The Universal Prover

March 30, 2025

1 The problem

The *perfectly reliable automation of natural language deductive reasoning* is an important task. It would be a significant contribution to artificial general intelligence (AGI), and the AI system that achieved that automation could be incorporated into any other AI system as a natural language deductive reasoning module. It could do for deductive reasoning what ordinary calculators did for arithmetic.

At present, we have two kinds of AI systems for the automation of deductive reasoning:

(i) Our most advanced general-purpose AI systems are Generative AI (**GenAI**) systems like ChatGPT. They perform natural language reasoning tasks, but they are notoriously unreliable at them, and the underlying technology, which is a sophisticated form of predictive text, makes it impossible for them to ever achieve 100% reliability.

(ii) The automated theorem provers (known both in the industry and the research literature as **provers**) of Symbolic AI/Good Old-Fashioned AI (**GOFAI**) are 100% reliable, but they do not understand or produce output in natural languages, and they are extremely difficult to use. Even most professional mathematicians do not use them for this reason.[1]

Thus, any person or organization that wishes to benefit from the automation of deductive reasoning tasks *but* requires that automation to be 100% reliable faces an

[1] When mathematicians use theorem provers, they must first learn to code and then figure out how to express their problem in computer-friendly language. As mathematician Michael Harris of Columbia University put it: "By the time I've reframed my question into a form that could fit into this technology, I would have solved the problem myself" (quoted in *Quanta Magazine*, August 27, 2020).

impossible dilemma: a choice between unreliability (which is ruled out) and virtual unusability. This, in practice, makes it impossible for the vast majority of such potential end users to avail themselves of the extremely powerful computational resources that are already in principle available for their purposes.

2 An analysis of the problem

The basic reason why the perfectly reliable automation of natural language deductive reasoning cannot be achieved by means of the technology underlying our most advanced general-purpose AI systems (at present, in 2025) is very simple. It is easiest to understand by considering, by way of analogy, the problem of automating chess playing. A GenAI chess engine would, at each point, make the move that, in its judgment, is most likely to lead to a win (or, barring that, a draw) given both the moves made in the game so far and a vast data set of games of chess, perhaps including ones in which it attempts to play against itself, *but with no knowledge of the rules of chess*. Given enough data and suitable weights for its learning algorithm, the chess engine might become very good at *simulating* playing chess, and it might even typically perform very well against the casual human chess player, but it could be relied upon to occasionally make not just a bad move but an *illegal* move—say, attempting to castle a second time when it observes the pieces in the characteristic pre-castling configuration—because, by its very design, it is incapable of assigning a probability of zero to a move that is actually illegal (or to any move). Needless to say, no one ever seriously proposed building a chess engine that works like this. Yet, difficult as it may be to believe, this is exactly how our most advanced natural language reasoning engines work: they do not know *any* rules of deductive logic; rather, they simulate deductive reasoning in the way the chess engine imagined above simulates chess-playing, with predictable results.

This is why GenAI, as an approach to the automation of natural language deductive reasoning, is a dead end (and a very expensive one). GenAI reasoning engines, like our imaginary GenAI chess engines, can be expected to get better if we continue to expend more resources training them, but (i) with diminishing returns, and—more importantly—(ii) no matter the amount of resources we throw at training them, we can expect them to occasionally lapse into total incoherence ("hallucinations") in ways in which an intelligent human would never do. In contrast, a well-designed GOFAI prover not only never makes mistakes in its reasoning but we can *prove* that it will never make a mistake in its reasoning.

GOFAI, however, is also a dead end as an approach to the automation of natural language deductive reasoning. This is not because the rules of deductive logic are

too complex for humans to write down—on the contrary, they are *very simple*.[2] Nor is it because natural language is too complex for humans to be able to write down algorithms for the application of those rules to natural language—doing so is *merely tedious*; the main obstacle there is boredom, and that obstacle is easy to overcome with the tried and tested method of the cash incentive.[3] The problem, rather, is that a pure GOFAI system for automating natural language deductive reasoning will not *do* anything. It will consist of several grammars (equivalently: parsers), translations, and a proof-checker (a parser for proofs). Humans do not know how to write down good algorithms for *generating* the proofs to be checked by the proof-checker or for *formalization*, which is essentially a matter of finding the most likely parse trees for ambiguous sentences/texts.

3 The solution: UP

3.1 Motivation and overview

In a way, the solution is obvious. It's the same as the solution to the problem of automating chess-playing: our best chess engines have a GOFAI component that ensures that an illegal move is never made and a GenAI component that searches the space of game trees consisting of only legal moves for the best move. (This is how DeepMind's AlphaZero works.) Because the desiderata for a deductive reasoning engine are a little different from those for a chess engine, we want something slightly different, however—what we want is a *modular* GOFAI/GenAI hybrid with the following properties:

(i) The system has GOFAI modules that perform the tasks that can be performed by GOFAI and GenAI modules for the remaining tasks. (The system will have this much in common with a good chess engine.)

(ii) The tasks are divided between GOFAI and GenAI in a way that does not compromise the *transparency*, *reliability*, and *control* that come with good GOFAI automation: *Not only* must the system be designed so that it is incapable of making an illegal move (reliability), *but also* so that we can check its work (transparency) and intervene in its process as needed to ensure that it is working on the problem we intended to direct it to work on (control).

[2] All of the rules of deductive logic can be reduced to ten or so rules for manipulating formulae constructed out of variables and the inclusion symbol "$\subseteq_\sigma$" by function application and function abstraction: see §4.2.2.

[3] Which is not even needed when you have academics employed by universities working on it; some of them actually enjoy this kind of work.

The Universal Prover (UP) is the GOFAI component of this modular AI system. It consists of several GOFAI modules assembled into a structure with "gaps" for GenAI modules that, when filled, will produce an AI system with properties §3(i) and §3(ii). The gaps are so designed that they can be filled with off-the-shelf GenAI components, requiring minimal customization/fine-tuning, so that someone who wants to use UP to build their own natural language deductive reasoning module can do so without giving the matter much thought. (There will also be a POC version with all the gaps filled by us.) On the other hand, for someone with very ambitious plans for a natural language deductive reasoning module, the sky is the limit: they are free to use whatever resources are available to them for training purpose-made GenAI components to fill the gaps.

The *power* of the resulting AI system will be a function of the GenAI modules that fill the gaps: e.g., one can assign the task of searching for proofs to a so-so open source prover, to the neural prover underlying DeepMind's AlphaProof or to something even more powerful, or to anything in between. What UP contributes is not power but a design that *extends a guarantee of perfect reliability as well as*—optionally—*-transparency and manipulability to the entire AI system* constructed by filling its gaps.

Remark 1. *Transparency and manipulability are optional, in the sense that there is nothing to stop someone wishing to do so from enclosing the resulting system in a black box that "hides the work" and only shows the results.*

Such applications of UP may be desirable for some purposes. For example, it may be impractical to include in the customer user interface of an automatic cashier/checkout that incorporates UP an option for inspecting its chains of reasoning. In-N-Out, for example, can be expected not to want its customers to spend any time inspecting the chain of reasoning that led its checkout to recommend adding a strawberry shake to an order of a Double-Double and French fries, or inspecting the parse tree it assigns to the customer's request for "a Double-Double and French fries".

On the other hand, In-N-Out will presumably find value in a different user interface (for employees, not customers) that does show the work in case of (e.g.) a disputed transaction.

3.2 The task and its subtasks

The *basic kind of task* that a UP-based natural language deductive reasoning module is meant to perform is that of determining whether a given sentence in given natural language ("the conclusion") deductively follows from a given set of sentences of that same language ("the premises"). (For purposes of illustration, in what follows

we will assume that the natural language is English.) This includes the problem of determining whether a given set of sentences is deductively *consistent*, since a deductively consistent set of sentences is one from which no contradiction (a pair of a sentence and its negation) follows deductively.

We break up the task into *seven* subtasks: *four* that are best handled by GOFAI and *three* that are best handled by GenAI:

(i) **Disambiguation:** Disambiguating ordinary English input into sentences of a disambiguated version of English, which we will call *English as a Formal Language* (*EFL*). Since this is a matter of guessing what disambiguation the user or author of the sentences (if distinct from the user) would accept, it is a task for GenAI (such as an LLM).

(ii) **EFL/HOL translation:** Translating the problem from EFL into an enriched version of the language of higher-order logic (HOL)—a GOFAI task.

(iii) **Proof search:** To find a solution, specifically a proof in UP's formal system—a task for GenAI (such as the neural prover that is at the core of DeepMind's AlphaProof).

(iv) **Proof-checking:** To check the solution(s) offered by the GenAI prover, i.e., proof-checking—a GOFAI task.

(v) **HOL/EFL translation:** To translate the solution(s) from HOL into from EFL—a GOFAI task.

(vi) **Syntactic sugaring:** To rewrite the EFL solution into a form that is easy for a human reader to understand, a process known in programming as *syntactic sugaring*. (This is a form of *EFL-to-EFL paraphrase*, since EFL already includes the sugary variants of all of its sentences.) By definition, syntactic sugaring includes the use of abbreviating conventions (if used to aid readability), such as conventions for eliminating parentheses, subscripts, superscripts, etc. This includes the abbreviation of proofs by the omission of obvious steps. However, sugaring is not always abbreviation: it can also increase the character count (e.g., by replacing "$\forall_t$" with "every property of propositions" or by adding signposts like "therefore" and "which contradicts our hypothesis" to proofs) as well as replace strings by two-dimensional representations: e.g., rewriting exponents:

$$((\hat{}\omega)((\hat{}\omega)((\hat{}\omega)\omega)))$$

as iterated superscripts:

$$\omega^{\omega^{\omega^{\omega}}},$$

which are certainly easier to read, or using different sizes of parentheses to make it easier to count parentheses and to tell which right parentheses go with which left ones, as in:[4]

$$\Big(\big(p \to (q = \top)\big) \wedge \big(\neg p \to (q = \bot)\big)\Big)$$

Syntactic sugaring is a GenAI task, presumably for an *output LLM*.

(vii) **Sugar-checking:** Unlike the work of the input LLM, the work of the output LLM (or similar, at the previous step), can be and will be mechanically verified. This *can* be done because there are explicit conventions[5] according to which syntactic sugaring proceeds. For example, one such convention is that omitted parentheses around arrows group to the right, so when you see the string:

$$t \to t \to t \to t$$

you know that, if the convention was followed, it is a rewriting of this string:

$$(t \to (t \to (t \to t))),$$

and, in cases where no unique string can be recovered from the sugaring, it will still be possible to check mechanically whether the output LLM has given us *a* permissible sugaring of the original. This *must* be done to ensure that the output LLM does not introduce any mistakes (such as changing the meaning of the original when it adds sugar, or producing a sugary-looking string that has no meaning at all). This process is like proof-checking, but what it verifies is the accuracy of the syntactic sugaring and other human-reader-friendly alterations introduced at the previous step—hence "sugar-checking"—thus ensuring that the output LLM does not alter the meaning of the solution. This is a job for GOFAI.

[4]These last two examples are not, strictly speaking, cases of *syntactic* sugaring, since syntax pertains to strings, but they are called "syntactic sugaring".

[5]In good logic textbooks, anyway, and there will be in UP as well

3.3 The rationale for UP's design

UP is a GOFAI module consisting of GOFAI submodules that perform tasks §3.3(ii), §3.3(iv), §3.3(v), and §(iv)(vii). It has gaps to be filled by GenAI modules that perform tasks §3.3(i), §3.3(iii), and §3.3(vi). Except for marketing purposes, and for building an AI system that we ourselves (those working on the project) will use, it is not important to us what GenAI modules are plugged in; the guarantee of perfect reliability, again, comes from the design of UP, which provides the rules according to which natural language deductive reasoning will proceed once the gaps are filled.

Let us next consider, in more detail, why UP has this design.

3.3.1 The nature of UP's guarantee of correctness

Let's begin with the nature of the guarantee of correctness that UP's design confers on any natural language deductive reasoning module based in it. It is the strongest possible kind of guarantee, because it is actually a *mathematical theorem* that:

(i) no such module (computer program) will ever give an incorrect answer to a question of the form "Is φ a deductive consequence of Γ?", where φ is a disambiguated sentence and Γ a set of disambiguated sentences; and

(ii) no such module will ever present an invalid chain of deductive reasoning when asked to justify its answer.

Thus, we should be as confident in the correctness of the answers and justifications given by any UP-based deductive reasoning module as we are in the truth of any mathematical theorem (once it has been proved, and this one has been): 100%.

That's the beauty of GOFAI: *When it comes to cognitive tasks that can be so precisely defined that there is an algorithm for determining what counts as a mistake in peforming the task, it is possible to come up with a modular design for a hybrid GOFAI/GenAI system such that it is a mathematical theorem that no system with that design—no matter what its GenAI components are like, and no matter how unreliable or reliable they are at performing their own subtasks—will ever make a mistake in performing the task.*

Of course, not every cognitive task is like this.[6] We are very lucky in that

[6]For example, computer vision, no matter how advanced and reliable, will never enjoy a mathematical guarantee of correctness, because there is no algorithm for determining whether an artificial visual system is making a mistake. In fact, it seems plausible that a properly functioning artificial (or natural) visual system *ought to* get things wrong, in some sense, in some cases. (See Williamson 1990 for some reasons why.) Of course, if we individuate cognitive tasks finely enough, it will be easy to find ones for which some notion of correctness can be given a precise enough definition for mathematical guarantees of correctness for an AI system peforming it to be imaginable. *Playing chess* is one such.

deductive reasoning, which is a significant component of AGI, is—and has been known to be since 1879.

The theorem described above cannot be stated here in full detail. That would require defining, among other notions, "deductive reasoning module" and "based on UP", as well as the design of UP itself—not down to the last detail, but in enough detail for it not to be appropriate for this document. However, without delving any deeper into these details than we already have, it will be possible to convey the basic idea of the theorem, which is extremely simple: no matter how you fill the gaps in UP, the resulting system will not produce any chains of deductive reasoning that are not in conformity with the rules of UP's *extended formal system*, by which we mean the combination of the formal system in which UP's HOL proofs are constructed with UP's rules of EFL ↔ HOL translation and UP's rules of syntactic sugaring[7]—that's §3.3.1(ii), and it's easy to see why it's true. §3.3.1(i) immediately follows, since the system will say that φ a deductive consequence of Γ if, and only if, it has found a derivation of φ from Γ in the extended deductive system. In software engineering terms, our theorem is a case of *software verification*, or of proving that a computer program, or a class of computer programs, does what it is supposed to do.

At this point, a philosophically-minded reader feel tempted to ask us how we know that the *rules* of UP's extended formal system are correct. They should resist that temptation, because we never claimed to know that those rules were correct, nor do we know what it would mean to attribute correctness to the rules of a formal system.[8] The question of whether it is a good idea to adopt that formal system over its competitors—such as some extension of Lean, for example—is not to be confused with the question of the (provable) correctness of deductive reasoning modules based

[7] A translation is a step in a chain of deductive reasoning. Think of the rules of translation as extending the rules of UP's deductive system by a rule like:

> If a sentence φ can be transformed into a sentence ψ by the rules of translation, then one may infer ψ from φ and vice versa.

Syntactic sugaring is also a step in a chain of deductive reasoning. Think of the rules of syntactic sugaring as extending the other rules by a rule like:

> If a sentence φ can be transformed into a sentence ψ by the rules of syntactic sugaring, then one may infer ψ from φ and vice versa.

[8] An answer one sometimes encounters in philosophical writing goes like this: the rules are correct only if they never allow one to derive a *false* sentence from a set of *true* sentences. It is, in fact, true—and even provable!—that the rules in question never allow one to derive a false sentence from a set of true sentences, but we do not think this banal observation in any way *justifies* the adoption of those rules, because, in proving it, we must *use those same rules* (and even stronger rules/axioms, if we formalize the theory of truth in the standard way, following Tarski 1933).

on UP, i.e., the question of whether their deductive reasoning (provably) conforms to the rules in question. The latter question is the topic of this subsection.

3.3.2 A division of labor, enabling a new paradigm for AI

The gaps are where they are in UP because placing them just there makes possible a certain desirable division of labor between GOFAI and non-GOFAI components that makes possible *not only* the guarantee of correctness considered above but *also* one in which the only possible sources of error are, broadly speaking, human error—and this, in turn enables a new paradigm in AI.

Clearly, the only possible source of error in a UP-based system for automating deductive reasoning is in the operation of the input LLM, and there the ultimate responsibility lies with the human user. If the human user approves an unintended disambiguation, then that is squarely on the human user; and if the human user decides not to intervene and to allow the input LLM to take its best guess as to what is meant, any unintended disambiguations are again the responsibility of the human user (one cannot avoid responsibility for oversight by deciding not to oversee).

This separation of tasks into those that can be achieved with perfect reliability and transparency by GOFAI and those that cannot be achieved with perfect reliability and transparency by anything, and a division of labor in which GOFAI performs the former and ultimate responsibility for the latter lies with human users is, in our view, essential for any acceptable AI system. It is absent in the current AI paradigm, but see §5.1.

3.3.3 Ease of assembly

There is yet one more reason why we have placed UP's gaps as we have, which has already been mentioned: another of our desiderata for UP is that it should be able to serve as a prefabricated GOFAI structure that can quickly, cheaply, and without giving much thought to the matter be assembled into a complete deductive reasoning module by the addition of widely available, inexpensive prefabricated GenAI slabs: an input LLM, an output LLM, and a neural prover.

4 A comparison with Lean

4.1 A quick comparison

A comparison with Lean, which is a similar system that solves a similar but not quite as general a problem, will be useful. Lean is a system for automating deductive reasoning in an artificial language (the *Lean Language*), and specifically for

automating reasoning in pure (as opposed to applied) mathematics. Lean has *three* gaps that must be filled by GenAI modules:

First, *the disambiguation and translation of ambiguous natural language into the Lean Language* is a single gap, which it would be natural to fill with an input LLM.

Second, *the search for proofs in the Lean Language, in Lean's formal system*, is a gap naturally filled with a neural prover.

Third, *translation from the Lean Language into ambiguous natural language along with syntactic sugaring* is a single gap, naturally filled by an output LLM.

DeepMind's AlphaProof fills all three gaps as indicated above. While AlphaProof is a formidable AI mathematician, as witnessed by its performance on 2024 IMO problems,[9] in virtue of its being based on Lean, AlphaProof lacks all of the characteristic virtues of UP—as will any automated deductive reasoning module based on Lean.

4.2 A more detailed comparison

Let's now compare UP with Lean in more detail, and consider the components of UP that contribute to its virtues along with their counterparts (or the absence of counterparts) in Lean.

4.2.1 Disambiguated natural language

In EFL, the disambiguated natural language used by UP, each sentence is its own parse tree, making parsing a trivial task. Each "word" of the language (each constant and each variable) is assigned both a *type* and a *grammatical category*.

Types are familiar from both logic[10] and programming, and in UP they are used both for the usual purpose—i.e., for checking that terms are well-typed—as well as for ensuring that the translations between EFL and HOL used by UP are correct. (Correct translations are compositional—i.e., each word and each formation rule has its own translation—and preserve the types of all words.) As in logic and programming, the type of a word is indicative of (but does not fully determine) its meaning. For example, in "if $x = y, y = x$", "if" has the type $t \to (t \to t)$, and this tells us that its meaning is a function from propositions to functions from propositions to propositions, but this does not set it apart from "and" or "or", which have the same type but different meanings.

[9] The version of AlphaProof that yielded the impressive performance actually used humans to fill the first gap.

[10] Where they originated, in the work of Bertrand Russell in the very early 20th century.

Grammatical categories are used for ensuring that EFL sentences are grammatical—meaning, roughly, that they are acceptable to a normal adult native speaker of English. The need for grammatical categories is easy to illustrate using the same example: "if $x = y, y = x$" is grammatical but "implies $x = y, y = x$" is not, even though "if" and "implies" have the same type and the same meaning. For this reason, we need to put some further tags on "if" and "implies" when we formulate the grammar of EFL, and we call these further tags "grammatical categories". Exactly which tags are used in the official grammar is of little importance, because the tags will be replaced by various other tags in different settings designed for different classes of user. For example, for some users, in some situations, giving "implies" the tag "Verb" and "if" the tag "Boolean" will be optimal.

The procedure we follow in specifying the grammar of EFL will look familiar to many linguists (it is essentially the same one, give or take, that was introduced by the logician Richard Montague, whose work has been very influential in linguistics), but the details are not important. What is important is how EFL sentences look (when properly typeset), and that they should wear their meanings on their sleeves, even for users who are not familiar with Montague's work or with the relevant areas of linguistics. Since UP should not force the user interface of any AI system constructed around it to display EFL sentences in any particular way, but it should make it easy for any programmer constructing such an AI system to solve the typesetting problem, UP uses LaTeX code for writing EFL sentences. Thus, by using a LaTeX compiler, one can make EFL sentences look nice without giving any thought to typesetting or graphic design, but one can also choose to display them in any other way one likes.

What does the LaTeX look like when compiled? There is no unique way it will look. Each EFL sentence will have a default LaTeX encoding, but UP will also have many alternative encodings to choose from. For example, there will be one that uses numerical indices to represent binding, as in:

$$[\text{every philosopher}]_1 \text{ thinks that he}_1 \text{ is clever,}$$

as well as one that represents binding by curved arrows above the sentence, among other alternatives.

Even though such matters of typesetting/graphic design are irrelevant to the job of UP, they are very important to us, because, if we do not have a POC version of an AI system based on UP in which all of this is in place, we will not be able to convince end users to adopt UP. They can only adopt UP by adopting some AI system based on it, and we must present them with one that has a highly intuitive and elegant GUI through which they can appreciate the real distinctive virtues of UP.

Note that, for an EFL sentence to "wear its meaning on its sleeve", it must be displayed together with its translation into HOL. For example, "implies" and "con-

tradicts" have the same type and grammatical category, but they differ in meaning, and what accounts for that is their being translated into HOL terms that differ in meaning. How to represent this is another matter of typesetting/graphic design, and the details don't matter here, but getting the details right will be important for our POC.

There is also much more to the structure (as opposed to typesetting/graphic design) of EFL than can even be sketched here, so we'll forego the sketches, but we nevertheless mention two examples of what else there is. First, since the unit of translation is not a sentence in the usual ("grammatical") sense, the grammar of EFL is not a grammar of sentences in the usual sense but of the genuine units of translation, which we may call *texts*. Books and journal articles are examples of texts. So are certain sorts of dialogues: e.g., between a chatbot and its user. (Luckily for us, LaTeXis well designed for representing the structure of a text.) Second, the grammar of *proofs* is an important part of the grammar of EFL—one that makes explicit much of what is implicit in logic textbooks. A GenAI system may be given the job of finding candidate abbreviations of proofs, but it cannot be given final say: our grammar will determine what is and is not an acceptable abbreviation of a proof.

Disambiguated natural language is nowhere to be found in Lean. Consequently, AlphaProof goes directly from ambiguous natural language to Lean to back, by an untransparent and unreliable process that the user cannot control or correct, except by doing the translations by hand (as was actually done to achieve AlphaProof's impressive IMO performance).

4.2.2 The language HOL

The language we are calling "HOL" is one that has been (give or take some frills and notational conventions) in use in mathematics/logic since Russell and Whitehead's *Principia Mathematica* (1910) as well as in automated theorem proving (Isabelle uses a close variant). Its primitive symbols and constructions are ubiquitous in professional mathematics and even, as we already mentioned, in high school mathematics texts. Its only primitive symbols are:

- The standard subset symbols "$\subseteq$" (of different types), which mean just what they appear to mean ("$\subseteq$" may be pronounced "every", as in "every man is mortal").
- An infinite stock of variables (of each type), which also look, if given the default typesetting, just like the variables of mathematics textbooks.

Its only constructions (formation rules) are:

- Function application, which looks like this: "fx".

- Function abstraction, which we write, following tradition, "$\lambda x.M$"; in a typical mathematics textbook this is written as "$x \mapsto M$" or "the function that maps x to M", or something similar (these variant constructions are included in EFL).

In contrast, the Lean Language includes several basic constructions (notably, ones with so-called dependent types) that are not to be found anywhere in normal mathematics and that are difficult to master without extensive study. (For what it's worth, none of the UP team members know how to use dependent types.)

4.2.3 Translations between EFL and HOL

It is well known that all mathematical/logical notions can be defined using only the resources specified above, so translating purely mathematical/logical EFL sentences into HOL is unproblematic.

Non-logical/mathematical notions like *man* and *mortal*, however, are not definable using only these resources, so there is a sense in which we cannot translate a sentence like "Every man is mortal" into HOL. But there is also a sense in which we can, and that is the sense we have in mind when we say that we explore the deductive consequences of "Every man is mortal" by exploring the deductive consequences of its translation into HOL: non-logical words like "man" and "mortal" translate, in this sense, into variables of the same type. Because deductive logic does not care about the difference between one non-logical word and another (of the same type), when we ask about the deductive consequences of "Every man is mortal", we might as well be asking about the deductive consequences of "Every F is G". This is the normal procedure in mathematics (consider, e.g., the proof of the independence of Euclid's parallel postulate[11]), and it is also our procedure.

There is a unique compositional translation from EFL into HOL, but not from HOL into EFL. When going in the latter direction, we must keep track of context. This is not as difficult as it sounds, because we do not actually first translate to HOL and then put our prover(s) to work. Rather, we give the prover the original EFL and the translation (in the form of rewrite rules, not all of which will be used in a

[11] Which, like any independence proof, can be reconstructed as one that proceeds by first "translating" all primitive terms of the science under consideration (in this case, geometry) into variables and then showing, by constructing a model that supplies suitable values for the variables, that the "translation" of the postulate whose independence is to be proved cannot be derived from the "translations" of the other postulates (nor can the "translation" of its negation). In the case of geometry, we can let the primitive terms be (following Hilbert) "point", "line", "plane" "between", "lies on", and "congruent". When we are interested in what deductively follows from some given postulates (axioms) one might formulate using those terms, we replace each of them with a distinct variable of the same type as the original term and then explore the deductive consequences of the resulting "translations".

typical proof—the usual procedure in automated theorem proving). For this reason, the proof we get from the prover that we add to UP will already tend to look a lot like EFL reasoning, requiring minimal further translation.

Lean does not provide any translations between the Lean Language and any disambiguated natural language, with the result that the only thing AlphaProof adds to Lean that is of value to users demanding transparency and a guarantee of correctness is the prover that searches for proofs in Lean.

4.2.4 The formal system

The *formal systems* of UP and Lean—i.e., the rules of proof (or rules of inference) according to which they go about their deductive reasoning—are both provably consistent according to the usual mathematical standard of consistency.[12] However, Lean's formal system is provably stronger than UP's,[13] so, in a sense, UP is "safer" than Lean: someone who does not accept the usual mathematical standard of consistency would have a reason to think: "Even though both are consistent by the usual standard of proof, UP is less likely to be inconsistent than Lean." However, when it comes to the question that matters in the present context—namely, the question of which formal system we want our AI to conduct its deductive reasoning in—we take such philosophical considerations to be, if not completely irrelevant, nevertheless of so little significance in comparison with questions of a more practical nature that we can safely ignore them.

In case what we have just said sounds glib, let's be clear that it isn't: If we are given a choice between two formal systems, S_1 and S_2, and there is no serious doubt about consistency of either, but it is clear that one is less "practical" than the other in a sense that would makes its adoption lead to more mistakes in its use (or to other tangible badness, such as slowing down the search for proofs), then, other things being equal, we should choose the more practical system even if it is stronger (and therefore, in a sense, more likely to be inconsistent) than the less practical one.

[12] I.e., consistent relative to ZFC. There is no such thing as a proof of consistency *simpliciter* (except in an inconsistent formal system), as we learned from Gödel in 1930. Mathematicians tend to call consistency relative to ZFC simply "consistency" because they assume that ZFC is consistent, not because they can prove it to be consistent but because no one has ever found a derivation of a contradiction in it, even though it is the most extensively studied formal system in history.

[13] Again, relative to ZFC—a qualification that will be elided in what follows. A quick sketch of a proof: Because Lean's formal system has a type of all finite types, that type can be used (as in Tarski's original 1933 construction) as the "unifying type" in the construction of a definition of truth for UP's formal system (all of whose types are finite); this, in turn, yields (again, following Tarski 1933) an easy proof of the consistency of UP's formal system in Lean's, which implies, by Gödel's second incompleteness theorem, that Lean's formal system is stronger than UP's (if UP's is consistent).

In any case, when it comes to the choice between the formal systems of UP and Lean, the former is favored by both the practical and the philosophical considerations. There are three main practical considerations:

Intensionality vs. extensionality: Lean's formal system is *extensional*, meaning that it cannot recognize any distinctions between propositions that have the same truth value. This makes that system unsuitable for the automation of deductive reasoning outside of pure mathematics, where such distinctions are constantly drawn. For example, we want to deny that the probability (P) that

$$r \;:=\; \text{it will rain in Melbourne tomorrow}$$

is either 0 or 1, but if we are reasoning in an extensional system, we can do so only at the cost of contradicting ourselves, because in an extensional system we can prove that

$$\text{either } r = \top \text{ or } r = \bot$$

and, no matter how we formalize our theory of probability, we can prove that

$$P(\top) = 1 \text{ and } P(\bot) = 0$$

so we can prove that

$$\text{either } P(r) = 1 \text{ or } P(r) = 0.$$

This does not mean that Lean's formal system cannot be used for reasoning about probability or other topics that require one to draw non-extensional distinctions. There are well-known *tricks* that can be used for embedding intensional formal systems into extensional ones. However, no such tricks are included in Lean, and thus, insofar as a deductive reasoning module based on Lean (such as AlphaProof) is able to do any deductive reasoning involving probabilities other than 0 and 1, the key parts of that reasoning are carried out by its input and output LLMs (or other GenAI modules), so there can be no guarantee of correctness for that reasoning.

In contrast, the formal system in which UP's proofs are constructed is not extensional but is *intensional* (meaning that it cannot draw any distinctions between logically necessarily coextensional propositions), and this turns out to be just the right strength for the automation of any deductive reasoning about any topic whatsoever.

Natural language: No natural language deductive reasoning can be carried out in Lean's formal system. In contrast, there is no natural language deductive reasoning

that cannot be carried out in Lean's extended formal system.

Standard rules of proof: The core (non-extended) formal system of UP is made up entirely of standard rules of proof that will be readily understood by mathematically literate users, making it easy for such users to manually check the proofs if they wish to do so. In contrast, the rules of Lean's proof system are notorious for being incomprehensible to anyone but specialists in Lean.

5 The uses of UP

What's UP good for? Many things, one of which is the introduction (or re-introduction) of a new paradigm in AI. We'll begin with that, and then consider some examples of UP's many uses.

5.1 A new paradigm for AI

Once UP's division of labor in the automation of natural language deductive reasoning is available, a new paradigm for AI systems—one that is both vastly cheaper and more reliable than the current one—will also become available. In this new (or actually old but forgotten[14]) paradigm, AI systems know (or believe, assert, ...) things by representing some of their knowledge (beliefs, etc.) by means of a sparse set of disambiguated sentences drawn from trusted sources and by deducing, by means of a UP-based module, the rest of what they know (believe, etc.) from those sentences. Such an AI system will be incapable of "hallucination". There will be no guarantee that everything it deduces is true, but there *will* be a guarantee that, *if it deduces anything false*, then it will have done so by deducing that falsehood from one or more false sentences in the sparse set of disambiguated sentences drawn from the trusted sources, and there are only two ways for a false sentence to get into that set: either by an incorrect disambiguation of a true sentence or by an error in the original trusted source. In either case, the problem can be fixed by finding false sentences in the sparse set and either deleting entirely or replacing them with true sentences (the replacements may be correct disambiguations of the sentences found in the original trusted source). The internal workings of such an AI system will be perfectly transparent: its errors, unlike those of an LLM, will always be due to some false sentences in the sparse set that, when excised, will prevent those errors from recurring. Such an AI system will feature LLMs or other GenAI systems as component modules, but—as in the design of UP itself—in ways that do not compromise its transparency, reliability, and controllability.

[14]The paradigm is as old as the method of formalization (Frege 1879).

For many uses of UP, the "trusted source" of the sparse set of sentences need not be a source that *we* (actually) trust. There are many reasons why we may be interested in what deductively follows from sentences whose source someone else trusts (or what we hypothetically trust).

5.2 Examples of uses of UP

It is not easy to think of an example of an AI system that could not be greatly improved by the addition of a module for automating all deductive reasoning with perfect reliability, transparency, plus control over the disambiguation/paraphrase of the user's input.

Clearly, any AI system that uses natural language for interacting with its user would be: such a system needs to know what deductively follows from what the user says, as well as what deductively follows from any responses the system is considering giving.

In this connection, it should be emphasized that, although deductive consequence is only directly defined for (disambiguated) *declarative* sentences, the system would deliver equally dramatic improvements in performance with *commands* (imperative sentences) and *questions* (interrogative sentences). For example, (i) as a matter of deductive logic, a text-to-image generator that receives the instruction "Generate an image of 4 pairs of sunglasses where every pair with a letter printed on one lens has a number printed on the other lens" must (if it is to comply with the prompt) ensure that, in the generated image, each pair without a number printed on one lens does not have a letter printed on the other lens. To take another example, (ii) as a matter of deductive logic, a document editor's Help chatbot that is asked "Is it true that, if a Sans Serif font includes quantifier symbols, then all of its math symbols also come in boldface?" must (if it is to give a correct answer) give an affirmative answer if, and only if: if not all of a Sans Serif font's math symbols come in boldface, then it does not include quantifier symbols. Although the principle of deductive logic that underwrites both (i) and (ii) is completely elementary, GenAI reasoners will inevitably make mistakes in tasks that require applying it.

For the reader who would like to have more specific examples of the usefulness of UP, we note that all of the most important applications of UP are instances of the same schema, and thus it will be more profitable for the user to come up with their own instances of the schema (which pertain to their own interests or to what they think of as important tasks for AI). We will describe this schema in what follows, then give a small number of examples of its instances, and let the reader's imagination fill in the rest.

Here is the schema: First, we use UP to assist disambiguate some set of sentences (or some text) and a sentence; then we ask UP whether that sentence is a deductive

consequence of that set of sentences, and (normally) we get an answer along with a proof of the answer.

Here is a short list of examples:

First example: the FBI and the BLT killer. The case of the BLT killer—a once-prolific serial killer in the US who earned his nickname by leaving half-eaten bacon, lettuce, and tomato sandwiches at all crime scenes—has been cold for decades, but he has recently resurfaced and is presumed to be holding a hostage in his basement. The FBI has less than 24 hours to decide which basement in Wichita, Kansas to raid, if any, so it has less than 24 hours to discover the identity of the BLT killer—something that may well be a deductive consequence of the information it already has, but, if it is, the FBI's human agents will not be able to perform the deduction in less than 24 hours (they weren't able to do it in the last 30 years). When they give the job to the UP Reasoner (the special FBI version, which has access to all of the information to which the FBI has access, within a few minutes, either (i) a name and an address is found or (ii) a proof is found that the BLT killer's identity is not a deductive consequence of that information. If the outcome is (ii), but the answer is implicit in the information—though not deductively implied by it—then a GenAI module, which excels at generating plausible intermediate hypotheses, can assist by querying the agents working on the case to quickly find the answer. Naturally, the agent leading the investigation understands that the UP Reasoner's answer is no more accurate than the sparse set of sentences from which it deduced it and the disambiguations of those sentences, and will review the process that yielded the answer before deciding to act on it. (Note that the UP Reasoner here works somewhat like the "AI agents" recently introduced by the industry—for example, it can query various local law enforcement agencies, search the internet, read any relevant article or book, etc.—but, unlike the "agents", its deductive reasoning is impeccable, and its design guarantees that human errors and omissions are the only possible source of error.)

Three remarks about this example:

Fits, any organization—and indeed any person—is faced with many situations relevantly like this every day, although usually with lower stakes, and only an AI system based on UP can resolve them in a reliable and transparent way. For example, the answers to many questions—some of them urgent—about how I should go about assigning grades in an undergraduate course I am teaching are deductive consequences of the information contained in the vast number of emails I have exchanged with my Department Chair together with University policy documents X, Y, and Z. If I need an answer to one of those questions tonight, before the grades are due, and the Chair is out of touch, then I'm simply out of luck. This a job for the UP Reasoner (the special University version). It is not a job for Google Gemini or any other GenAi gizmo; those are both incapable of finding non-trivial deductive

connections and liable to make mistakes of reasoning.

Second, as is well known, the FBI and other organizations do use AI systems for tasks relevantly similar to the above, and some of the relevant AI systems—e.g., ones for DNA profiling—cannot in any way be improved upon by UP (except although their natural language user interfaces can be). What's missing, and what is urgently needed, is a single AI system, like the Reasoner, imagined above, that combines all of the information to which all of the other AI systems have access into a sparse set of sentences (regarded as having a trusted source) and searches for their deductive consequences. No other AI system is capable of doing this in a reliable and transparent way, and, because of the volume of the information, humans are not capable of doing it at all.

Third, to non-experts in logic, the "review the process" mentioned above may sound like fantasy, given that the sparse set of sentences from which the Reasoner reasons is too large for any human to survey, but there are good empirical reasons to believe that, at least in the vast majority of cases of other than purely academic interest, there will be a very small, humanly surveyable set of premises that would need to be checked by the reviewer. (The vast majority of the sentences in the sparse set will not be used in the deduction at all, and, of the small number that will be used, an even smaller number will require human attention.)

Second example: Putin's world view. In the first example, UP was used in the most obvious way: for discovering facts about (non-psychological aspects of) the world: We have some sentences, which we presume to be true, and we ask which further sentences deductively follow from them, and which are also true given our presumption. Doing this may enable us to rescue a hostage, to discover a life-saving drug, to submit our undergraduate grades on time, etc. In another important class of cases, we are interested in the deductive consequences of some sparse set X of sentences not because we presume X to be true but because either every member of X is accepted by a certain agent a or we take X to be a good model of a's beliefs, and we are interested in the question "What is the world like from a's point of view?" This is a natural question to ask when we are interested in questions about what a would do under various hypothetical circumstances, since—at least in one simple model that is surprisingly useful when applied to real-world cases—an agent's actions are determined by how the agent represents the world as being together with the way the agent desires the world to be. A less simple and more widely applicable model (known as decision theory) takes the agent's rational actions to be determined by the conditional probabilities the agent assigns to alternative states of the world (conditional on choices made by the agent) and how valuable or desirable the agent takes those states of the world to be. But no matter how sophisticated we get in our modelling, we must be able to probe the deductive consequences of the sentences that represent the agent's beliefs (perhaps to a degree) as well as

the deductive consequences of the sentences the agent desires-true (perhaps to a degree) for the modelling to be of any use to us. For example, when constructing and deploying a model of the mind of Vladimir Putin—particularly to predict how he would respond to the presence of NATO troops in Crimea—we must be able to demonstrate that the sentence, "If there are NATO troops in Crimea, then there are NATO troops on Russian territory", is a deductive consequence of other statements that Putin believes or accepts. However, this sentence itself is not explicitly included in the sparse set of statements used to represent his beliefs (or degrees of belief). The example may seem trivial, because, if we are reasoning intuitively about Putin's beliefs, then of course we will attribute to him the belief: "If there are NATO troops in Crimea, then there are NATO troops on Russian territory." But when we are constructing a formal model (i.e., a computer simulation-*cum*-model) of Putin's beliefs to use for predicting his actions—which we must do, for the same reason why we have replaced intuitive reasoning about the weather with formal models for predicting the weather—it is not at all a trivial matter how to represent his beliefs using sentences that deductively entail "If there are NATO troops in Crimea, then there are NATO troops on Russian territory." That requires the automation of natural language deductive reasoning by GOFAI methods—something that no known alternative to UP can deliver.

Third example: Simplifying algorithms. Programmers spend a lot of their time trying to come up with algorithms that are simpler than the simplest currently known algorithm for computing some given function. The reasoning they engage in when they do this cannot be profitably automated or even formalized without the automation of natural language deductive reasoning. At present, the available options are to either (i) not formalize/automate the reasoning at all or to (ii) actually define the algorithms at issue by hand in some programming language or other, and then to do a lot of further very tedious manual work in order to obtain a formal result about those pieces of code in an ATP/proof assistant such as Isabelle. (i) is too prone to error; (ii) is too expensive. There is also: (iii) ask the latest model by OpenAI (or similar) to do the work—but that's even more error-prone than (i). Here, UP is potentially a game-changer, because it actually enables programmers to get the code and the theorems about it they need by doing (i) (but doing it with a little more care than usual). If you describe an algorithm in natural language, an AI system based on UP will help you disambiguate your description of it so that it singles out a unique algorithm about which the system can automatically prove things, as well as proving things about its encodings in particular programming languages. (E.g., "find the shortest Quine in the such-and-such programming language" is already completely unambiguous EFL. How good a UP-based AI system will be at solving problems like this depends in the awesomeness of the prover we plug into one of its gaps.)

6 Anticipated further contributions to AI

While breaking new ground in GenAI is not one of our ambitions, we think that it is likely that work on project will lead to the development of new GenAI methods for achieving the following important tasks—which have, unfortunately, been overlooked in current AI work:

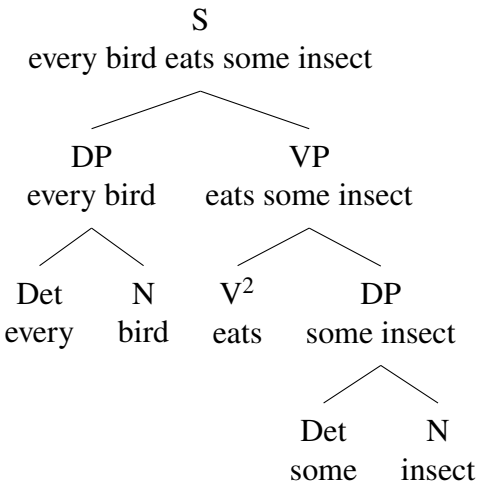
6.1 Automating the writing of grammars

We hope to be able to use machine learning to extract a usable *explicit* grammar of an arbitrary natural language from a large enough corpus. Writing grammars that generate all and only the "grammatical" (roughly, acceptable-sounding) sentences of a given natural language is *extremely* tedious and boring work that can be automated—although no one so far has figured out how—and its automation will make possible the rapid expansion of UP to cover all known natural languages. Note that this problem is *not* the same as that of training a model to recognize and to produce grammatical sentences of a natural language—that problem has already been solved by LLMs. A good LLM, such as the latest ChatGPT, is actually "better at grammar" than the vast majority of human speakers, in that it will make grammatical mistakes with a far lower frequency than the vast majority of human speakers, and will recognize ungrammatical sentences far more reliably as well. Thus, LLMs have solved the problem of automatically extracting an *implicit* representation of a natural language grammar from a corpus. The problem we want to solve is that of automatically extracting an *explicit* representation of a grammar from a natural language corpus, one that associates each word with one or more grammatical categories (for which we may wish to use labels like "Verb", "Adjective", etc., although the most efficient grammars are unlikely to include categories that correspond exactly to those of grade school grammar[15]), and supplies a list of rules by which each a sentence may be assembled out of its constituent words (e.g., "Sentence $\longrightarrow$ NounPhrase + VerbPhrase"[16]). A model with an explicit representation of a natural language grammar will then be able to "show its work" when asked how, exactly, it arrived at the conclusion that a given string of words typed by a user

[15] As we learn from mathematical linguistics (e.g., the work of J. Lambek), and especially that work in this area that is done with the purpose of facilitating formalization (the work of R. Montague and various later researchers influenced by that work). Not that we think that mathematical linguists have discovered the most efficient means of defining natural language grammars, but their methods are nevertheless better than those of grade school grammar.

[16] To give an example of a formation rule in the style made famous by N. Chomsky in the 1950s.

was a grammatical sentence of English, e.g., by displaying a parse tree like:



(1)

Our problem might well turn out to be easier than the protein folding problem that was effectively tackled by DeepMind's AlphaFold.[17]

6.2 Automating lexicography/morphology

In a sense, the specification of the lexicon (the words and their grammatical categories), including the various 'forms' of various words (e.g., plural vs. singular—commonly known as morphology) is included in the specification of a grammar, but it is a subtask that can benefit from its own special methods, which we also have ideas about. In particular: various lexicons (e.g., dictionaries) are rich sources of information on lexical items and their 'forms', some of which cannot be extracted from a corpus. The automation of the the extraction of that information from a lexicon—which we hope to achieve—will both accelerate our own work on UP and have many other applications.

6.3 Automating disambiguation

Once we have a grammar for a natural language, whether written by hand or produced by machine learning, the further task of suggesting plausible disambiguations (in our case, corresponding EFL parse trees) for each sentence generated by the grammar will have to be automated. This will involve pairing each sentence with a different sort of parse tree, one that breaks it down into parts that can be used for translating

[17] Which, we note, was more akin to the problem of extracting an implicit grammar from a corpus than the problem actually before us.

the sentence, as intended, into HOL (that is the purpose of EFL). Note that the problem is not that of finding *every* EFL-parse tree for a given sentence (typically there are too many) nor of finding *at least one* EFL-parse tree for it (most of them would yield implausible translations) but of finding the unique EFL-parse tree that captures the meaning most likely intended by the writer/speaker of the sentence—or, if there is not a unique one, then a manageably short list of options from which the user can choose (or from which the system itself can choose at random, for some uses of UP).

We already know, on the basis of our own small-scale experiments as well as the experiments reported in the literature,[18] that LLMs will be reasonably good at this task. However, a model trained *exclusively* for this task might well approach or exceed human-level performance at natural language writing/speech comprehension. Generating data for the training of such a model would be a significant challenge, which we have ideas about how to meet.

6.4 Automating the design of EFL

EFL is a work in progress. We do not expect to have hit upon the optimal grammar for disambiguated English, and much of the point of the work described above is to automate significant parts of the further development of EFL. Perhaps most obviously, because we want EFL's parse trees to look just like those of the plain grammar (e.g., (1)), with a few labels—e.g., for types—added, the continuous revision of the rules of the grammar of EFL will be informed by the insights gained by applying our methods for extracting grammars from corpora.

6.5 Automating formalization

A natural language deductive reasoning module based on UP already *is* a device for automating formalization (of texts, theories, and whole sciences)—which is to say, it can easily be used for that purpose. However, there is important work to be done on what kind of further non-GOFAI augmentation of such a system would make it especially good at autoformalization. For example, if we autoformalize a textbook in quantum field theory, we want the result to be consistent, but we can expect that we will only get consistent autoformalizations that exclude some sentences in the book. A search of all such autoformalizations would be useless even if it could be carried out (which it cannot be). Luckily, LLMs are very good at making guesses about which sets of (disambiguated) sentences are likely to have been intended, and GOFAI methods can be used for eliminating the inconsistent sets among these.

[18] The research that led to AlphaProof is especially relevant here.

There are, then, good reasons to be optimistic that UP can be used as a component of a specialized autoformalization module that that outperforms the state of the art in the area.

Autoformalization is another area in which progress can accelerate the progress we make on the development of UP. Perhaps most obviously, we can generate the largest library of formalized mathematics in existence by autoformalizing all of published human mathematics, and then allow UP to use any result it finds in that library as a lemma in its reasoning.

6.6 Automating non-monotonic forms of reasoning

Non-monotonic[19] forms of reasoning (e.g., counterfactual/subjunctive reasoning and inductive reasoning) have stubbornly resisted formalization since Frege introduced the method of formalization in 1879. We agree with the mainstream view (among logicians) that the reason for this is that these forms of reasoning cannot be formalized at all. But that doesn't mean that they cannot be automated. It seems to us *obvious* that they can be automated reasonably well by a combination of GOFAI and non-GOFAI methods presently available to us. In particular, LLMs, being a sophisticated form of predictive text, seem better suited than any other AI systems available at present for this task, which involves both deleting premises (those that can no longer be assumed given a new premise) and adding premises beyond those that follow by pure deductive logic from the premises that remain. Here are three examples:

Counterfactual reasoning: Non-trivial counterfactual (AKA subjunctive) conditionals answer questions about how the world *would have been* (in certain respects) it had been different from the way it actually is (in certain other respects). Consider the following contrast between indicative (AKA material) and counterfactual conditionals:

If Oswald didn't kill JFK, then someone else did. (Indicative.) (2)

If Oswald hadn't killed JFK, then someone else would have. (Counterfactual.) (3)

(3) is false; (2) is true (presumably—unless, e.g., Oswald was part of a conspiracy that included a back-up assassin who would have killed Kennedy if Oswald failed). In fact (insofar as we are ruling out a conspiracy), we can confidently assert that, if Oswald hadn't killed JFK, then JFK would have continued his re-election campaign

[19] A consequence relation $\Rightarrow$ is said to be monotonic when $\Gamma, \varphi \Rightarrow \psi$ whenever $\Gamma \Rightarrow \psi$, i.e., what $\Rightarrow$-follows from a certain set of premises also $\Rightarrow$-follows from those same premises together with any other premises.

on November 23, 1963. To arrive at this judgment we do something like this: We suppose 'Oswald didn't kill JFK', and we also assume many other facts that we know to hold, but not all—in particular, we do not assume that *someone* killed JFK, on November 23, 1963, nor do we assume that JFK died on that date or within a few days of it—and we then see whether 'JFK continued his re-election campaign on November 23, 1963' follows deductively from what we have supposed. There is no known way of specifying the rules according to which we decide which known facts to assume and which ones not to assume when we do this, but we know that LLMs are already quite good at simulating what we do when we do this.

Decision-theoretic reasoning: In decision-theoretic reasoning, we consider the consequences (in some non-monotonic sense) of our taking various actions available to us.

Inductive reasoning: Inductive reasoning proceeds from instances to generalizations: e.g., an inference of the form

All Fs observed thus far are G / All Fs are G

will be a good inductive inference in some contexts, for some Fs and Gs. Inductive reasoning is constrained by the rules of probability and logic in various ways (e.g., if φ is logically inconsistent with ψ, then φ/ψ is not good a inductive inference), but so far attempts to spell out the rules of inductive reasoning, insofar as they go beyond those minimal constraints, have yielded no agreement on the rules. And yet, at least for a variety of special domains, *there is a definite sense in which we know how to automate inductive reasoning*, since that is exactly what machine learning does: it extracts patterns (generalizations) from their instances. LLMs, for example, are so good at predicting the next token in a string of characters that LLM-based chatbots can leverage this ability for fooling human interlocutors into thinking that they understand English. LLMs and other GenAI gizmos, however, do not express their inductive reasoning (from their training data to their implicit generalizations) in any language, whereas inductive reasoning, in the sense in which we can hope to automate it, is reasoning with sentences. There are many ways we can think of to use the resources of GenAI for automating deductive reasoning, and we expect to make some progress in this area.

Two further points:

First, every one of the six tasks listed here ((6.1)-(6.6)) (including the first two subtasks of the (6.6), as well as, obviously, its third subtask) is a matter of automating inductive reasoning in some sense.

Second, there is no way to automate inductive reasoning in any of these senses (or in any others) in a provably correct or transparent way: because there are no formal rules of inductive reasoning, we cannot prove that our automated inductive reasoner

will always perform its inductive reasoning in accordance with the rules. The closest thing to a *rule of inference* of inductive reasoning is the learning algorithm of a model that performs inductive reasoning as it ought to, but such an algorithm, since it has no premises and no conclusion, is not a rule of inference. The direction of evaluation here, as we see it, is the opposite of what we have in the automation of deductive reasoning: *first* we automate some species of inductive reasoning in a passable way in some model, and *then* we use that model for evaluating various proposed rules of that species of reasoning.

All of the examples cited above of areas in which advances in non-GOFAI automation are likely to be achieved are examples of possible future **work that is "empirical"** in the sense that we cannot hope to *prove*, in advance of adopting a particular method, that it will deliver the results that we want. Experimentation will be required, but that is the norm in AI nowadays, and we are by no means averse to experimentation.