EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MŠMT
MINISTRY OF EDUCATION,
YOUTH AND SPORTS

CTU
CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

SEMINAR CIIRC CTU

Czech Institute of Informatics, Robotics, and Cybernetics
Czech Technical University in Prague

Lecturer: Moa Johannson
Title: Exploring Mathematical Conjecturing with Large Language Models
Project name: Artificial Intelligence and Reasoning
Project Registration No: CZ.02.1.01/0.0/0.0/15_003/0000466
Venue: Room B-571, Jugoslávských partyzánů 1580/3, Praha 6
Date: 18. 4. 2023, 10:00 – 11:00

Exploring Mathematical Conjecturing with Large Language Models

EuroProofNet Workshop on Practical Aspects of Machine Learning in Theorem Proving

Moa Johansson, joint work with Nick Smallbone. Prague 18 April 2023

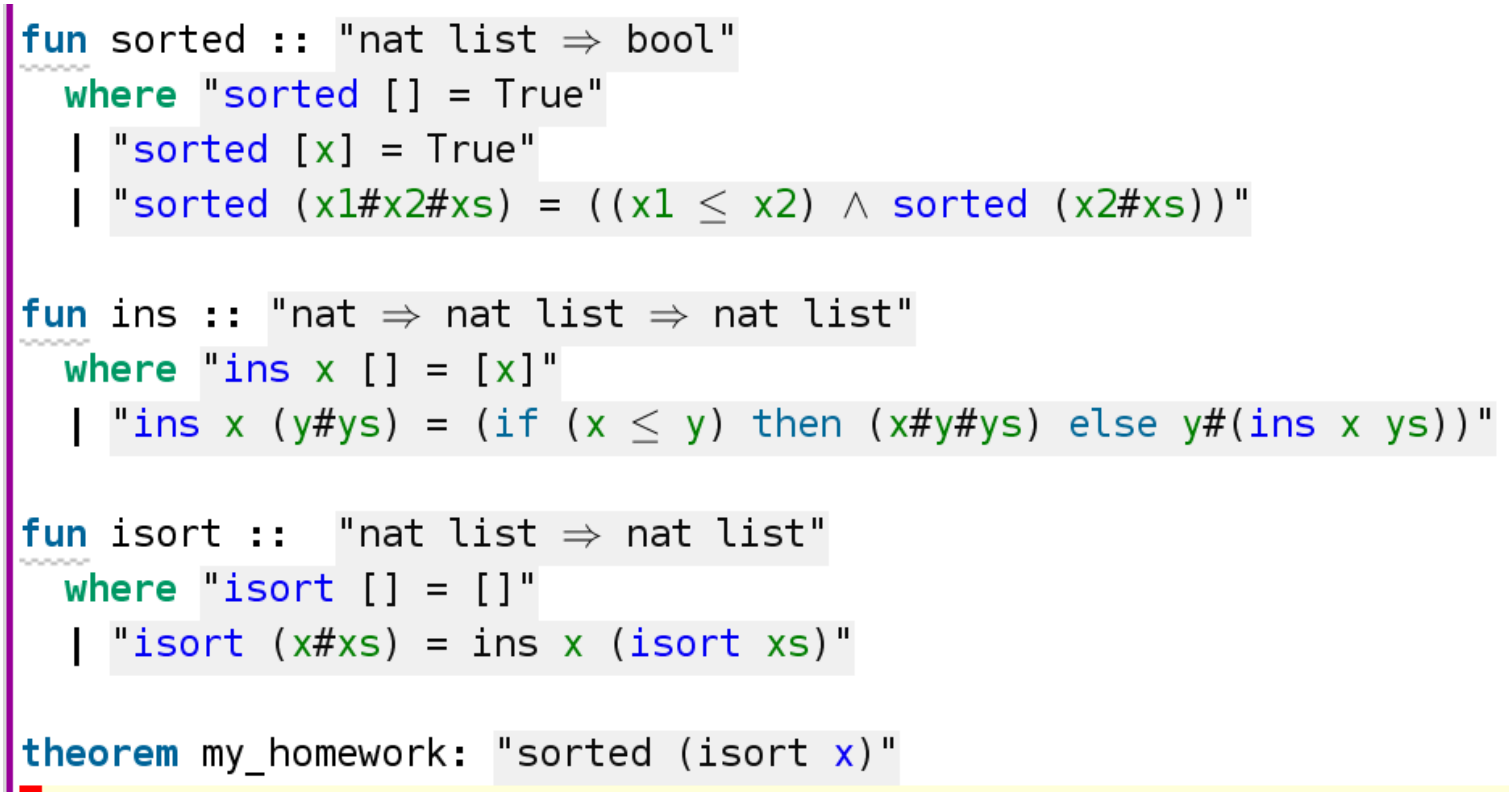
CHALMERS

Exploring and conjecturing

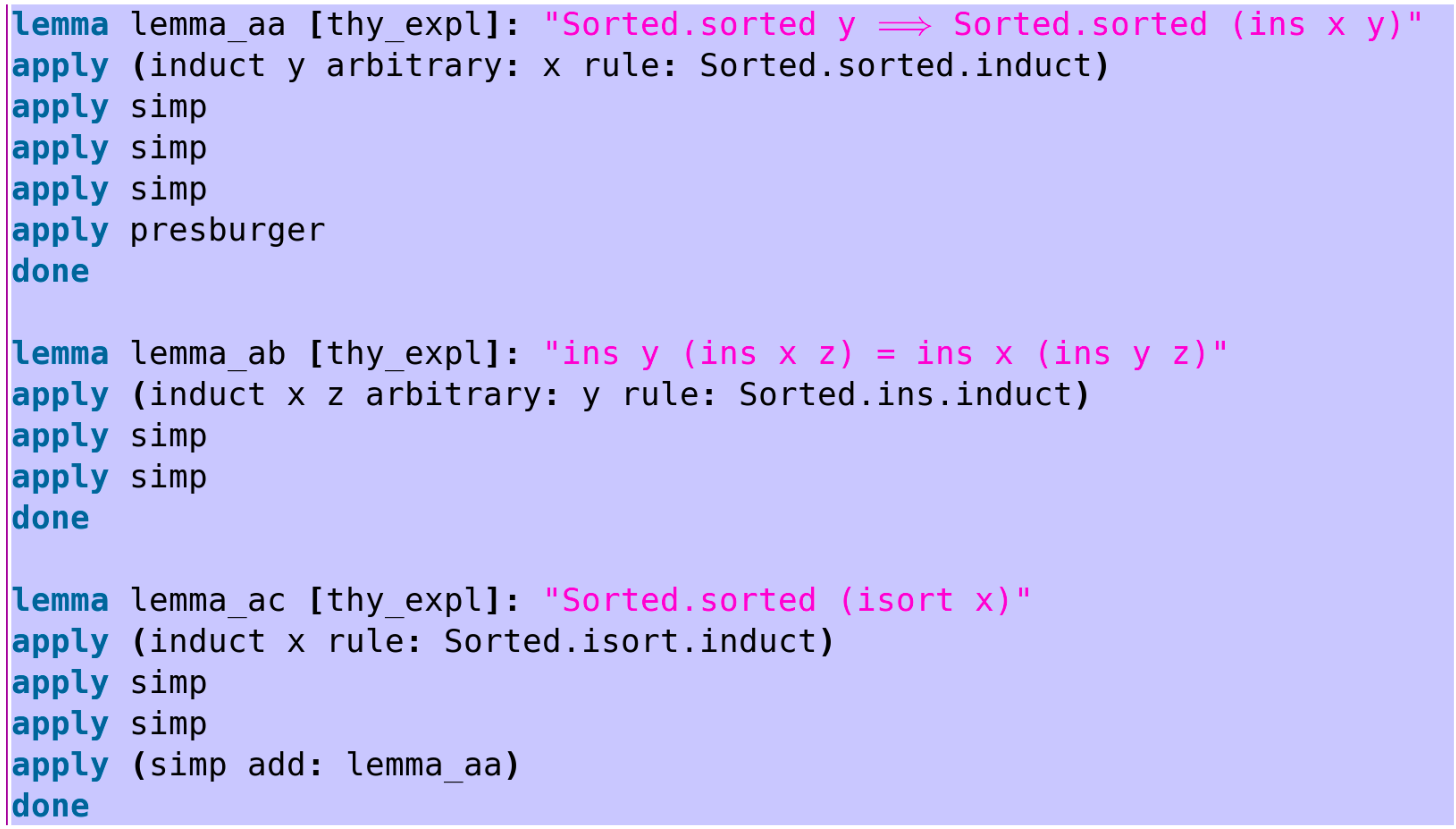
- **Theory Exploration:** invent conjectures about a mathematical theory.
  - New, Interesting, Non-trivial.
- Long **history of symbolic methods**/systems:
  - AM (Lenat 1976), Grafitti (Fajtlowicz 1988), HR (Colton 2000), MATHsAiD (McCasland 2010).
  - Our work:
    - Lemmas for inductive proofs: IsaCoSy, Hipster.
    - Properties about functional programs: QuickSpec.
- Recently also **neural methods**:
  - Mizar (Urban & Jakubův 2020), HOL Light (Rabe et al. 2021)

Example: Hipster

Symbolic theory exploration in Isabelle/HOL



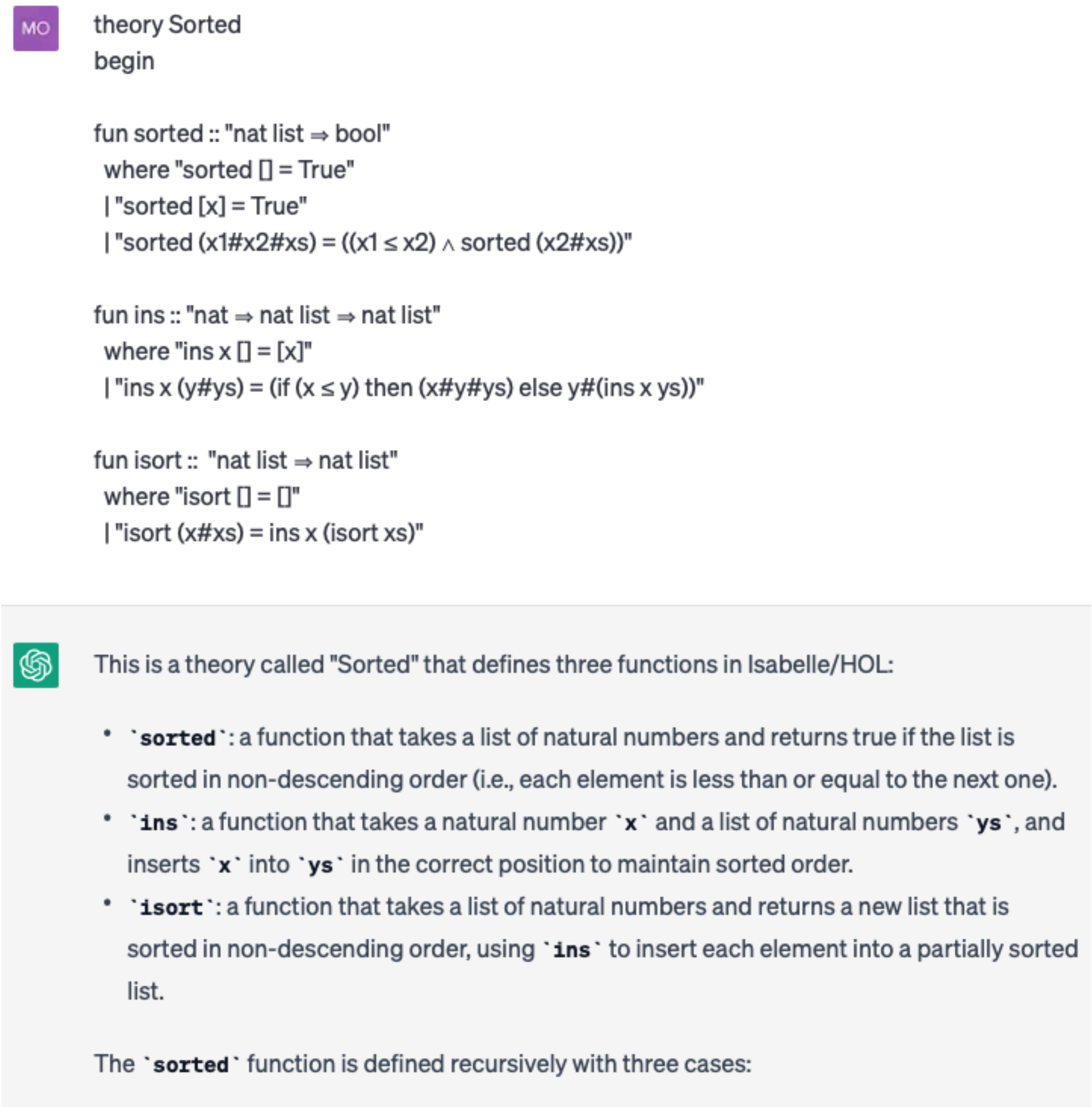
```
fun sorted :: "nat list ⇒ bool"
  where "sorted [] = True"
  | "sorted [x] = True"
  | "sorted (x1#x2#xs) = ((x1 ≤ x2) ∧ sorted (x2#xs))"

fun ins :: "nat ⇒ nat list ⇒ nat list"
  where "ins x [] = [x]"
  | "ins x (y#ys) = (if (x ≤ y) then (x#y#ys) else y#(ins x ys))"

fun isort ::  "nat list ⇒ nat list"
  where "isort [] = []"
  | "isort (x#xs) = ins x (isort xs)"

theorem my_homework: "sorted (isort x)"
```



```
lemma lemma_aa [thy_expl]: "Sorted.sorted y ⟹ Sorted.sorted (ins x y)"
apply (induct y arbitrary: x rule: Sorted.sorted.induct)
apply simp
apply simp
apply simp
apply presburger
done

lemma lemma_ab [thy_expl]: "ins y (ins x z) = ins x (ins y z)"
apply (induct x z arbitrary: y rule: Sorted.ins.induct)
apply simp
apply simp
done

lemma lemma_ac [thy_expl]: "Sorted.sorted (isort x)"
apply (induct x rule: Sorted.isort.induct)
apply simp
apply simp
apply (simp add: lemma_aa)
done
```

Can ChatGPT generate lemmas zero shot?

ChatGPT about common list functions

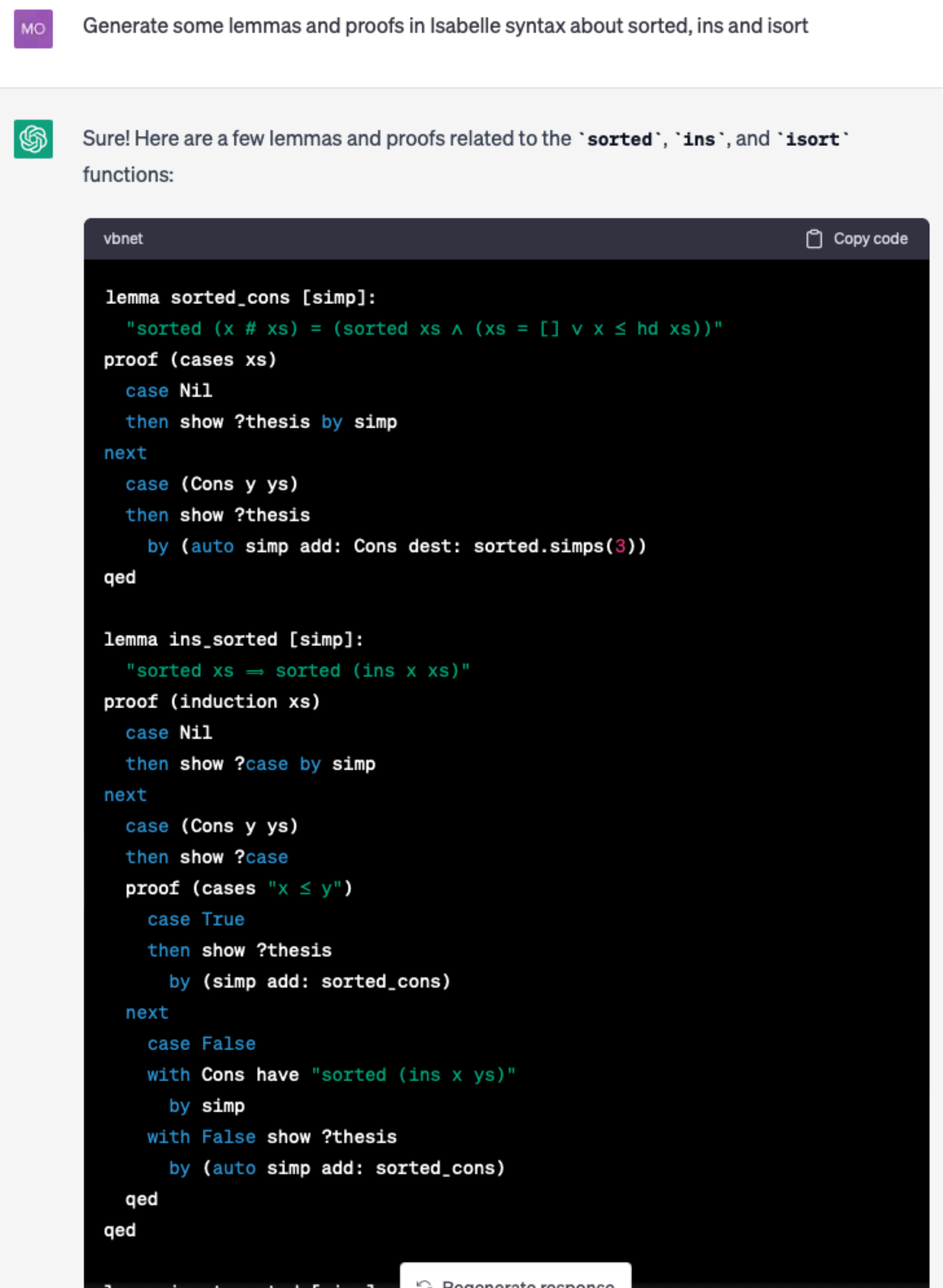


```
theory Sorted
begin

fun sorted :: "nat list ⇒ bool"
 where "sorted [] = True"
 | "sorted [x] = True"
 | "sorted (x1#x2#xs) = ((x1 ≤ x2) ∧ sorted (x2#xs))"

fun ins :: "nat ⇒ nat list ⇒ nat list"
 where "ins x [] = [x]"
 | "ins x (y#ys) = (if (x ≤ y) then (x#y#ys) else y#(ins x ys))"

fun isort :: "nat list ⇒ nat list"
 where "isort [] = []"
 | "isort (x#xs) = ins x (isort xs)"
```

This is a theory called "Sorted" that defines three functions in Isabelle/HOL:

- `sorted`: a function that takes a list of natural numbers and returns true if the list is sorted in non-descending order (i.e., each element is less than or equal to the next one).
- `ins`: a function that takes a natural number `x` and a list of natural numbers `ys`, and inserts `x` into `ys` in the correct position to maintain sorted order.
- `isort`: a function that takes a list of natural numbers and returns a new list that is sorted in non-descending order, using `ins` to insert each element into a partially sorted list.

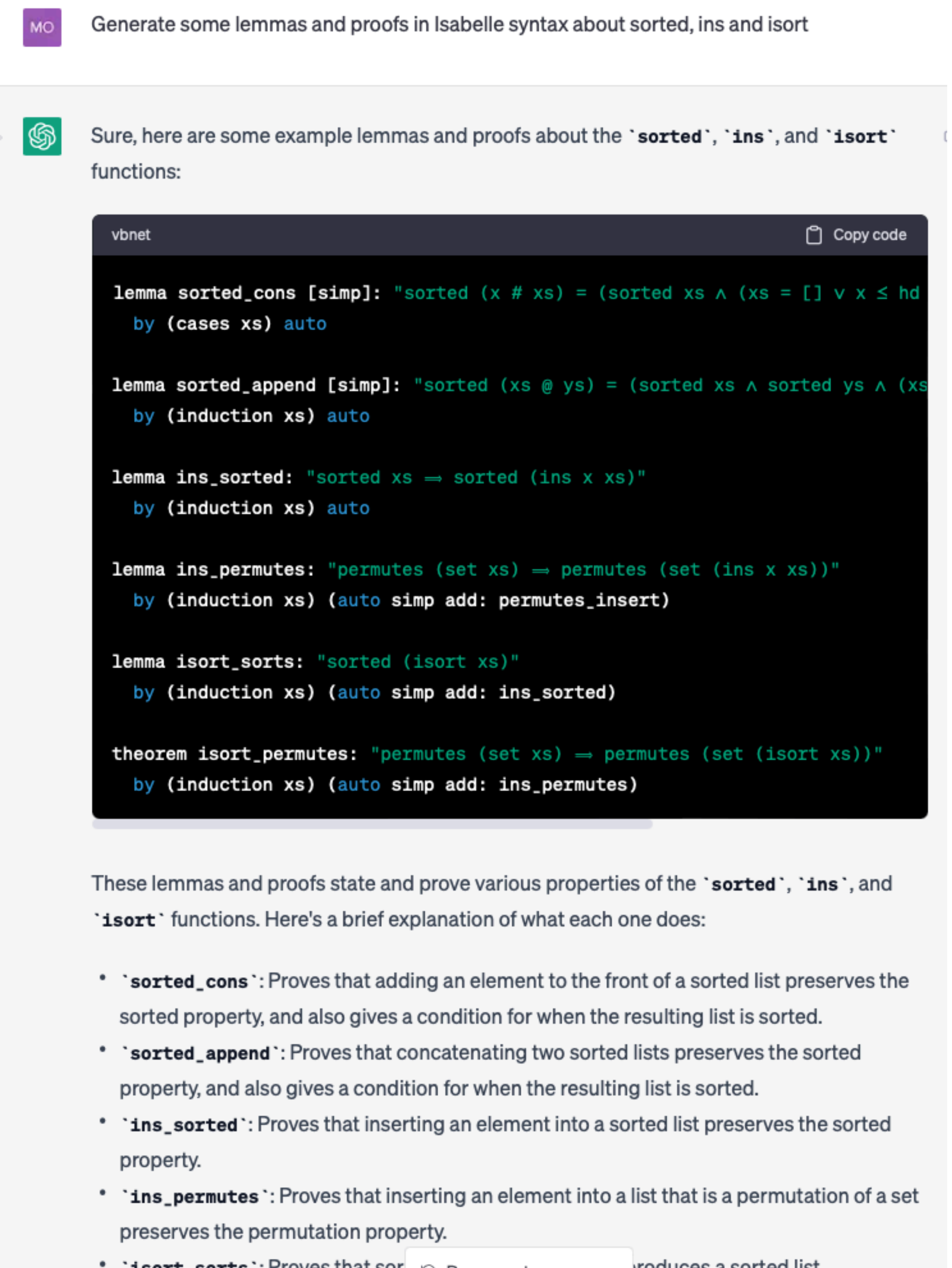
The `sorted` function is defined recursively with three cases:



Generate some lemmas and proofs in Isabelle syntax about sorted, ins and isort

Sure! Here are a few lemmas and proofs related to the `sorted`, `ins`, and `isort` functions:

```
lemma sorted_cons [simp]:
  "sorted (x # xs) = (sorted xs ∧ (xs = [] ∨ x ≤ hd xs))"
proof (cases xs)
  case Nil
  then show ?thesis by simp
next
  case (Cons y ys)
  then show ?thesis
    by (auto simp add: Cons dest: sorted.simps(3))
qed

lemma ins_sorted [simp]:
  "sorted xs ⟹ sorted (ins x xs)"
proof (induction xs)
  case Nil
  then show ?case by simp
next
  case (Cons y ys)
  then show ?case
  proof (cases "x ≤ y")
    case True
    then show ?thesis
      by (simp add: sorted_cons)
  next
    case False
    with Cons have "sorted (ins x ys)"
      by simp
    with False show ?thesis
      by (auto simp add: sorted_cons)
  qed
qed
```



Generate some lemmas and proofs in Isabelle syntax about sorted, ins and isort

Sure, here are some example lemmas and proofs about the `sorted`, `ins`, and `isort` functions:

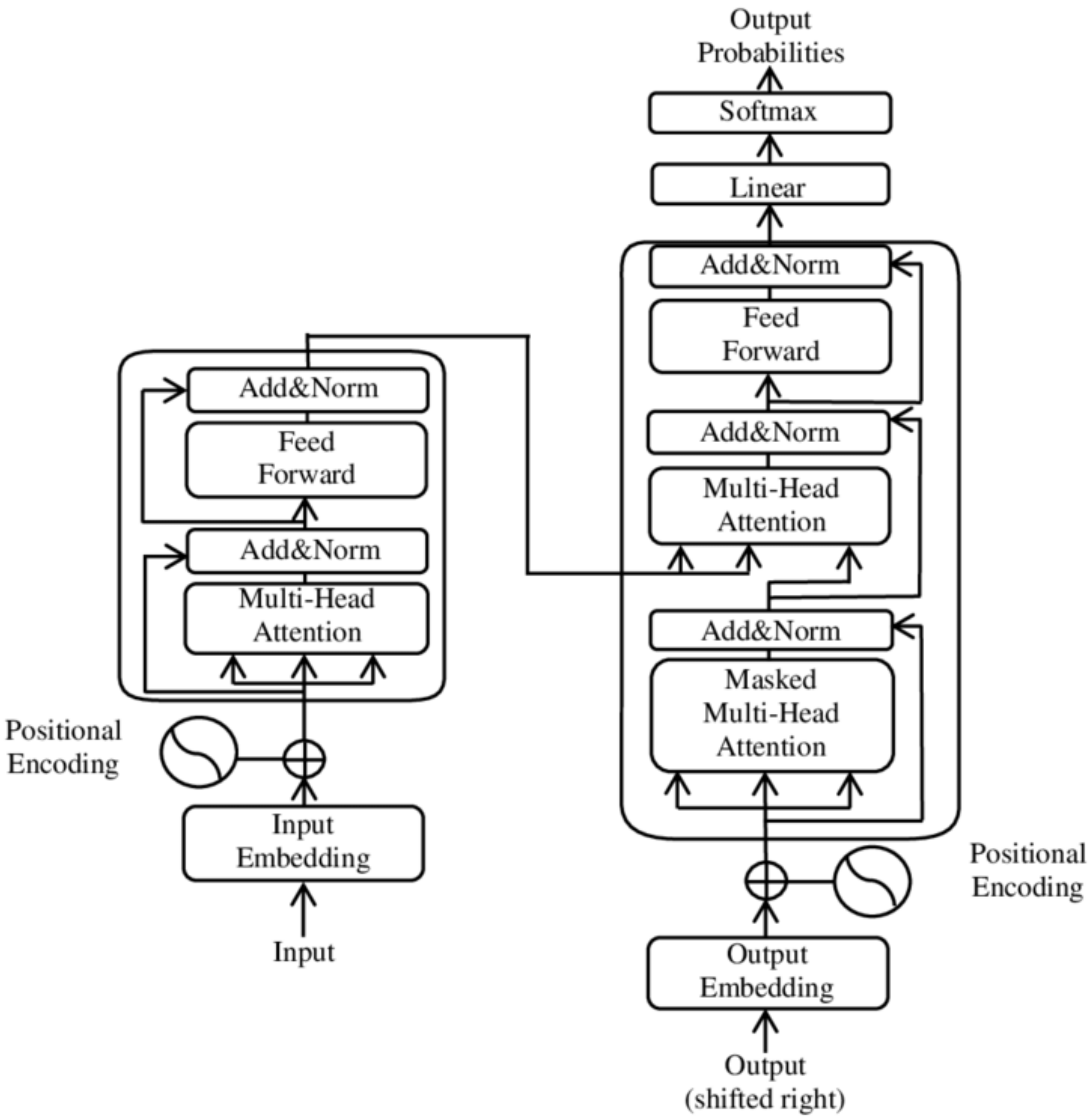
```
lemma sorted_cons [simp]: "sorted (x # xs) = (sorted xs ∧ (xs = [] ∨ x ≤ hd
  by (cases xs) auto

lemma sorted_append [simp]: "sorted (xs @ ys) = (sorted xs ∧ sorted ys ∧ (xs
  by (induction xs) auto

lemma ins_sorted: "sorted xs ⟹ sorted (ins x xs)"
  by (induction xs) auto

lemma ins_permutes: "permutes (set xs) ⟹ permutes (set (ins x xs))"
  by (induction xs) (auto simp add: permutes_insert)

lemma isort_sorts: "sorted (isort xs)"
  by (induction xs) (auto simp add: ins_sorted)

theorem isort_permutes: "permutes (set xs) ⟹ permutes (set (isort xs))"
  by (induction xs) (auto simp add: ins_permutes)
```

These lemmas and proofs state and prove various properties of the `sorted`, `ins`, and `isort` functions. Here's a brief explanation of what each one does:

- `sorted_cons`: Proves that adding an element to the front of a sorted list preserves the sorted property, and also gives a condition for when the resulting list is sorted.
- `sorted_append`: Proves that concatenating two sorted lists preserves the sorted property, and also gives a condition for when the resulting list is sorted.
- `ins_sorted`: Proves that inserting an element into a sorted list preserves the sorted property.
- `ins_permutes`: Proves that inserting an element into a list that is a permutation of a set preserves the permutation property.
- `isort_sorts`: Proves that sor roduces a sorted list

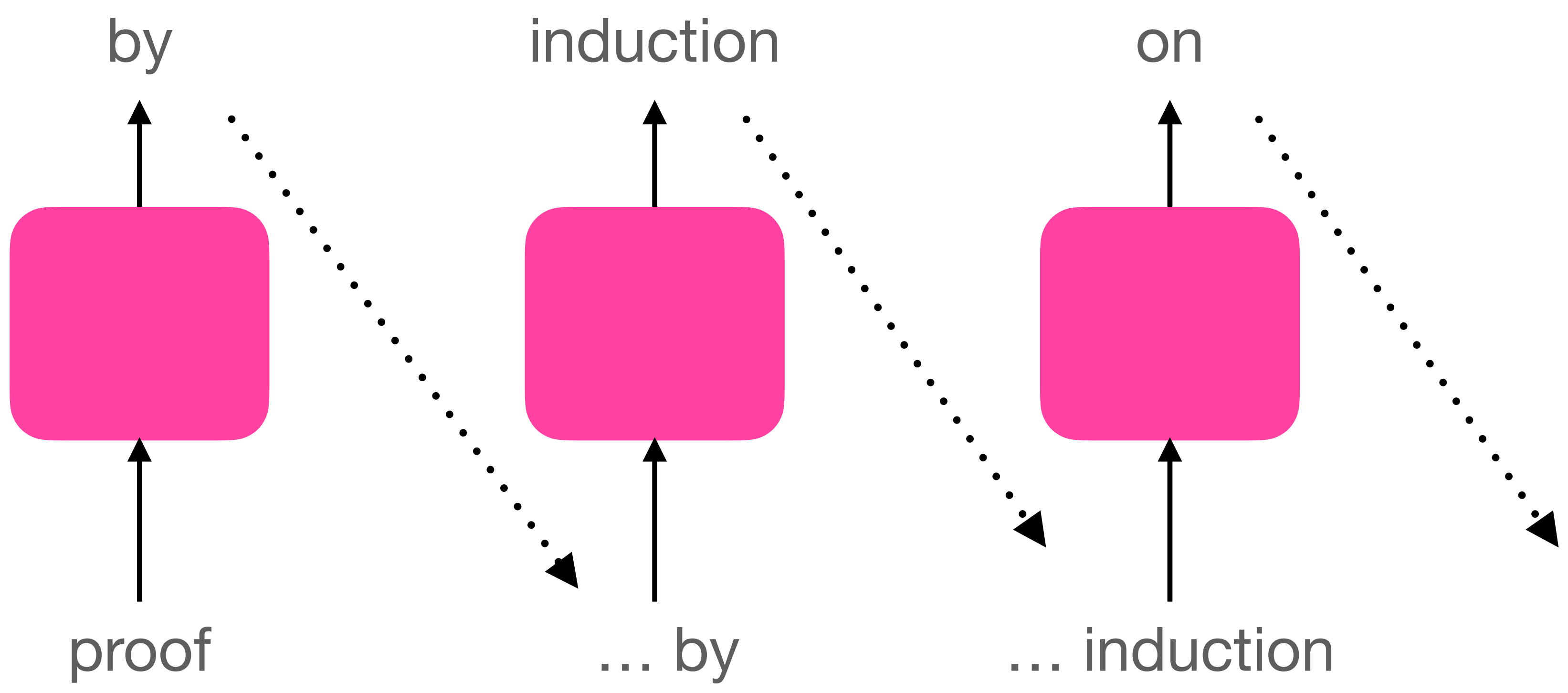
What is a Language Model really?

- A **probability distribution** over a text X:
  - $P(x_1, x_2, \ldots, x_n) = P(x_1) * P(x_2 \mid x_1) * P(x_2 \mid x_1, x_2) * \ldots * P(x_n \mid x_1, \ldots x_{n-1})$
- Implemented as a (large) neural network.
- **Transformer** architecture - learns which parts of the text are most relevant (attention layer).
- **Autoregressive** - generates tokens of text that continues prompt.



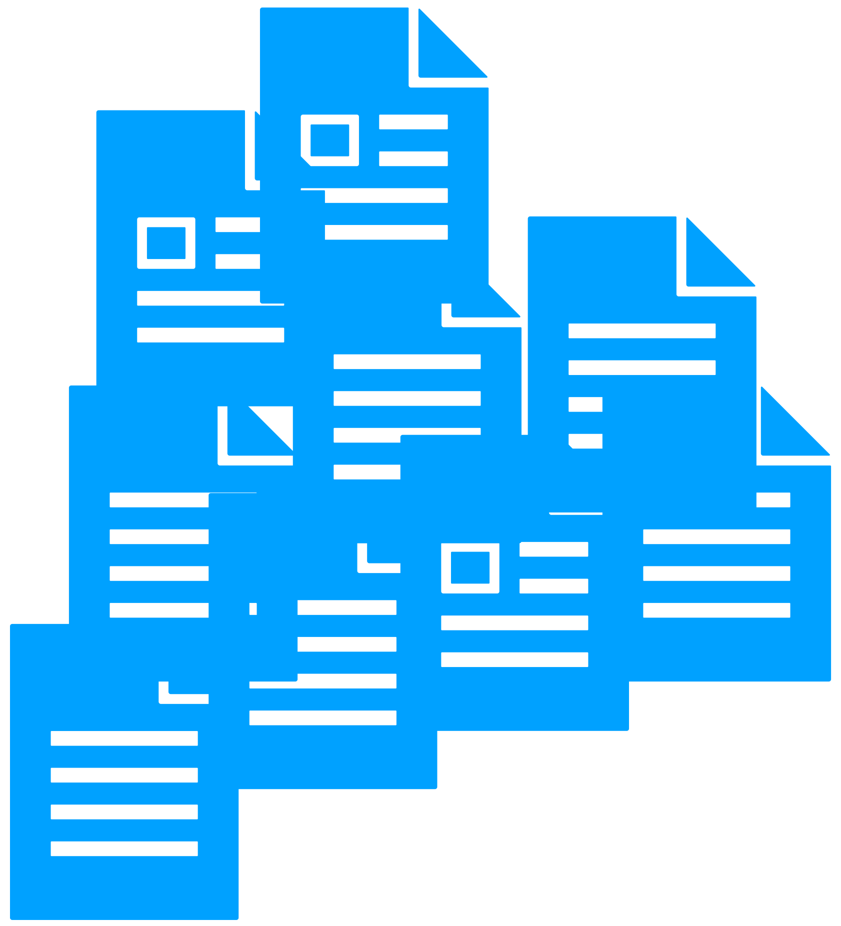
What is a Language Model really?

- A **probability distribution** over a text X:
  - $P(x_1, x_2, \ldots, x_n) = P(x_1) * P(x_2 \mid x_1) * P(x_2 \mid x_1, x_2) * \ldots * P(x_n \mid x_1, \ldots x_{n-1})$
- Implemented as a (large) neural network.
- **Transformer** architecture - learns which parts of the text are most relevant (attention layer).
- **Autoregressive** - generates tokens of text that continues prompt.



Training Data

- NB: We don't have the details of ChatGPT's exact training data. Proprietary!



LOTS of text from Internet, including code from GitHub, texts in various languages, Isabelle/HOL code…

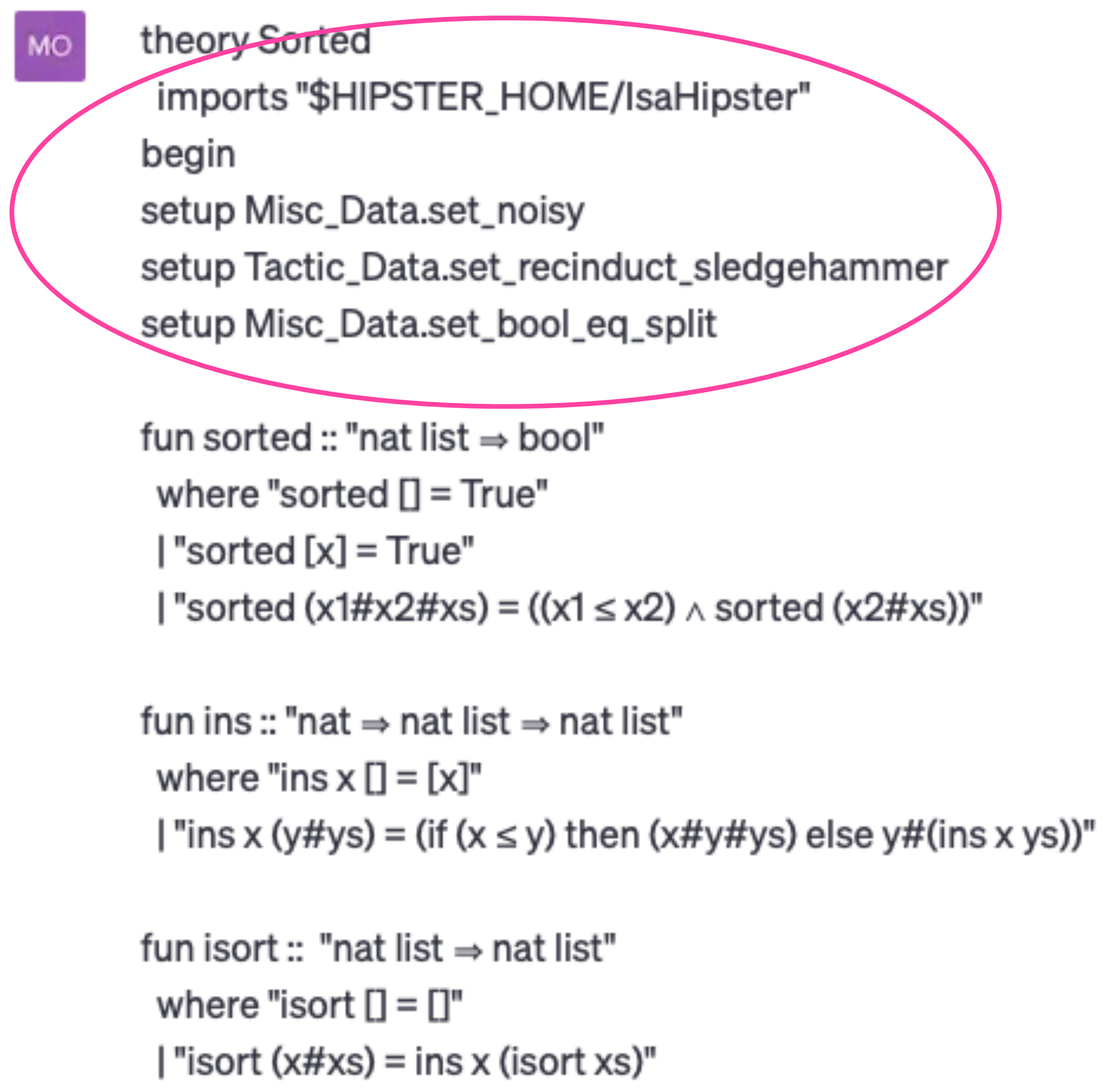
Instruction Fine Tuning,
Reinforcement Learning with Human Feedback (RLHF)

Often low paid human crowd sourced workers, users playing with free system.

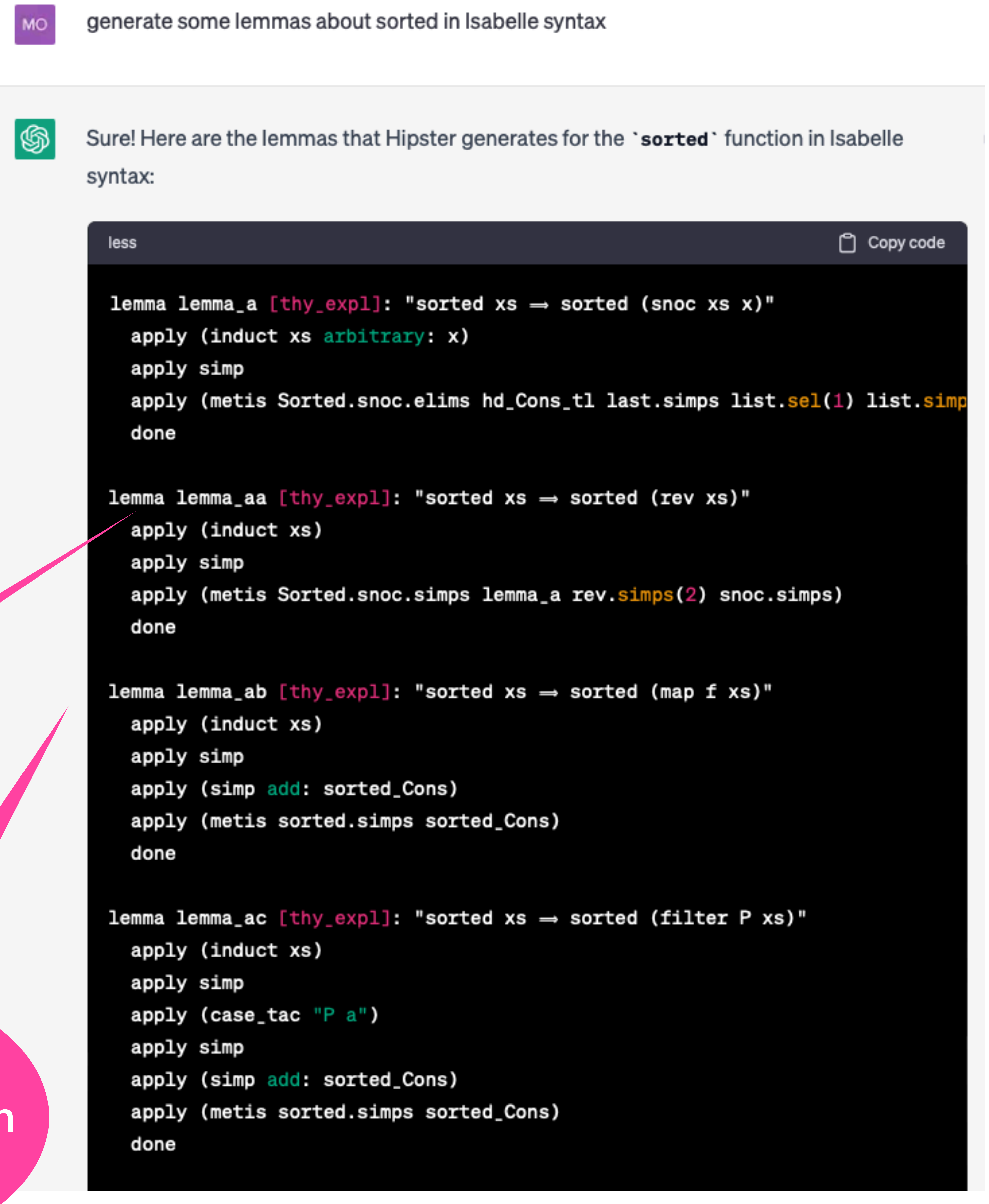
Dialogues with system

Do we need symbolic theory exploration systems anymore?

- Hallucinations not much of a problem - we check the results in the theorem prover afterwards.
- But how do we assess their capabilities on **truly novel** tasks?
    - List libraries and sorting are all over the internet…



```
theory Sorted
 imports "$HIPSTER_HOME/IsaHipster"
begin
setup Misc_Data.set_noisy
setup Tactic_Data.set_recinduct_sledgehammer
setup Misc_Data.set_bool_eq_split

fun sorted :: "nat list ⇒ bool"
 where "sorted [] = True"
 | "sorted [x] = True"
 | "sorted (x1#x2#xs) = ((x1 ≤ x2) ∧ sorted (x2#xs))"

fun ins :: "nat ⇒ nat list ⇒ nat list"
 where "ins x [] = [x]"
 | "ins x (y#ys) = (if (x ≤ y) then (x#y#ys) else y#(ins x ys))"

fun isort ::  "nat list ⇒ nat list"
 where "isort [] = []"
 | "isort (x#xs) = ins x (isort xs)"
```



generate some lemmas about sorted in Isabelle syntax

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

```
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

lemma lemma_ac [thy_expl]: "sorted xs ⟹ sorted (filter P xs)"
  apply (induct xs)
  apply simp
  apply (case_tac "P a")
  apply simp
  apply (simp add: sorted_Cons)
  apply (metis sorted.simps sorted_Cons)
  done
```

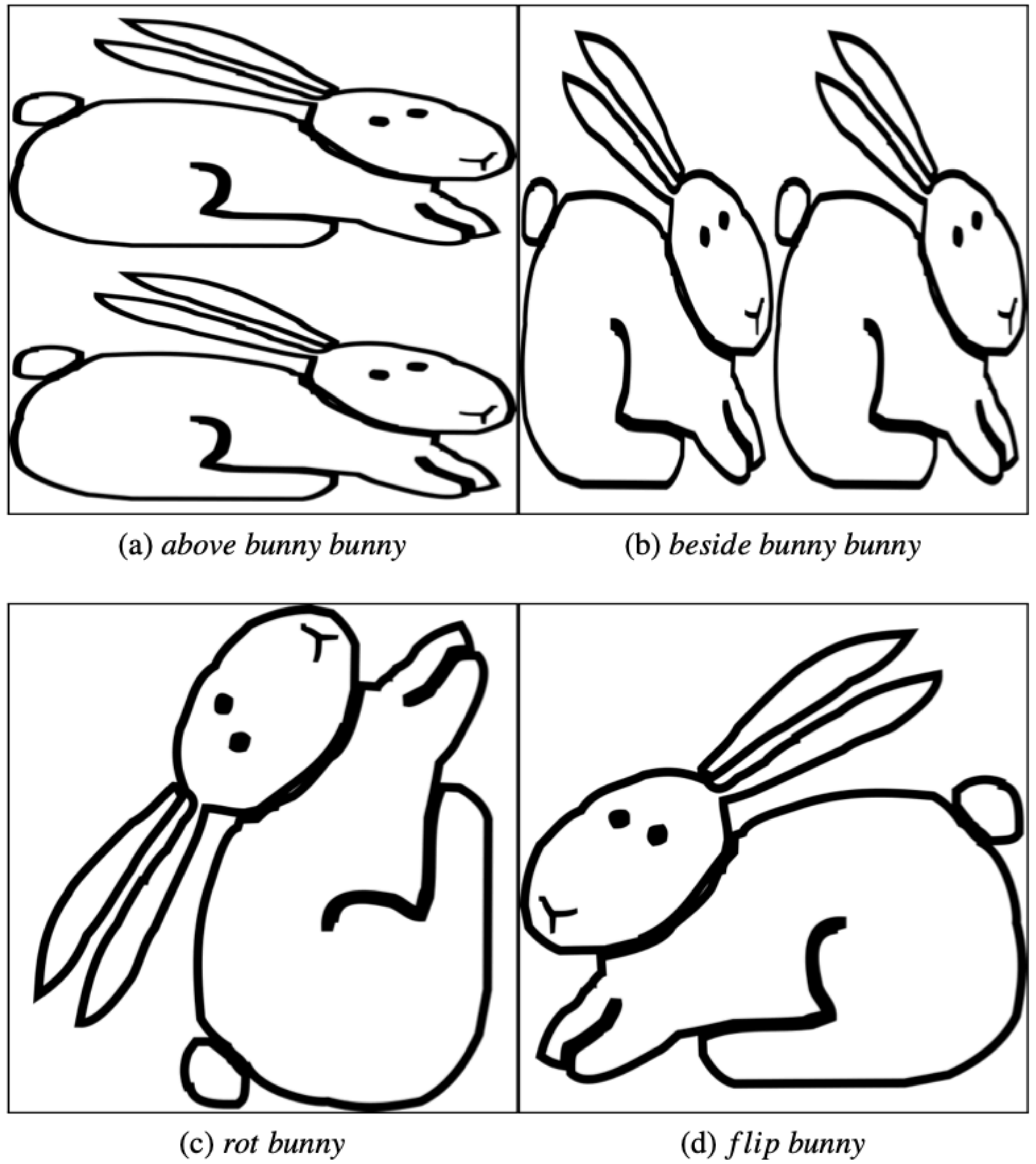
Hmm… imitates output format from Hipster…

All our benchmarks are on GitHub…

Another attempt

A Drawing library from Haskell in Isabelle/HOL

- Let's try something that at least is unlikely to exist in Isabelle.
- Caveat: This benchmark is however online as part of the QuickSpec benchmarks in Haskell.

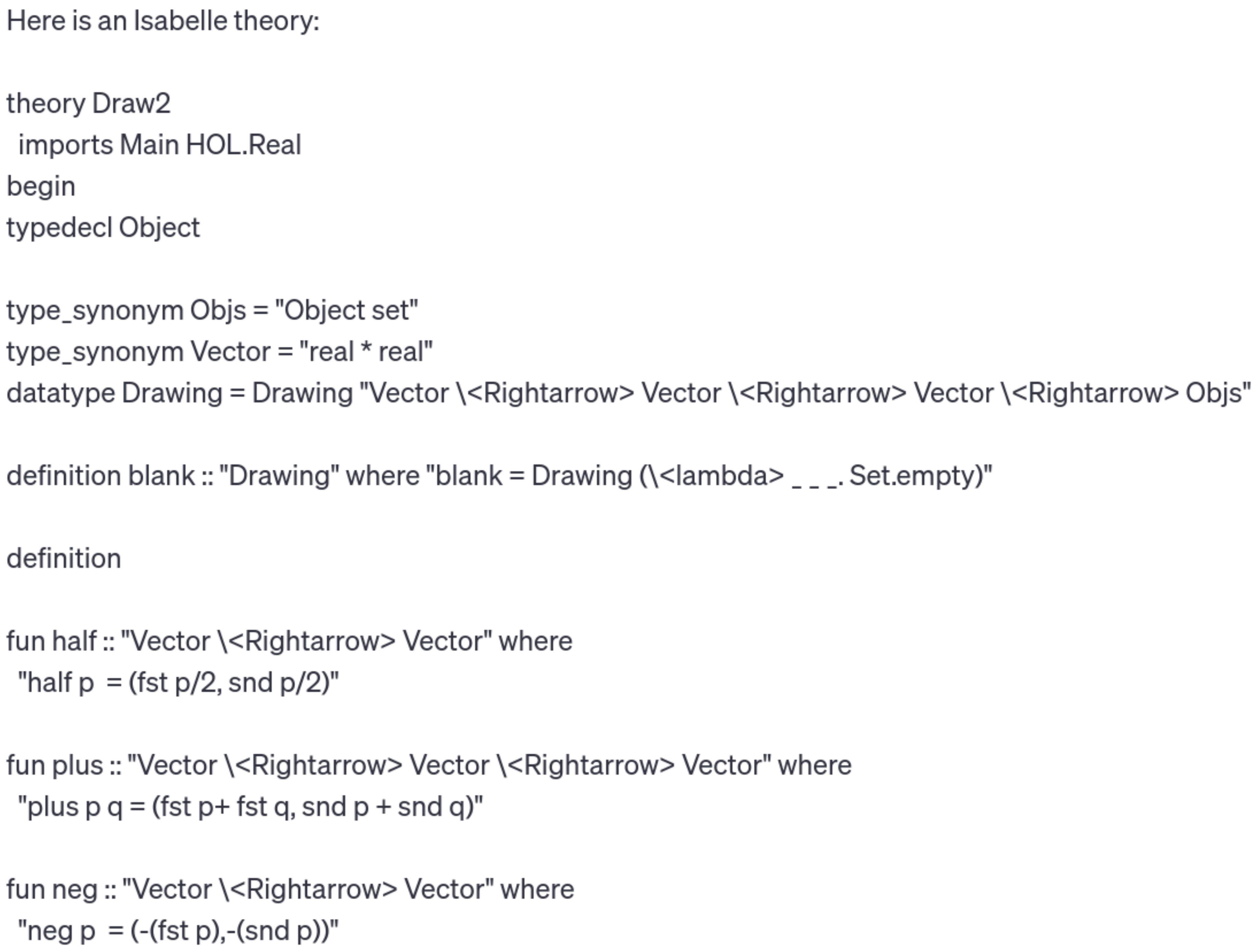


(a) *above bunny bunny* (b) *beside bunny bunny*

(c) *rot bunny* (d) *flip bunny*

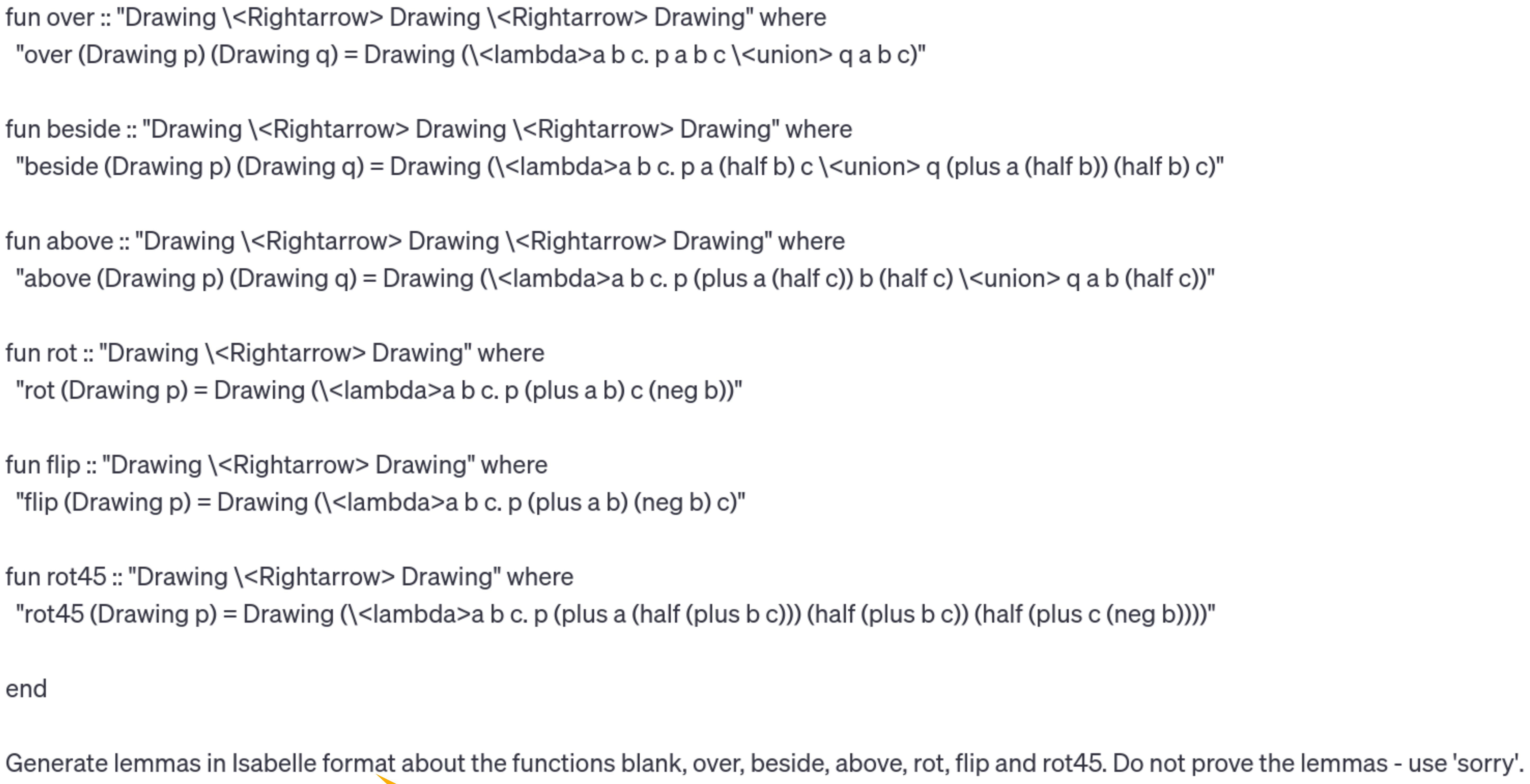
Fig. 6: Picture combinators and bunnies.

Prompt to ChatGPT:



Here is an Isabelle theory:

```
theory Draw2
  imports Main HOL.Real
begin
typedecl Object

type_synonym Objs = "Object set"
type_synonym Vector = "real * real"
datatype Drawing = Drawing "Vector \<Rightarrow> Vector \<Rightarrow> Vector \<Rightarrow> Objs"

definition blank :: "Drawing" where "blank = Drawing (\<lambda> _ _ _. Set.empty)"

definition

fun half :: "Vector \<Rightarrow> Vector" where
  "half p  = (fst p/2, snd p/2)"

fun plus :: "Vector \<Rightarrow> Vector \<Rightarrow> Vector" where
  "plus p q = (fst p+ fst q, snd p + snd q)"

fun neg :: "Vector \<Rightarrow> Vector" where
  "neg p  = (-(fst p),-(snd p))"
```



```
fun over :: "Drawing \<Rightarrow> Drawing \<Rightarrow> Drawing" where
  "over (Drawing p) (Drawing q) = Drawing (\<lambda>a b c. p a b c \<union> q a b c)"

fun beside :: "Drawing \<Rightarrow> Drawing \<Rightarrow> Drawing" where
  "beside (Drawing p) (Drawing q) = Drawing (\<lambda>a b c. p a (half b) c \<union> q (plus a (half b)) (half b) c)"

fun above :: "Drawing \<Rightarrow> Drawing \<Rightarrow> Drawing" where
  "above (Drawing p) (Drawing q) = Drawing (\<lambda>a b c. p (plus a (half c)) b (half c) \<union> q a b (half c))"

fun rot :: "Drawing \<Rightarrow> Drawing" where
  "rot (Drawing p) = Drawing (\<lambda>a b c. p (plus a b) c (neg b))"

fun flip :: "Drawing \<Rightarrow> Drawing" where
  "flip (Drawing p) = Drawing (\<lambda>a b c. p (plus a b) (neg b) c)"

fun rot45 :: "Drawing \<Rightarrow> Drawing" where
  "rot45 (Drawing p) = Drawing (\<lambda>a b c. p (plus a (half (plus b c))) (half (plus b c)) (half (plus c (neg b))))"

end
```

Generate lemmas in Isabelle format about the functions blank, over, beside, above, rot, flip and rot45. Do not prove the lemmas - use 'sorry'.

Generate lemmas in Isabelle format about the functions blank, over, beside, rot, flip and rot45.

Here are some lemmas about the functions blank, over, beside, above, rot, flip, and rot45 in Isabelle format. Note that these lemmas have not been proven and use 'sorry' as a placeholder for proofs.

```
theory Draw2Lemmas
  imports Draw2
begin

lemma blank_over_left: "over blank d = d"
  sorry

lemma blank_over_right: "over d blank = d"
  sorry

lemma over_associative: "over (over d1 d2) d3 = over d1 (over d2 d3)"
  sorry

lemma over_commute: "over d1 d2 = over d2 d1"
  sorry

lemma beside_blank_left: "beside blank d = d"
  sorry

lemma beside_blank_right: "beside d blank = d"
  sorry
```

```
lemma beside_associative: "beside (beside d1 d2) d3 = beside d1 (beside d2 d3)"
  sorry

lemma beside_commute: "beside d1 d2 = beside d2 d1"
  sorry

lemma above_blank_top: "above blank d = d"
  sorry

lemma above_blank_bottom: "above d blank = d"
  sorry

lemma above_associative: "above (above d1 d2) d3 = above d1 (above d2 d3)"
  sorry

lemma above_commute: "above d1 d2 = above d2 d1"
  sorry

lemma rot_rot: "rot (rot d) = d"
  sorry

lemma flip_flip: "flip (flip d) = d"
  sorry

lemma rot45_rot45_rot90: "rot45 (rot45 d) = rot d"
  sorry

lemma rot45_flip_rot45: "rot45 (flip (rot45 d)) = d"
  sorry
```

How did it do?

- Many conjectures were false (10/16).
  - Not huge problem - checked by theorem prover/counter example checker.
- Seems ChatGPT has **internalised some “templates”**
  - There is some identity element.
  - Binary functions are associative and commutative.
  - Unary functions are their own identity.
  - Binary functions distribute over one another (sometimes).
- **Different runs give different conjectures**.

Comparing to a symbolic system

QuickSpec

- QuickSpec consistently discovers lemmas **more systematically and predictably**.
- Which functions are associative, commutative etc.
  - `over x x = x` (not found by ChatGPT)
- How functions distribute over each other.
- Rotating 90 degrees four times is the identity function.
- Equivalent ways of laying out drawings in 2 x 2 grid, e.g.
  - `above (beside x y) (beside z w) = beside (above x z) (above y w)`

Comparing to a symbolic system

QuickSpec

- But: QuickSpec **fails to find conjectures outside its size limit**.
  - `rot45(rot45(rot45(rot45(rot45(rot45(rot45(rot45 x))))))) = x`
- Buggy definitions:
  - QuickSpec will miss or discover other properties.
  - ChatGPT may still suggest “intended” property, even though there is a bug in the function definition.

ChatGPT as a theory exploration system

Pros:

- **Names are informative**
  - Can sometimes add extra function that user hadn’t thought of.
- **Less restricted shapes of lemmas.**
  - Size, equations, implications…
- **Ability to recognise common patterns.**
  - C.f. RoughSpec (Einarsdottir et al. 2021)
- Can **suggest “correct" properties for buggy functions.**

ChatGPT as a theory exploration system

Cons:

- **Difficulty of fair evaluation.**
  - Hidden behind API, what’s in training data?
- **Unpredictable coverage.**
  - Stochastic: different runs give different results.
- **Hardware and energy consumption.**
  - QuickSpec happily runs on an old laptop… ChatGPT does not.
- Payment.
  - Commercial product, will theorem proving users really pay for this?

Final words

- Interesting complementary features to symbolic systems.
- But desirable with smaller, transparent system.
  - However, we had much better results with GPT-3.5 davinci and GPT-4. Smaller versions did not do well zero-shot.
- Smaller but carefully trained networks? Can they cope with syntax of multiple theorem proving systems?
- Neuro-symbolic approaches?