Chalmers University of Technology, Gothenburg, Sweden*

Abstract

The task of automating the discovery of mathematical conjectures has so far primarily been addressed in symbolic systems. However, a neuro-symbolic architecture seems like an excellent fit for this task. We can assign the generative task to a neural system without much risk, even if a few non-theorems slip through, the results are checked afterwards using a symbolic theorem prover or counter-example finder. In this initial case-study, we investigate the capabilities of GPT-3.5 and GPT-4 on this task. While results are mixed, we see potential in improving on the weaknesses of purely symbolic systems. A neuro-symbolic theory exploration system could, for instance, add some more variation in conjectures over purely symbolic systems while not missing obvious candidates.

Keywords

Theory Exploration, Conjecturing, Large Language Models

1. Introduction

We are interested in the problem of inventing new conjectures about a set of functions and datatypes, known as *theory exploration* or *automated conjecturing*. In this paper we discuss the potential of using a large language model (LLM) as part of a neuro-symbolic theory exploration system and conduct some initial experiments.

The problem of mathematical discovery has a long history in symbolic AI research with early systems based on specialised heuristics such as AM and Graffiti [1, 2], on to later works like HR (discovery of integer sequences) and MATHsAiD (algebra) [3, 4]. Our own research has focused on theory exploration for finding auxiliary lemmas for automating proofs by induction, or for investigating the properties of a functional program [5, 6]. We have studied this extensively using symbolic and heuristic methods, which perform well at generating relevant and interesting lemmas. What is considered interesting and relevant of course depend on context: typically a lemma is considered interesting if its proof is non-trivial, given current knowledge. We have built several systems, e.g. QuickSpec [7], which discovers equational conjectures and its extension, Hipster [8], which also searches for proofs in the proof assistant Isabelle/HOL. These system run on a regular laptop, are fast and perform well for small to moderate sized inputs (up to around 10 different functions, and term sizes up to 7–9 symbols on each side of the equation).

NeSy 2023, 17th International Workshop on Neural-Symbolic Learning and Reasoning, Certosa di Pontignano, Siena, Italy

moa.johansson@chalmers.se (M. Johansson); nicsma@chalmers.se (N. Smallbone)

0000-0002-1097-8278 (M. Johansson); 0000-0003-2880-6121 (N. Smallbone)

CEUR Workshop Proceedings (CEUR-WS.org)

Given larger theories, or asked for larger term sizes, the increased search space usually requires more computing power than a laptop to run quickly. Furthermore, the number of possible conjectures increases a lot, so the system tends to produce too many conjectures, and many of the conjectures are uninteresting. One solution to this was suggested in [9], where the user provides a set of templates describing the shapes of conjectures to explore, thus restricting the search space to lemmas which the user thinks has an appealing shape. We now want to take this one step further: can common templates be extracted from data? Could we even use a large language model to generate conjectures by analogy to previously seen mathematical statements directly?

As a case study, we investigate the capabilities of ChatGPT and its underlying LLMs, GPT-3 and GPT-4 [10, 11], on this task. We limit ourselves to generating conjectures and proofs in the syntax of the proof assistant Isabelle/HOL, which we can then run to check for validity. For Isabelle/HOL, there is prior work using various LLMs to generate formal versions of natural language statements [12, 13], as well as proof scripts [14, 15], but none of these have considered conjecture generation based on function and datatype definitions. Neural conjecturing has however been attempted for other systems, for instance Mizar [16], where a GPT-2 model is trained on the Mizar Mathematical Library. With a medium temperature setting, they managed to generate some novel and interesting lemmas, while lower settings repeated existing lemmas, and higher settings tended towards outputs in inconsistent or unparsable syntax. Another example is the skip-tree transformer model trained on data for the HOL Light proof assistant [17]. Here, between 10–30% of the generated conjectures were both true and novel, while the reminder consisted of instances of theorems from the training data, or false conjectures.

2. Case-study: GPT-4 as a Theory Exploration System

As a first experiment, we wanted to investigate the basic theory exploration capabilities of GPT-4, zero-shot, without any further fine-tuning. How good is it, given an Isabelle theory file, at generating interesting properties and proofs? How does it compare to symbolic systems such as Hipster? Has it been able to extract common patterns for equational statements? GPT-4 was at the time of writing only available via ChatGPT, so our experiments took the form of conversations with ChatGPT.

Caveat: GPT-4 has been trained on large amounts of code from GitHub. We found evidence (see Appendix B) that it has also been trained on the GitHub repositories of our symbolic theory exploration tools QuickSpec and Hipster, as it can produce output in exactly the same syntax as our tools. Likewise, it appears to have been trained on all the online libraries and proof archives of the Isabelle/HOL proof assistant. This makes it difficult to evaluate how good GPT-4 really is. To what extent is it generalising to suggest plausible lemmas and proof structures, or simply recalling a training example it has already seen?

2.1. Insertion sort

We started with the following Isabelle theory defining an insertion sort function, taken from Hipster:

```
theory Sorted
imports Main
begin
fun sorted :: "int list => bool"
where
  "sorted [] = True"
| "sorted [x] = True"
| "sorted (x # y # xs) = (x <= y /\ sorted (y#xs))"

fun ins :: "int => int list => int list"
where
  "ins x [] = [x]"
| "ins x (y#ys) = (if x <= y then (x#y#ys) else (y#(ins x ys)))"

fun isort :: "int list => int list"
where
  "isort [] = []"
| "isort (x#xs) = ins x (isort xs)"
end
```

The aim is to discover and prove that `isort` produces a sorted list. The proof requires a key lemma, also to be discovered, that inserting into a sorted list preserves its sortedness. Hipster can discover both lemmas and prove them automatically by structural induction.

We prompted GPT-4 with the Isabelle theory, followed by the sentence "Generate some lemmas about sorted, insert and isort, in Isabelle syntax." It responded with a syntactically-correct list of Isabelle lemmas and proofs, shown in full in Appendix A. Here we show an excerpt: the list of discovered lemmas, together with the proof of the lemma `sorted (isort xs)`.

```
sorted [x] = True
sorted xs ==> sorted (ins x xs)
sorted (isort xs)
length (ins x xs) = length xs + 1
length (isort xs) = length xs
count_list (ins x xs) a = count_list xs a +
  (if x = a then 1 else 0)
count_list (isort xs) a = count_list xs a
```

```
lemma isort_sorted [simp]: "sorted (isort xs)"
proof (induct xs)
case Nil
then show ?case by simp
next
case (Cons a xs)
then show ?case by (simp add: ins_sorted)
qed
```

Figure 1: The lemmas produced by GPT-4 for the insertion sort example, and its proof of lemma `isort_sorted`.

The lemmas are all correct and useful, and include the theorem that insertion sort produces a sorted list, the key lemma `ins_length` used in that proof, and other lemmas involving standard Isabelle functions such as `length`. All of the proofs are correct except for the key lemma `ins_sorted`, i.e. `sorted xs ==> sorted (ins x xs)`. That proof follows the correct structure (induction and two case splits), and most of the reasoning is correct, but one step is false (and can be removed) and two have incorrect justifications.

Performance of other LLMs: We also ran the experiment using a few other large language models. We did not have much success running the experiment using Codex, the variant of GPT-3.5 specifically designed for code generation, nor with any of the smaller versions of GPT-3.5 (ada, babbage, curie), but the largest version davinci produced good results. It did produce useful lemmas, but more of the proofs were incorrect compared to GPT-4. We also briefly experimented with the web interface of the open source BLOOM model[1] [18]. Due to its limited output size (64 tokens) it could only generate one lemma (in greedy mode) with proof: `(sorted (isort xs)) = True`, and the proof was not correct. A more thorough evaluation of additional models is further work.

2.2. Functional geometry

As GPT-4 has most likely seen the lists theory already, we decided to test it on a lesser-known theory, *functional geometry* [19]. It defines functions for building and combining *drawings*:

- `blank` represents a blank drawing;
- `over d1 d2` superimposes `d1` and `d2`;
- `beside d1 d2` draws `d1` next to `d2`;
- `above d1 d2` draws `d1` above `d2`;
- `rot d` draws `d` rotated clockwise by 90 degrees;
- `rot45 d` draws `d` rotated clockwise by 45 degrees;
- `flip d` draws `d` flipped horizontally.

We previously tested functional geometry using QuickSpec [7], which was able to discover many relevant lemmas. It has not to our knowledge been formalised in Isabelle before.

We prompted ChatGPT with the Isabelle theory file followed by the text "Generate lemmas in Isabelle format about the functions blank, over, beside, above, rot, flip and rot45. Do not prove the lemmas – use 'sorry'.[2]" It produced a list of syntactically correct Isabelle lemmas, each followed correctly by "sorry". However, the output varies drastically from run to run.

Here are the lemmas suggested by one run of GPT-4 – we have indicated which lemmas are correct by putting a star next to them. Notice that in this run, it mostly suggests "generic" lemmas such as associativity, commutativity and distributivity:

```
(*) over blank blank = blank
(*) over (over A B) C = over A (over B C)
(*) over A B = over B A
    beside (beside A B) C =
      beside A (beside B C)
    above (above A B) C =
      above A (above B C)
    rot (rot A) = A
(*) flip (flip A) = A
(*) rot45 (rot45 A) = rot A
    rot (flip A) = flip (rot A)
```

[1] https://huggingface.co/spaces/huggingface/bloom_demo

[2] `sorry` is an Isabelle command that skips a proof. We used it in order to reduce the size of the output, as in this case study, all the proofs can be solved automatically by Isabelle's `simp` tactic.

```
(*) rot (beside A B) =
      above (rot A) (rot B)
    flip (beside A B) =
      beside (flip A) (flip B)
(*) rot45 (over A B) =
      over (rot45 A) (rot45 B)
    rot45 (beside A B) =
      beside (rot45 A) (rot45 B)
    rot45 (above A B) =
      above (rot45 A) (rot45 B)
```

Here is a more successful run, where the system suggested lemmas specific to drawing, such as the fact that rotating a drawing by 45 degrees 8 times is the identity function:

```
(*) over blank d = d
(*) over (over d1 d2) d3 =
      over d1 (over d2 d3)
(*) over d1 d2 = over d2 d1
    beside (beside d1 d2) d3 =
      beside d1 (beside d2 d3)
    above (above d1 d2) d3 =
      above d1 (above d2 d3)
(*) rot (rot (rot (rot d))) = d
(*) flip (flip d) = d
(*) rot45 (rot45 (rot45 (rot45 (rot45
      (rot45 (rot45 (rot45 d))))))) = d
```

We found that GPT-4 usually produces the same kind of "generic" lemmas every time: (1) Binary functions are associative and commutative, and have the empty drawing as identity element. (2) Pairs of binary functions distribute over one another. (3) Unary functions are their own inverse (e.g. *rot (rot x) = x*), and commute with each other (e.g. *rot (flip x) = flip (rot x)*). Exactly which functions have these properties varies from run to run. *Most of these generic lemmas are false.*

About 50% of the time, it also produces useful drawing-specific lemmas such as *rot45 (rot45 x) = rot x* or *rot (rot (rot (rot x))) = x*. However, it also quite often produces muddled versions of these lemmas, where e.g. *rot* and *rot45* swap places or the drawing is rotated the wrong number of times. Sometimes it produces two mutually contradictory lemmas. And often lemmas are seemingly random combinations of drawing functions.

We also note that GPT-4 seems to have learnt useful "templates" for lemmas. For example, the output often states that a binary function is associative, regardless of the function name. This suggests that GPT-4 can be used to suggest lemma templates for theory exploration systems such as [9].

To summarise, GPT-4 is able to find useful lemmas much of the time, but each run gives a different set of lemmas, so there is not much consistency, and we risk missing interesting lemmas. There are some wrong lemmas, but that is not a problem in a theory exploration system, since the lemmas will be checked by Isabelle. By repeatedly running GPT-4, we do however obtain a useful and descriptive set of lemmas for the drawing theory.

GPT-4 vs symbolic theory exploration We also ran the symbolic theory exploration system QuickSpec [7] on the drawing theory. The results are shown in Appendix C; all the lemmas discovered by QuickSpec are correct.

On the one hand, QuickSpec covers the space of lemmas much more systematically than GPT-4. It discovers all the most important lemmas, e.g.:

- which functions are associative, commutative, etc. (the simple lemma (2) *over x x = x* is one that GPT-4 never generated);
- how functions distribute over each other (e.g. (7) *rot (beside y x) = above (rot x) (rot y)*);
- rotating a drawing by 90 degrees four times is the identity function (10);
- equivalent ways to lay out four drawings in a 2x2 grid, (4) *above (beside x y) (beside z w) = beside (above x z) (above y w)*.

On the other hand, QuickSpec fails to discover important lemmas about *rot45*, such as that rotating a document by 45 degrees 8 times is the identity function, and that two 45-degree rotations are equivalent to a 90-degree rotation (*rot45 (rot45 x) = rot x*). The first lemma is beyond QuickSpec's term size limit; GPT-4 can generate arbitrarily complex lemmas. Looking into why the second lemma was not discovered, we were surprised to discover that it does not hold, because of a bug in the implementation of *rot45*! This is an important difference between symbolic and neural theory exploration: a symbolic system can only suggest lemmas that hold in the explored theory, whereas a language model can suggest lemmas that perhaps the user might *want to hold*, based on its interpretation of what the user meant. In other words, the language model is not overly affected by minor errors in the theory.

Turning up the temperature Large language models have a "temperature" parameter, which controls how random its output is. At low temperature, outputs are nearly deterministic, and at high temperature quite unpredictable. We experimented with running ChatGPT at different temperatures, to see whether it might be more "creative" in its lemmas at higher temperature. This experiment used GPT-3.5 rather than GPT-4, since GPT-4 did not allow customising the temperature at the time of writing.

We found that reducing the temperature, even to 0, had no noticeable effect on the output. Increasing it from its default of 0.7 to 1, we saw a very slight increase in the variety of laws, though most were the same as before. However, it also started to produce many laws involving non-existent function symbols, such as `dist (a,b) = dist (b,c) ==> dist (a,b) = sqrt 2 ==> (a,c) in rot45_drawing x a b c`. Increasing the temperature further, the output became incoherent: `theorem [simp]: "`
`<And>w h nvt dvt. below vfinal besiden = above besiden_sym hbesiden_tsh"`

Obscuring the names We wanted to see how much GPT-4's output was influenced by the names of the functions and types. We therefore renamed all the symbols in the drawing theory, so that the type *Drawing* became *Monkey*, *beside* became *hello*, and so on. We prompted GPT-4 to generate lemmas for the program. There was a clear effect: all of the drawing-specific lemmas (such as *rot (rot (rot (rot x))) = x*) disappeared from the output, and only the generic lemmas (such as associativity and commutativity) remained. Evidently, the names of the functions are highly significant for GPT-4 – an important difference compared to symbolic methods.

2.3. Mathematical theories

Finally, we tested GPT-4 on some more mathematical theories from the Archive of Formal Proofs. These do not contain many function definitions, but largely consist of theorems and

proofs. We prompted GPT-4 to suggest useful additional lemmas, but not to provide proofs. We tried two theories.

1. Suppes' theorem for probability logic[3]: Here GPT-4 produced many standard theorems from probability, giving them appropriate names, such as `bayes_theorem`. Many lemmas were generalisations of lemmas occurring in the theory file. GPT-4 has most likely not been trained on this theory (which was added in 2023), but has probably seen probability theory from other sources.
2. Forcing[4] (a topic from set theory): here GPT-4 only produced lemmas that were already present in the theory file (usually even copying the name of the lemma). GPT-4 may have seen this theory (which was added in 2020) in its training, but forcing is likely a much less common topic than probability in the training data.

This suggests that GPT-4 is able to identify related lemmas in domains it recognises, but has little ability to create new lemmas in domains it is unfamiliar with. By contrast, symbolic theory exploration systems can explore new and unfamiliar theories just as well as familiar ones.

3. Discussion

Theory exploration seems like a domain where a neuro-symbolic architecture would be a perfect fit. So far most work in conjecture discovery has been using symbolic systems based on heuristic search over type-correct terms, then passed to a (symbolic) theorem proving system. The task of conjecturing can however benefit from the capabilities of LLMs of recognising patterns and effectively suggesting conjectures by analogy. Hallucinations are not a severe problem, as long as enough conjectures are true and interesting: if the LLM produces some conjectures that are false, they will be caught by the theorem prover or counter-example checker.

GPT-4 used as a theory explorer has some desirable features, some of which are missing from symbolic systems:

Names are informative. A LLM pays attention to the actual names of functions and types, and can thus occasionally mix in an additional function the user had not thought of, but which often makes sense to include.

Less restricted syntax. The neural system is less restricted than the symbolic system with respect to the shape of the lemmas. It can generate both equalities and implications, without being explicitly instructed to.

Ability to recognise common patterns. We also noticed that GPT-4 had a tendency to suggest common algebraic patterns which we expect to see, e.g. associativity for most binary functions. If the search space is large, focusing on such properties can be advantageous.

Properties of buggy functions. We notice that even for a function containing a subtle bug, GPT-4 produced sensible conjectures, that probably *ought* to hold. If these match the

[3] https://www.isa-afp.org/entries/Suppes_Theorem.html
[4] https://www.isa-afp.org/entries/Forcing.html

user's expectation, but the proof assistant can find a counter-example, they serve as good indications of a bug being present. A symbolic system, like QuickSpec, would instead generate other properties (or none at all), that do hold about the buggy function. This may be harder for the user to make sense of.

There are however also some disadvantages of using GPT-4 and LLMs in general for conjecture generation:

Difficulty of fair evaluation. The system is hidden behind a proprietary API, so we do not exactly know how it works. It is also impossible to know exactly what has been seen already during training to quantify which results are due to genuine capabilities of analogy formation, and which ones are repeating examples seen during training.

Coverage. The LLM tends to generate a limited number of conjectures, and as it is stochastic, it may well produce different conjectures on different runs. A symbolic system like QuickSpec, on the other hand, systematically explores the space of possible lemmas, and the user has control over the search strategy and will know what ought to be generated (or skipped). A neural system provides no such structure – the user cannot get an idea of how well the space of possible conjectures has been covered.

Hardware and energy consumption. A symbolic system like QuickSpec comfortably runs on a regular laptop and requires no prior training. GPT-4 on the other hand requires specialised hardware and consumes much more energy to train and query. How much better performance should we require for this much higher energy cost?

Payment. GPT-4 is a commercial product, and access costs money. It is questionable if a niche application like conjecture discovery for theorem proving is something anyone wants to pay for in the long run.

It is certainly interesting to continue investigations into integrating large language models in theory explorations, as they do have many desirable features, not covered by fully symbolic systems. From a research point of view, it would be desirable to work with a more transparent model, enabling more scientific evaluation practices. However, we noticed that we had the most success with generalisation and reasoning by analogy when using the largest models, GPT-3.5 davinci and GPT-4. Possibly, lemma generation thus benefits both from the larger underlying neural network, and having access to a large context window. Further experiments in systematically comparing also additional open source LLMs, are planned for further work. Another open question is whether a smaller, carefully trained network could perform the task equally well.

In some cases the symbolic and neural systems produce complementary sets of conjectures. One option is to run both systems in combination, using the symbolic system to produce a reliable baseline set of lemmas, and the neural system for creativity. A more sophisticated approach is to run both systems in combination, using the output of the symbolic system as part of a prompt to a fine-tuned neural system, giving the neural system some more information about a new theory.

References

[1] D. B. Lenat, AM, an artificial intelligence approach to discovery in mathematics as heuristic search, 1976.

[2] S. Fajtlowicz, On conjectures of Graffiti, Annals of Discrete Mathematics 38 (1988) 113–118.

[3] S. Colton, The HR program for theorem generation, in: A. Voronkov (Ed.), Automated Deduction—CADE-18, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 285–289.

[4] R. L. McCasland, A. Bundy, P. F. Smith, MATHsAiD: Automated mathematical theory exploration, Applied Intelligence (2017). URL: https://doi.org/10.1007/s10489-017-0954-8. doi:10.1007/s10489-017-0954-8.

[5] M. Johansson, L. Dixon, A. Bundy, Conjecture synthesis for inductive theories, Journal of Automated Reasoning 47 (2011) 251–289. URL: https://doi.org/10.1007/s10817-010-9193-y. doi:10.1007/s10817-010-9193-y.

[6] K. Claessen, M. Johansson, D. Rosén, N. Smallbone, Automating inductive proofs using theory exploration, in: Proceedings of the Conference on Automated Deduction (CADE), volume 7898 of *LNCS*, Springer, 2013, pp. 392–406.

[7] N. Smallbone, M. Johansson, K. Claessen, M. Algehed, Quick specifications for the busy programmer, Journal of Functional Programming 27 (2017). doi:10.1017/S0956796817000090.

[8] M. Johansson, D. Rosén, N. Smallbone, K. Claessen, Hipster: Integrating theory exploration in a proof assistant, in: Proceedings of CICM, Springer, 2014, pp. 108–122.

[9] S. H. Einarsdóttir, N. Smallbone, M. Johansson, Template-based theory exploration: Discovering properties of functional programs by testing, in: Proceedings of IFL'20, 2021.

[10] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, volume 33, Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[11] OpenAI, GPT-4 Technical Report, Technical Report, 2023. URL: https://cdn.openai.com/papers/gpt-4.pdf.

[12] A. Lewkowycz, A. J. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, Y. Wu, B. Neyshabur, G. Gur-Ari, V. Misra, Solving quantitative reasoning problems with language models, in: A. H. Oh, A. Agarwal, D. Belgrave, K. Cho (Eds.), Advances in Neural Information Processing Systems, 2022. URL: https://openreview.net/forum?id=IFXTZERXdM7.

[13] Y. Wu, A. Q. Jiang, W. Li, M. N. Rabe, C. E. Staats, M. Jamnik, C. Szegedy, Autoformalization with large language models, in: A. H. Oh, A. Agarwal, D. Belgrave, K. Cho (Eds.), Advances in Neural Information Processing Systems, 2022. URL: https://openreview.net/forum?id=IUikebJ1Bf0.

[14] A. Q. Jiang, W. Li, S. Tworkowski, K. Czechowski, T. Odrzygóźdź, P. Miłoś, Y. Wu, M. Jamnik, Thor: Wielding hammers to integrate language models and automated theorem provers,

in: A. H. Oh, A. Agarwal, D. Belgrave, K. Cho (Eds.), Advances in Neural Information Processing Systems, 2022. URL: https://openreview.net/forum?id=fUeOyt-2EOp.

[15] E. First, M. N. Rabe, T. Ringer, Y. Brun, Baldur: Whole-proof generation and repair with large language models, 2023. URL: https://arxiv.org/abs/2303.04910. doi:10.48550/ARXIV.2303.04910.

[16] J. Urban, J. Jakubův, First neural conjecturing datasets and experiments, in: C. Benzmüller, B. Miller (Eds.), Intelligent Computer Mathematics, Springer International Publishing, Cham, 2020, pp. 315–323.

[17] M. N. Rabe, D. Lee, K. Bansal, C. Szegedy, Mathematical reasoning via self-supervised skip-tree training, in: International Conference on Learning Representations, 2021. URL: https://openreview.net/forum?id=YmqAnY0CMEy.

[18] T. L. Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon, M. Gallé, et al., Bloom: A 176b-parameter open-access multilingual language model, arXiv preprint arXiv:2211.05100 (2022).

[19] P. Henderson, Functional geometry, Higher-Order and Symbolic Computation 15 (2002) 349–365.

A. Output of GPT-4 on the insertion sort theory

```
lemma sorted_single [simp]: "sorted [x] = True"
by simp

lemma ins_sorted [simp]: "sorted xs ==> sorted (ins x xs)"
proof (induct xs)
case Nil
then show ?case by simp
next
case (Cons a xs)
then show ?case
proof (cases xs)
case Nil
then show ?thesis by simp
next
case (Cons b xs)
then show ?thesis
proof (cases "x <le> a")
case True
then show ?thesis by (simp add: Cons)
next
case False
then have "ins x (a # b # xs) = a # ins x (b # xs)" by simp
also have "... = a # b # ins x xs" using Cons by simp
finally show ?thesis using Cons by (simp add: sorted.simps(3))
qed
qed
qed

lemma isort_sorted [simp]: "sorted (isort xs)"
proof (induct xs)
case Nil
then show ?case by simp
next
case (Cons a xs)
then show ?case by (simp add: ins_sorted)
qed

lemma ins_length [simp]: "length (ins x xs) = length xs + 1"
proof (induct xs)
case Nil
then show ?case by simp
```

```
next
case (Cons a xs)
then show ?case by (simp split: if_splits)
qed

lemma isort_length [simp]: "length (isort xs) = length xs"
proof (induct xs)
case Nil
then show ?case by simp
next
case (Cons a xs)
then show ?case by (simp add: ins_length)
qed

lemma ins_permutes [simp]: "count_list (ins x xs) a = count_list xs a + (if x = a then 1
    else 0)"
proof (induct xs)
case Nil
then show ?case by simp
next
case (Cons b xs)
then show ?case by (simp split: if_splits)
qed

lemma isort_permutes [simp]: "count_list (isort xs) a = count_list xs a"
proof (induct xs)
case Nil
then show ?case by simp
next
case (Cons b xs)
then show ?case by (simp add: ins_permutes)
qed
```

B. ChatGPT mimicking Hipster's output

MO generate some lemmas about sorted in Isabelle syntax

Sure! Here are the lemmas that Hipster generates for the `sorted` function in Isabelle syntax:

less Copy code

```
lemma lemma_a [thy_expl]: "sorted xs ⟹ sorted (snoc xs x)"
  apply (induct xs arbitrary: x)
  apply simp
  apply (metis Sorted.snoc.elims hd_Cons_tl last.simps list.sel(1) list.simp
  done

lemma lemma_aa [thy_expl]: "sorted xs ⟹ sorted (rev xs)"
  apply (induct xs)
  apply simp
  apply (metis Sorted.snoc.simps lemma_a rev.simps(2) snoc.simps)
  done

lemma lemma_ab [thy_expl]: "sorted xs ⟹ sorted (map f xs)"
  apply (induct xs)
  apply simp
  apply (simp add: sorted_Cons)
  apply (metis sorted.simps sorted_Cons)
  done
```

Figure 2: GPT-3.5 mimicking Hipster's output. Here, ChatGPT has first been presented with an Isabelle theory defining some functions about lists, and then asked to "Generate some lemmas in Isabelle syntax". The theory contained the import statement `imports "$HIPSTER_HOME/IsaHipster"`, which activates the theory exploration tool Hipster for Isabelle. We note that it exactly mimics Hipster's naming scheme for discovered lemmas (lemma_a, lemma_aa, lemma_ab and so on), and that it adds the tag `[thy_expl]` which Hipster puts on lemmas it has discovered. Furthermore, the proof scripts look very similar to what Hipster would produce using induction, simplification in the base-case and then and calls to automated provers via Sledgehammer to close the step-case. However, note that the proof scripts are not quite right, for instance lemma_ab there is a reference to a non-existent lemma `sorted_Cons`.

C. Output of QuickSpec on the drawing theory

```
== Functions ==
over :: Drawing -> Drawing -> Drawing

== Laws ==
  1. over x y = over y x
  2. over x x = x
  3. over (over x y) z = over x (over y z)

== Functions ==
beside :: Drawing -> Drawing -> Drawing
 above :: Drawing -> Drawing -> Drawing

== Laws ==
  4. above (beside x y) (beside z w) = beside (above x z) (above y w)
  5. over (above x y) (above z w) = above (over x z) (over y w)
  6. over (beside x y) (beside z w) = beside (over x z) (over y w)

== Functions ==
rot :: Drawing -> Drawing

== Laws ==
  7. above (rot x) (rot y) = rot (beside y x)
  8. beside (rot x) (rot y) = rot (above x y)
  9. over (rot x) (rot y) = rot (over x y)
 10. rot (rot (rot (rot x))) = x

== Functions ==
flip :: Drawing -> Drawing

== Laws ==
 11. flip (flip x) = x
 12. rot (flip (rot x)) = flip x
 13. above (flip x) (flip y) = flip (above x y)
 14. over (flip x) (flip y) = flip (over x y)

== Functions ==
rot45 :: Drawing -> Drawing

== Laws ==
 15. rot45 (flip (rot x)) = flip (rot45 x)
 16. over (rot45 x) (rot45 y) = rot45 (over x y)
 17. rot45 (rot (rot45 (rot (flip x)))) =
```

```
       rot (flip (rot45 (rot (rot45 x))))
 18. rot (rot (rot45 (rot (rot45 (flip x))))) =
       flip (rot (rot45 (rot (rot45 (rot x)))))
 19. rot (rot (rot45 (rot (rot45 (rot x))))) =
       flip (rot (rot45 (rot (rot45 (flip x)))))
 20. rot45 (rot (rot45 (rot45 (rot (rot x))))) =
       rot (rot45 (rot (rot45 (rot (rot45 x)))))
 21. rot45 (rot45 (rot45 (rot45 (rot (flip x))))) =
       rot (flip (rot45 (rot45 (rot45 (rot45 x)))))

== Functions ==
blank :: Drawing

== Laws ==
 22. flip blank = blank
 23. rot blank = blank
 24. rot45 blank = blank
 25. over x blank = x
 26. above blank blank = blank
 27. above blank (rot45 (rot (rot x))) =
       rot (rot (above blank (rot45 x)))
 28. above (rot45 (rot45 (rot45 x))) blank =
       rot45 (rot45 (beside (rot45 x) blank))
 29. beside (rot45 (rot45 (rot45 x))) blank =
       rot45 (rot45 (above (rot45 x) blank))
 30. rot (rot (rot45 (rot (rot45 x)))) = above blank (beside blank x)
 31. rot (rot45 (rot (rot (rot45 x)))) = above (beside x blank) blank
 32. above blank (rot45 (rot45 (rot (rot x)))) =
       rot (beside blank (rot45 (rot (rot45 x))))
 33. above blank (rot45 (rot45 (rot (rot45 x)))) =
       rot45 (rot (rot45 (above (rot45 x) blank)))
 34. above (rot45 (flip (rot45 (rot45 x)))) blank =
       rot45 (flip (rot45 (above blank (rot45 x))))
 35. above (rot45 (rot (rot45 (rot45 x)))) blank =
       rot45 (rot (rot45 (above blank (rot45 x))))
 36. above (rot45 (rot45 (flip (rot45 x)))) blank =
       rot45 (rot45 (flip (beside blank (rot45 x))))
 37. above (rot45 (rot45 (rot (rot45 x)))) blank =
       rot45 (rot45 (rot (above (rot45 x) blank)))
 38. beside (rot45 (flip (rot45 (rot45 x)))) blank =
       rot45 (flip (rot45 (beside blank (rot45 x))))
 39. beside (rot45 (rot (rot45 (rot45 x)))) blank =
       rot45 (rot (rot45 (beside blank (rot45 x))))
 40. beside (rot45 (rot45 (flip (rot45 x)))) blank =
```

```
      rot45 (rot45 (flip (above (rot45 x) blank)))
41. beside (rot45 (rot45 (rot (rot45 x)))) blank =
      rot45 (rot45 (rot (beside blank (rot45 x))))
42. flip (rot45 (rot (rot45 (rot45 (flip x))))) =
      above blank (beside blank (rot45 (rot x)))
43. flip (rot45 (rot (rot45 (rot45 (rot x))))) =
      above blank (beside blank (rot45 (flip x)))
44. rot45 (rot (rot45 (flip (rot45 (rot x))))) =
      above blank (beside (rot45 (flip x)) blank)
45. rot45 (rot (rot45 (rot45 (rot (flip x))))) =
      flip (above blank (beside blank (rot45 x)))
```