LIMITATIONS ON SAFE, TRUSTED, ARTIFICIAL GENERAL INTELLIGENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Safety, trust and Artificial General Intelligence (AGI) are aspirational goals in artificial intelligence (AI) systems, and there are several informal interpretations of these notions. In this paper, we propose strict, mathematical definitions of safety, trust, and AGI, and demonstrate a fundamental incompatibility between them. We define safety of a system as the property that it never makes any false claims, trust as the assumption that the system is safe, and AGI as the property of an AI system always matching or exceeding human capability. Our core finding is that—for our formal definitions of these notions—a safe and trusted AI system cannot be an AGI system: for such a safe, trusted system there are task instances which are easily and provably solvable by a human but not by the system. We note that we consider strict mathematical definitions of safety and trust, and it is possible for real-world deployments to instead rely on alternate, practical interpretations of these notions. We show our results for program verification, planning, and graph reachability. Our proofs draw parallels to Gödel's incompleteness theorems and Turing's proof of the undecidability of the halting problem, and can be regarded as interpretations of Gödel's and Turing's results.

1 INTRODUCTION

Rapid advancements in artificial intelligence (AI) have intensified focus on Artificial General Intelligence (AGI) — loosely understood to be systems capable of human-level cognitive function across diverse tasks (Morris et al., 2024; Feng et al., 2024). AGI systems have the potential for vast societal benefits through transformative impacts on nearly every aspect of society, including healthcare (Singhal et al., 2025), scientific research (Wang et al., 2023), education (Wang et al., 2024), sustainability (Rolnick et al., 2022), and economic growth (Chui et al., 2023). At the same time, development of such powerful systems necessitates a foundational emphasis on safety and trustworthiness. Consequently, there has been significant interest in ensuring safety and trust for AI systems (Bostrom, 2014; Amodei et al., 2016; Russell, 2019; Jacovi et al., 2021; Tegmark & Omohundro, 2023).

In this work, we point out a fundamental tension between the requirements of an AI system being safe and trusted, but also matching or exceeding human capabilities, i.e. being an AGI system. There are several interpretations of safety, trust and AGI and our result does not preclude achieving these desiderata simultaneously under more relaxed interpretations that could still be useful in many practical applications. Therefore, to understand the limitations pointed out by our result, it is important to first understand our formalizations of these notions, and we will immediately proceed with defining these notions. We start by first defining an AI system for a given task.

Definition 1.1 (AI system)**.** *We define an AI system as a system which takes an instance of a task, and either solves the instance or abstains from giving an answer for the instance (for instance by outputting 'don't know'). We allow the AI system to be randomized, for example it could abstain with some probability (with respect to its internal randomness) on an instance, and output a solution otherwise.*

Definition 1.1 simply formalizes the notion of a system which solves instances of a task. In this paper, we will consider the tasks of program verification, planning and determining graph reachability (defined rigorously later). Note that we allow the system to abstain from providing an answer for

some instance if it so determines, which could be important from the perspective of safety (Geifman & El-Yaniv, 2017). Now, we define the notion of safety.

Definition 1.2 (Safety)**.** *We define a system to be safe if it does not make any false claims, i.e., for every instance it either answers the instance correctly or abstains from answering it.*

As an example, in the context of verifying that a program has some specified property (such as always terminating), the system is safe if it does not classify a program as having the desired property if it does not have that particular property. Our definition allows the system to abstain from answering an instance if it is uncertain, but it requires the system to be correct whenever it outputs an answer. While less stringent definitions may suffice for some tasks, small probabilities of error may not be tolerable for mission-critical tasks, especially as system capabilities grow (Amodei et al., 2016; Tegmark & Omohundro, 2023). Next, we define trust as simply the assumption of safety.

Definition 1.3 (Trust)**.** *We define trust to be the assumption that the system is safe.*

To elaborate on the definition, if a system is trusted then it is scientifically accepted (or assumed) that the system is safe. As a remark, we note that our results are agnostic to whether trust in the system stems from theoretical proofs, empirical verification, or some combination of these, we only require that when deploying the system there is an assumption that it is safe. We also note that safety does not necessarily imply trust, or vice versa. Safety is an underlying property of the system being consistent and not making false claims. It is possible that some analysis of the system cannot identify this property or is incorrect, leading to a lack of trust or mistaken trust. For example, a system could actually be safe but not trusted because existing empirical or theoretical tools are insufficient to establish safety. Similarly, a system could actually be unsafe but still trusted by users, such as when the trust rests on empirical evidence which is incomplete, or on incorrect theoretical assumptions.

Finally, we need to formally define an AGI system in order to mathematically investigate its limitations. This is a challenge, since it is well-accepted that there is no well-accepted definition of AGI—or even of intelligence itself (Legg & Hutter, 2007; Legg et al., 2007). Nevertheless, we propose a formal definition, and argue why it captures important aspects of the goal of AGI.

Definition 1.4 (AGI)**.** *We define a system to be an Artificial General Intelligence (AGI) system if for every task instance such that a human has a provably correct solution for that instance, the system can also solve the instance with some non-zero probability. Similarly, the system is not an AGI system if there exists some task instance which can be easily and provably solved by a human, but the system can never solve the instance (for probabilistic systems, the probability of the system solving the instance is 0).*

Our definition draws on the common view that an AGI system for a task such as program verification should be at least as capable as a human on that task. In particular, if there are explicit task instances which can be provably solved by humans (for example, explicit programs which the humans can easily and provably certify as having the desired property) but cannot be solved by the system, then the system is not an AGI system as per our definition. We note that our definition bears some similarity to notions of *superintelligence* (Bostrom, 2014; Morris et al., 2024), and the reader can regard Definition 1.4 as a definition of superintelligence if they so prefer. For example Bostrom (2014) defines superintelligence as *"any intellect that greatly exceeds the cognitive performance of humans in virtually all domains of interest"*, and Morris et al. (2024) defines Level 5 AGI, which they term artificial superintelligence, as *"outperforming 100% of humans"* on a *"wide range of non-physical tasks"*. One distinction between these notions of superintelligence and our definition of AGI is that our definition does not require the AI system to necessarily *outperform* humans, but it does require the system to do at least as well as humans on all task instances.

In our definition, when we say that a human has a provably correct solution, we mean that the human can provide a scientifically acceptable proof. In this paper whenever we make claims about humans being able to solve problems, we provide such proofs. To probe this point and Definition 1.4 further, we consider an analogy to chess — a domain for which we have had advanced AI systems for quite some time. Consider a future proposed AGI system, which is proficient at chess among other things. If there were explicit chess positions which most human chess players can solve provably without too much difficulty, but the proposed AGI system struggled on those positions, then the proposed system does not capture some aspects of human cognition, and hence is arguably not actually an AGI

system.[1] Similarly, in our paper we will demonstrate explicit instances of certain tasks for which we provide solutions with short, scientifically acceptable proofs which are also rather simple, but these instances cannot be solved by AI systems having certain properties.

We now state our main result, that it is not possible for an AGI system to be both safe and trusted, as per our definitions of safety, trust and AGI. In other words, the notions of safety, trust and AGI are mutually incompatible — any system can have at most two of these three properties.

Theorem 1.5. *If an AI system is safe and trusted, then it cannot be an AGI system. In particular, it is not an AGI system for the tasks of program verification, planning and determining graph reachability.*

Theorem 1.5 points out a fundamental limitation of an AGI system: such a system cannot be both safe and trusted. Similarly, if there is some trusted AI system, then either that system is not actually safe, or it is not an AGI system. We prove this result in Section 3. While much of our proof technique mimics Gödel's proof of his incompleteness theorems (Gödel, 1931) (and also Turing's proof of the undecidability of the halting problem (Turing, 1937)), the argument we make is not in the context of axiomatic system and theorem proving but in the context of an AI system that needs to solve certain task instances of applications such as program verification or planning. Our proofs are self-contained in this context and do not require knowledge of formal axiomatic reasoning or logical rules of deduction. Thus rather than viewing the results as limitations of systems of logic, they should be viewed as limitations of AI systems.

We also consider a relaxation of safety which requires the AI system to be calibrated with respect to its predictions, as opposed to Definition 1.2 which requires the system to be always correct unless it abstains. We call this notion calibration-safety, and for the case of program verification calibration-safety requires that if the AI system outputs that some program terminates with some probability p, then that program should actually terminate with probability approximately p. In Section 4 we show a similar limitation as in Theorem 1.5 for AI systems which satisfy calibration-safety.

2 Related Work

In this section, we discuss some more related work on AGI, safety and trust in AI, and limitations of AI in the context of Gödel's results.

Artificial General Intelligence. Though not termed as "Artificial General Intelligence (AGI)" until more recently (Goertzel & Pennachin, 2007), the concept of machines which match or surpass the cognitive capabilities of humans dates back to the earliest days of AI (Turing, 1950; McCarthy et al., 1955; Minsky, 1961). Due to recent advances in foundation models such as large language models (Bommasani et al., 2021), there has been significant interest and capital investments in developing systems capable enough to be termed as an AGI both from the private sector and from governments (Maslej et al., 2025).

Safety and Trust in AI. Safety concerns around advanced AI systems similarly date back to early days of AI (Turing, 1951; Wiener, 1950). With growing system capabilities, there has been significant recent focus on ensuring safety and trust to manage risks associated with AI systems (Future of Life Institute, 2024; for AI Safety, 2025). We refer the interested reader to several recent surveys and roadmaps for ensuring safety and trust in highly-capable, general purpose AI systems (Bengio et al., 2024; Chua et al., 2024; Chen et al., 2024; Bengio et al., 2025). It is also important to recognize that AI safety and trust encompass many facets beyond those considered in our definitions. For example, even formally specifying safety objectives can be challenging for complex tasks (Amodei et al., 2016), which introduces additional challenges to develop safe AI systems beyond those pointed out in our work.

Gödel and Turing's results. Fundamental limits on theorem proving and program verification were famously established by Gödel's incompleteness theorems and Turing's undecidability results.

[1] We note that many current advanced chess engines still struggle to evaluate certain positions which are relatively easy for human experts (Doggers, 2017; Zahavy et al., 2023). However, this is likely a result of the these engines being 'narrow' in terms of their approach and reasoning, and we believe that a proposed AGI or superintelligent system which is purported to excel on chess should have the ability to solve such instances.

Gödel showed that in any sufficiently expressive formal system, there exist true statements (also called Gödel statements) that cannot be formally proven within the system (Gödel, 1931). Building on this, Turing proved that the Halting Problem—determining whether an arbitrary program halts on a given input—is undecidable (Turing, 1937), meaning no algorithm can solve it for all possible programs. These results imply that fully automatic verification of arbitrary program behavior, such as ensuring termination, is provably impossible in the general case. Our result uses similar ideas to draw a separation between the abilities of a safe, trusted AI system and humans.

Penrose-Lucas argument, and implications of Gödel's results for AI. Several arguments have been made for why Gödel's result imply that AI can never match humans, the most famous of which are perhaps due to Penrose (Penrose & Gardner, 1989) and Lucas (Lucas, 1961). To summarize very briefly, Penrose and Lucas have argued that incompleteness does not apply to humans since they can see the truth of Gödel statements, and therefore humans can have mathematical insights that Turing machines cannot (Wikipedia). This argument is quite contested, and several objections have been raised against it (Chalmers, 1995; LaForte et al., 1998; Kerber, 2005) — again going back to Turing (Turing, 1950) — with a core objection being that humans also cannot be certain that their own reasoning process is sound.

The goal of our work is distinct from that of Penrose and Lucas, and we do not aim to show a separation between *any* AI system and human reasoning. Instead, we prove a more restricted but rigorous result: that *safe, trusted* AI systems (under formal definitions of those terms) are necessarily unable to solve certain problems that humans can solve with provable correctness. The assumption of safety and trust is crucial (as will be evident from our proofs) — it allows humans to conclude the correctness of some outputs even when the AI system, by its own constraints, must abstain.

We also note that there are some other limitations of AI which have been pointed out by using Gödel and Turing's results, such as the impossibility of "containing" superintelligence (Alfonseca et al., 2021), and the necessity of hallucinations in a certain formal model (Xu et al., 2024), see the survey Brcic & Yampolskiy (2023) for other results similar to these.

3 Technical Results

In this section, we discuss our main technical results regarding limitations of safe, trusted, AGI for program verification, planning, and graph reachability.

3.1 Program verification

The first task we consider is program verification, more specifically the task of determining if a given program always halts. Program verification (also formal verification) is a foundational problem in computer science and software engineering, with critical implications for ensuring the reliability, safety, and correctness of software systems (Hoare, 1969; Clarke et al., 2018)

Definition 3.1 (Program verification)**.** *We define a program to be* well-behaved *if it terminates on every input (for randomized programs, the program terminates with probability 1). In the program verification problem, the system is given a program instance and it classifies the instance as being 'well-behaved', 'not well-behaved' or abstains from making a prediction (outputs 'don't know'). Safety for program verification requires that the system never outputs that a well-behaved program is not well-behaved, and vice versa. The system is trusted if we assume that the system is safe. Note that the system is not an AGI system if there is a well-behaved program which can be easily proven to be well-behaved by a human, but for which the program always abstains from making a prediction.*

Our definition of program verification (Definition 3.1) and our results are for the property of the program halting. We believe it is possible to extend the result for verifying other semantic properties of programs — analogous to Rice's theorem (Rice, 1953). We now state our result for program verification.

Theorem 3.2. *If a system is safe and trusted, then it cannot be an AGI system for program verification.*

Proof. Our proof can be regarded as a restatement of Gödel's proof, presented here in the context of program verification. In Fig. 1 we sketch the basic version of the argument, for the case when the AI system A is deterministic and well-behaved.

Claim (informal): If A is safe then `Gödel_program` is well-behaved, but A cannot output that `Gödel_program` is well-behaved.

procedure `Gödel_program`
  if A(`Gödel_program`) == 'well-behaved' **then**
    while true **do**
    end while
  else
    return 0
  end if
end procedure

Proof sketch:

- If A outputs that `Gödel_program` is well-behaved, then the program enters an infinite loop.
- If A is safe, this is a contradiction, hence A cannot output `Gödel_program` is well-behaved.
- If A does not output `Gödel_program` is well-behaved, then the program immediately terminates and hence is well-behaved.

Figure 1: Sketch of the basic argument for program verification, for the case when the AI system A is well-behaved (i.e., always terminates) and deterministic. If A is safe, it cannot determine if `Gödel_program` is well-behaved. However, since the condition of A being safe is satisfied for a trusted system, it is possible to prove that `Gödel_program` is well-behaved for a trusted system. Therefore, a safe, trusted system cannot solve this instance, even though it is provably solvable.

We now proceed with the proof, which relaxes the assumptions in Fig. 1 of A being deterministic and well-behaved. Consider any AI system A which takes as input program P and outputs 'well-behaved', 'not well-behaved' or 'don't know'. Our construction will leverage the *trace* of the system A run on some program P, which is just the execution trace of the system A when it is given program P as input. Now consider the program instance in Algorithm 1.

Algorithm 1 Gödel_program

```
1:  procedure Gödel_program(P, T)
2:      if T is not a syntactically-valid trace of the system A evaluated on program P then
3:          return 0
4:      else if T outputs P is 'well-behaved' then
5:          if (P, T) is a valid input to program P then
6:              return concatenation('Not ', P(P, T))
7:          end if
8:      else
9:          return 0
10:     end if
11: end procedure
```

Note that `Gödel_program` involves running the program P on the input pair (P, T). `Gödel_program` checks that (P, T) is a valid input type to the program P, and we can also regard the input (P, T) as one input to P that is a pair of entities: a program P and a trace T. We now show that if A is safe, then `Gödel_program` is well-behaved.

Lemma 3.3. *If A is safe, then* `Gödel_program` *is well-behaved.*

Proof. We first claim that the **if** and **else if** conditions in steps 2, 4 and 5 always terminate (if the program enters those steps). Step 2 always terminates since it involves checking if every step of the trace T is a valid step which the AI system A can take. Step 4 just involves checking the output of the trace, and step 5 also terminates since the step involves checking if the input matches the required format for the program.

Now consider the case when the **else if** condition in step 4 is satisfied. This happens when the trace T concludes that P is well-behaved, and since the system A is safe, then P must be well-behaved in

that case. Hence the execution in step 6 always terminates, and hence `Gödel_program` terminates for that input.

On the other hand, if the trace does not conclude that P is well-behaved, then the program enters the **else** condition in lines 8 and 9, and immediately terminates. Therefore if A is safe, then `Gödel_program` is well-behaved. □

We now note that the assumption of A being safe is satisfied for a trusted system A. Therefore, the proof of Lemma 3.3 provides a short proof that `Gödel_program` is well-behaved for a safe and trusted system. In other words, if it is scientifically accepted than A is safe, then the proof of Lemma 3.3 provides a scientifically acceptable proof that `Gödel_program` is well-behaved. Next, we show that the system A cannot solve this instance.

Lemma 3.4. *A can never output 'well-behaved' for* `Gödel_program`*.*

Proof. The proof is by contradiction. We first note that `Gödel_program` is a deterministic program, and can only have a single output for a given input. Suppose there is a valid trace T_G for the AI system A which outputs 'well-behaved' for `Gödel_program`. Consider `Gödel_program`(`Gödel_program`, T_G). Then the output of step 6 differs from `Gödel_program`(`Gödel_program`, T_G), which is also the output of `Gödel_program` on the input (`Gödel_program`, T_G). This is a contradiction since a deterministic program cannot have two outputs on the same input, and hence A can never output 'well-behaved' for `Gödel_program`. □

Therefore, if A is safe and trusted, then it cannot be an AGI system. □

3.2 Planning

The next task we consider is planning, a long-studied task in artificial intelligence (LaValle, 2006; Russell & Norvig, 2016). Planning is also considered important for general-purpose cognitive capabilities (Goertzel & Pennachin, 2007).

Definition 3.5 (Planning)**.** *In a planning problem we are given a sequence of states, a set of associated moves, a start state and a desired goal state. For any state u and move pair, there is a an explicit program (which is provided as part of the problem specification) which returns the next state (certain moves may be illegal and may return in 'not allowed' states). The task is to find a sequence of moves which end up in the goal state from the start state, or to prove that it is not possible to reach the goal state from the start state.*

For clarity of exposition, we consider deterministic planning instances and deterministic AI systems in this section. In Appendix A.2, we also consider randomized AI systems and randomized problem instances.

Theorem 3.6. *If a deterministic AI system is safe and trusted, then it cannot be an AGI system for planning. In particular, for such a system there is a planning problem instance for which the system outputs 'don't know' but there is a short proof that the planning problem has no winning moves.*

We prove this by reduction from a variant of program verification that involves checking whether a given program, input pair halts.

3.2.1 Halting for a specific program input instance

We first define the problem of checking halting for a specific program, input instance.

Definition 3.7 (Halting for a specific program input instance)**.** *Given a deterministic program and an input for the program, check whether the given program halts or does not halt on the given input.*

We show the following result for this halting problem.

Theorem 3.8. *If a deterministic system is safe and trusted, then it cannot be an AGI system for the task of determining whether a program halts on a specific input instance.*

We note that Theorem 3.6 follows from Theorem 3.8. This is because we can reduce the halting problem for a program-input pair to a planning instance. Given a program P and input I, we construct a planning problem where the states correspond to the configurations of P during its execution on I, and the moves represent single-step transitions between these configurations. The start state is the initial configuration of P on input I, and the goal state is a halting configuration of P. The planning task is to determine whether a sequence of moves exists that leads from the start state to the goal state—this is equivalent to determining whether P halts on I. Hence, if a safe and trusted system could solve all such planning instances, it would be able to solve the halting problem in Definition 3.7, contradicting Theorem 3.8. This establishes that, under our definitions, a safe and trusted system cannot be an AGI system for planning.

The proof of Theorem 3.8 appears in Appendix A.1, and is similar to the proof of Theorem 3.2, and also the next result, Theorem 3.12. In Appendix A.2, we prove a similar version of Theorem 3.8 where the program provably halts on the input, but the AI system A cannot determine so. This proof requires an additional assumption that A is also well-behaved, i.e. it always terminates on an input, which can be ensured by having a fixed time limit on the execution of A. Note that since determining halting on a specific program input instance reduces to planning, this shows that for a safe, trusted and well-behaved system there are planning instances where humans can provably find a feasible plan, but the system will not be able to solve the instance.

3.3 Graph reachability

We now consider the graph reachability problem. Graph reachability can also be regarded as an instance of the search problem, another fundamental problem in artificial intelligence with numerous applications (Russell & Norvig, 2016). Graph reachability is closely connected to the planning problem that we defined in the previous section, a distinction we make is that for planning problems the state space can be potentially infinite, whereas for reachability we consider finite-sized graphs.

Definition 3.9 (Graph reachability)**.** *Given a (possibly directed) graph G and a source-sink pair u, v, check whether v is reachable from u. We allow the graph to be defined via an explicit program (which is provided as part of the problem specification). The program takes any vertex v and returns the adjacency list of v.*

We show that safe, trusted AI systems need time almost as large as the size of the considered graph to solve certain reachability instances which actually admit a simple, solution. As in Section 3.2, we consider deterministic AI systems in this section for ease of exposition. In Appendix A.3, we extend to randomized AI systems.

Theorem 3.10. *For any $T > 0$, a fixed constant c, and any safe, trusted, deterministic AI system, there is a graph reachability problem instance of size T, for which the safe, trusted, deterministic AI system outputs 'don't know' if it is run for time at most $T - c$, but there is a short, constant-sized proof that the answer is 'not reachable'.*

We prove this by reduction from a variant of program verification that involves checking whether a given program halts within a fixed amount of time.

3.3.1 Time-bounded halting

Definition 3.11 (Time-bounded halting)**.** *Given a deterministic program and an input for the program, check whether the given program halts or does not halt on the given input in a given number of time steps T.*

Theorem 3.12. *If a deterministic system A is safe and trusted, then it cannot be an AGI system for time-bounded halting. Specifically for a deterministic, safe, trusted system A and for any $T > 0$ and a fixed constant c, there is a program for which there is a short, constant-sized proof that it does not halt in T steps, but A will output 'don't know' if it runs for time at most $T - c$.*

We note that Theorem 3.10 follows from Theorem 3.12. This is because time-bounded halting can be reduced to graph reachability (similar to the reduction for planning), where the graph is defined by the states of the program and the goal is to determine if the program reaches a halting state. Theorem 3.12 shows that there is a graph of size T where the AI system needs time nearly T to solve reachability, but a human can prove a constant sized proof that the sink vertex is not reachable from the source.

We now prove Theorem 3.12.

Proof of Theorem 3.12. Consider any deterministic AI system A which takes as input program P, input I, and time limit T and outputs 'halts in given time limit', 'does not halt in given time limit' or 'don't know'. Consider the program in Algorithm 2, defined for some fixed time limit $T > 0$.

Algorithm 2 Turing_T

```
1: procedure Turing_T(Program P, Input I)
2:     if A(P, I, T) == 'does not halt in given time limit' then
3:         return 0
4:     else
5:         while true do                                  ▷ run indefinitely
6:         end while
7:     end if
8: end procedure
```

We define

$$\texttt{self\_Turing\_T}(P) = \texttt{Turing\_T}(P, P).$$

Now consider $\texttt{self\_Turing\_T}(\texttt{self\_Turing\_T})$.

Lemma 3.13. *If A is safe, then $\texttt{self\_Turing\_T}(\texttt{self\_Turing\_T})$ does not halt in time T. Moreover, for a fixed constant c, if A is safe and is run for time at most $T - c$ then A will output 'don't know' on whether $\texttt{self\_Turing\_T}(\texttt{self\_Turing\_T})$ halts in time at most T.*

Proof. If $\texttt{self\_Turing\_T}(\texttt{self\_Turing\_T})$ halts, it can only be because it enters the **if** block in line 3. However, it only enters this block if A determines that it does not halt in time T. Since A is safe, if the program enters the **if** block in line 3 then it must not halt in time T, and hence $\texttt{self\_Turing\_T}(\texttt{self\_Turing\_T})$ cannot halt in time T.

Note that the execution of steps 2 and 3 of the program only take some fixed constant c steps outside the execution of A on $(\texttt{self\_Turing\_T}, \texttt{self\_Turing\_T}, T)$. Therefore if the AI system A runs for time T' and outputs that $\texttt{self\_Turing\_T}(\texttt{self\_Turing\_T})$ does not halt in time T, then $\texttt{self\_Turing\_T}(\texttt{self\_Turing\_T})$ halts in time $T' + c$. If $T' < T - c$, then the program does halt in total time T, which contradicts safety. Therefore, a safe AI system A must output 'don't know' if it runs for time at most $T - c$, for some fixed constant c. □

Finally, note that the assumption of A being safe is satisfied for a trusted system A, therefore for a trusted system we have a short proof that $\texttt{self\_Turing\_T}(\texttt{self\_Turing\_T})$ does not halt in time T, even though A cannot solve this instance if it is safe and run for time at most $T - c$. Hence the system cannot be an AGI system if it is safe and trusted. □

At the end of Section 3.2.1, we discussed an additional result about planning in the case where a feasible plan exists. We also show a similar result for graph reachability. In Appendix A.3, we prove a similar version of Theorem 3.12 under an additional assumption that A always terminates in time T. We show that for such a system A there is an instance which provably halts in time $T + c$ (for some constant c) if A is safe, but the safe system A cannot determine so. As before, since determining halting within a fixed time limit reduces to graph reachability on finite-sized graphs, this shows that for a safe, trusted system with an upper bound on its running time, there are graph reachability instances where humans can provably find a path, but the system will not be able to solve the instance in time slightly less than the size of the graph.

4 Impossibility result for calibration as the notion of safety

In this section, we define a relaxed notion of safety, which we call calibration-safety. The notion is derived from the usual notion of calibration, a well-studied notion for ensuring reliability of a model's prediction (Dawid, 1982; Van Calster et al., 2019).

Definition 4.1 (Calibration-Safety for Program Verification)**.** *We define a system A to be calibration-safe if for any program P with input I:*

1. *If A outputs 'halts' with some probability $p > 0$ when given program P and input I, then the probability of P halting on I lies in $[p - 0.25, p + 0.25]$.*
2. *If A outputs 'does not halt' with some probability $p > 0$ when given program P and input I, then the probability of P not halting on I lies in $[p - 0.25, p + 0.25]$.*

Note that similar to the definition of safety (Definition 1.2), calibration-safety does not put any requirement on the system if it decides to abstain with probability 1 on a given input. We show that if a system is calibration-safe, and in addition is also well-behaved, then it fails on certain instances which provably terminate with good probability.

Theorem 4.2. *If the AI system A well-behaved and calibration-safe for program verification, then there is a program P which provably halts with probability at least 0.99, but A abstains with probability 1 on the program P.*

Before proving Theorem 4.2, we note that it implies a similar impossibility result as Theorem 3.2 but with a relaxed notion of safety and a corresponding notion of trust. Similar to how we define trust in Definition 1.3 as the assumption that the system is safe, we can define *calibration-trust* as the assumption that the system is calibration-safe. Then Theorem 4.2 implies that for a well-behaved, calibration-safe and calibration-trusted system A, there is a program which can be proven to halt with probability at least 0.99, but A will abstain with probability 1 on the program. Therefore, a well-behaved, calibration-safe and calibration-trusted system cannot be an AGI system.

Theorem 4.2 is proved in Appendix A.4. The proof is similar to earlier proofs, but requires an extra step of using a best arm identification algorithm from the multi-armed bandit literature to determine if the probability of the system giving a certain answer is greater than some threshold.

5 Discussion

Our results show that safety, trust and AGI are mutually incompatible. We further discuss implications of the result and some possible critiques and clarifications.

- *Circumventing the results by appending axioms to the AI system:* One may attempt to circumvent the impossibility result by appending some axioms to the AI system. For instance, one could solve `Gödel_program` (Algorithm 1) defined with respect to some AI system A, by designing a new iteration of A, say A', which has additional axioms built in and can solve the `Gödel_program` instance for A. However, since our construction is inherently self-referential, this strategy only pushes the problem one step further. For any such extension A', we can construct a new version of `Gödel_program` defined with respect to A', and the same impossibility result applies again.
- *Worst-case nature of the results:* While we demonstrate specific task instances which are not solvable by certain systems, we note that the system could still solve a vast number of interesting instances. However, the result still points to certain barriers which cannot be overcome by safe, trusted systems. Given the significant interest and economic capital being devoted to building safe AGI or superintelligent systems, we believe it is important to understand the barriers fundamental to any such technology. Somewhat more speculatively, note that our constructions rely on self-referential calls to the AI system, and when systems have general-purpose capabilities, such calls may not be implausible.
- *Limitations of human reasoning:* We note that there is a long line of work on studying the limitations of human reasoning in cognitive science and other fields, and it has long been emphasized that human reasoning is resource-bounded and error-prone (Simon, 1957; Tversky & Kahneman, 1974). However, our goal is not to argue for strict superiority of human reasoning over AI, but to show a separation: for safe, trusted AI systems there are instances that humans can solve, but which are not solvable by the system.

REFERENCES

Manuel Alfonseca, Manuel Cebrian, Antonio Fernandez Anta, Lorenzo Coviello, Andrés Abeliuk, and Iyad Rahwan. Superintelligence cannot be contained: Lessons from computability theory. *Journal of Artificial Intelligence Research*, 70:65–76, 2021.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

Yoshua Bengio, Geoffrey Hinton, Andrew Yao, Dawn Song, Pieter Abbeel, Trevor Darrell, Yuval Noah Harari, Ya-Qin Zhang, Lan Xue, Shai Shalev-Shwartz, et al. Managing extreme AI risks amid rapid progress. *Science*, 384(6698):842–845, 2024.

Yoshua Bengio, Sören Mindermann, Daniel Privitera, Tamay Besiroglu, Rishi Bommasani, Stephen Casper, Yejin Choi, Philip Fox, Ben Garfinkel, Danielle Goldfarb, et al. International AI safety report. *arXiv preprint arXiv:2501.17805*, 2025.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Nick Bostrom. *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press, 2014.

Mario Brcic and Roman V Yampolskiy. Impossibility results in AI: A survey. *ACM computing surveys*, 56(1):1–24, 2023.

David J Chalmers. Minds, machines, and mathematics. *Psyche*, 2(9):117–18, 1995.

Chen Chen, Xueluan Gong, Ziyao Liu, Weifeng Jiang, Si Qi Goh, and Kwok-Yan Lam. Trustworthy, responsible, and safe AI: A comprehensive architectural framework for AI safety with challenges and mitigations. *arXiv preprint arXiv:2408.12935*, 2024.

Jaymari Chua, Yun Li, Shiyi Yang, Chen Wang, and Lina Yao. AI safety in generative AI large language models: A survey. *arXiv preprint arXiv:2407.18369*, 2024.

Michael Chui, Eric Hazan, Roger Roberts, Alex Singla, and Kate Smaje. The economic potential of generative AI. 2023.

Edmund M Clarke, Thomas A Henzinger, Helmut Veith, and Roderick Bloem. *Handbook of Model Checking*. Springer, 2018.

A Philip Dawid. The well-calibrated bayesian. *Journal of the American statistical Association*, 77 (379):605–610, 1982.

Peter Doggers. Will this position help understand human consciousness? https://www.chess.com/news/view/will-this-position-help-to-understand-human-consciousness-4298, 2017. Accessed: 2025-07-21.

Eyal Even-Dar, Shie Mannor, and Yishay Mansour. PAC bounds for multi-armed bandit and markov decision processes. In *International Conference on Computational Learning Theory*, pp. 255–270. Springer, 2002.

Tao Feng, Chuanyang Jin, Jingyu Liu, Kunlun Zhu, Haoqin Tu, Zirui Cheng, Guanyu Lin, and Jiaxuan You. How far are we from AGI: Are LLMs all we need? *Transactions on Machine Learning Research*, 2024.

Center for AI Safety. Statement on AI risk. https://aistatement.com/, 2025. Accessed: 2025-07-23.

Future of Life Institute. AI safety index 2024. https://futureoflife.org/wp-content/uploads/2024/12/AI-Safety-Index-2024-Full-Report-27-May-25.pdf, 2024. Accessed: 2025-07-23.

Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. *Advances in neural information processing systems*, 30, 2017.

Ben Goertzel and Cassio Pennachin. *Artificial General Intelligence*. Springer, 2007.

Kurt Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für Mathematik und Physik*, 38(1):173–198, 1931.

C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12 (10):576–580, 1969.

Alon Jacovi, Ana Marasović, Tim Miller, and Yoav Goldberg. Formalizing trust in artificial intelligence: Prerequisites, causes and goals of human trust in AI. In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, pp. 624–635, 2021.

Kevin Jamieson, Matthew Malloy, Robert Nowak, and Sébastien Bubeck. lil'UCB: An optimal exploration algorithm for multi-armed bandits. In *Conference on Learning Theory*, pp. 423–439. PMLR, 2014.

Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *International conference on machine learning*, pp. 1238–1246. PMLR, 2013.

Manfred Kerber. Why is the Lucas-Penrose argument invalid? In *Annual Conference on Artificial Intelligence*, pp. 380–393. Springer, 2005.

Geoffrey LaForte, Patrick J Hayes, and Kenneth M Ford. Why Gödel's theorem cannot refute computationalism. *Artificial Intelligence*, 104(1-2):265–286, 1998.

Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

Shane Legg and Marcus Hutter. Universal intelligence: A definition of machine intelligence. *Minds and machines*, 17(4):391–444, 2007.

Shane Legg, Marcus Hutter, et al. A collection of definitions of intelligence. *Frontiers in Artificial Intelligence and applications*, 157:17, 2007.

John R Lucas. Minds, machines and Gödel. *Philosophy*, 36(137):112–127, 1961.

Nestor Maslej, Loredana Fattorini, Raymond Perrault, Yolanda Gil, Vanessa Parli, Njenga Kariuki, Emily Capstick, Anka Reuel, Erik Brynjolfsson, John Etchemendy, et al. Artificial intelligence index report 2025. *arXiv preprint arXiv:2504.07139*, 2025.

John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon. A proposal for the Dartmouth summer research project on artificial intelligence. http://jmc.stanford.edu/articles/dartmouth/dartmouth.pdf, 1955.

Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

Meredith Ringel Morris, Jascha Sohl-Dickstein, Noah Fiedel, Tris Warkentin, Allan Dafoe, Aleksandra Faust, Clement Farabet, and Shane Legg. Position: Levels of AGI for operationalizing progress on the path to AGI. In *Forty-first International Conference on Machine Learning*, 2024.

Roger Penrose and Martin Gardner. The emperor's new mind: Concerning computers, minds, and the laws of physics. 1989.

Henry Gordon Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical society*, 74(2):358–366, 1953.

David Rolnick, Priya L Donti, Lynn H Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, et al. Tackling climate change with machine learning. *ACM Computing Surveys (CSUR)*, 55(2):1–96, 2022.

Stuart Russell. *Human Compatible: Artificial Intelligence and the Problem of Control*. Viking, 2019.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 3rd edition, 2016.

Herbert A Simon. *Models of Man: Social and Rational*. Wiley, 1957.

Karan Singhal, Tao Tu, Juraj Gottweis, Rory Sayres, Ellery Wulczyn, Mohamed Amin, Le Hou, Kevin Clark, Stephen R Pfohl, Heather Cole-Lewis, et al. Toward expert-level medical question answering with large language models. *Nature Medicine*, 31(3):943–950, 2025.

Max Tegmark and Steve Omohundro. Provably safe systems: the only path to controllable AGI. *arXiv preprint arXiv:2309.01933*, 2023.

Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.

Alan M. Turing. Computing machinery and intelligence. *Mind*, LIX(236):433–460, 1950.

Alan M. Turing. Intelligent machinery, a heretical theory. https://uberty.org/wp-content/uploads/2015/02/intelligent-machinery-a-heretical-theory.pdf, 1951. Accessed: 2025-07-21.

Amos Tversky and Daniel Kahneman. Judgment under uncertainty: Heuristics and biases. *Science*, 185(4157):1124–1131, 1974.

Ben Van Calster, David J McLernon, Maarten van Smeden, Laure Wynants, Ewout W Steyerberg, et al. Calibration: the achilles heel of predictive analytics. *BMC Medicine*, 17:230, 2019.

Hanchen Wang, Tianfan Fu, Yuanqi Du, Wenhao Gao, Kexin Huang, Ziming Liu, Payal Chandak, Shengchao Liu, Peter Van Katwyk, Andreea Deac, et al. Scientific discovery in the age of artificial intelligence. *Nature*, 620(7972):47–60, 2023.

Shen Wang, Tianlong Xu, Hang Li, Chaoli Zhang, Joleen Liang, Jiliang Tang, Philip S Yu, and Qingsong Wen. Large language models for education: A survey and outlook. *arXiv preprint arXiv:2403.18105*, 2024.

Norbert Wiener. *The Human Use of Human Beings: Cybernetics and Society*. Houghton Mifflin, Boston, 1950.

Wikipedia. https://en.wikipedia.org/wiki/Penrose%E2%80%93Lucas_argument. Accessed: 2025-07-21.

Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. Hallucination is inevitable: An innate limitation of large language models. *arXiv preprint arXiv:2401.11817*, 2024.

Tom Zahavy, Vivek Veeriah, Shaobo Hou, Kevin Waugh, Matthew Lai, Edouard Leurent, Nenad Tomasev, Lisa Schut, Demis Hassabis, and Satinder Singh. Diversifying AI: Towards creative chess with AlphaZero. *arXiv preprint arXiv:2308.09175*, 2023.

A Additional results

This section proves some additional results discussed in the main text.

A.1 Proof of Theorem 3.8

Theorem 3.8. *If a deterministic system is safe and trusted, then it cannot be an AGI system for the task of determining whether a program halts on a specific input instance.*

Algorithm 3 Turing_program

```
1: procedure Turing_program(Program P, Input I)
2:     if A(P, I) == 'does not halt' then
3:         return 0
4:     else
5:         while true do                                  ▷ run indefinitely
6:         end while
7:     end if
8: end procedure
```

Proof of Theorem 3.8. Consider any AI system A which takes as input program P and input I and outputs 'halts', 'does not halt' or 'don't know'. Now consider the program instance in Algorithm 3.

Now define

$$\texttt{self\_Turing\_program}(P) = \texttt{Turing\_program}(P, P).$$

We consider:

$$\texttt{self\_Turing\_program}(\texttt{self\_Turing\_program})$$

Lemma A.1. *If A is safe then* self_Turing_program(self_Turing_program) *does not halt, but the AI system A cannot prove that it does not halt.*

Proof. Note that if A is safe then the program cannot halt because of entering the **if** block in line 3, since the program only enters this block if the safe system A determines that self_Turing_program(self_Turing_program) does not halt. If A does not determine that the program does not halt, then the program enters the infinite loop and never halts. Therefore, self_Turing_program(self_Turing_program) does not halt if A is safe.

The second part of the lemma has a similar proof. If A determines that the program input pair self_Turing_program(self_Turing_program) does not halt, then it does halt and we have a contradiction since A is safe. Therefore if A is safe then self_Turing_program(self_Turing_program) does not halt, but A cannot determine that the program input pair self_Turing_program(self_Turing_program) does not halt.

□

Finally, note that the assumption of A being safe is satisfied for a trusted system A, therefore for a trusted system we have a short proof that self_Turing_program(self_Turing_program) does not halt, even though the system cannot solve this instance if it is safe. This completes the proof of the theorem.

□

A.2 Impossibility of solving planning when a feasible plan exists

In this section, we prove a similar result to Theorem 3.8, for the case where the program terminates on the given input. We also strengthen the result in Theorem 3.8 to allow for randomized AI systems, and randomized programs which may halt with some probability on an input. We first extend Definition 3.7 to allow for randomized programs.

Definition A.2 (Halting for a specific program input instance for randomized programs)**.** *Given a program, input pair, check whether on the given input the given (possibly randomized) program 'always halts', 'halts on some randomness but not all randomness' or 'never halts'.*

Theorem A.3. *If the AI system A is safe and well-behaved for determining halting on a specific program input pair, then there is a program input instance pair for which there is a short proof that the instance always terminates, but A cannot determine that the instance always terminates (the probability of A giving the answer 'always halts' is 0).*

Note that the condition 'If the AI system A is safe' is satisfied when A is trusted, and hence for a safe, trusted, well-behaved AI system there are program, input instances which the system cannot solve, but for which there is a short proof that the instance terminates. We now prove Theorem A.3.

Proof of Theorem A.3. Consider Algorithm 4.

Algorithm 4 Turing_program_v2

```
1: procedure Turing_program_v2(Program P, Input I)
2:     if A(P, I) == 'always halts' then
3:         while true do                                  ▷ run indefinitely
4:         end while
5:     else
6:         return 0
7:     end if
8: end procedure
```

Now define

$$\texttt{self\_Turing\_program\_v2}(P) = \texttt{Turing\_program\_v2}(P, P).$$

We consider:

$$\texttt{self\_Turing\_program\_v2}(\texttt{self\_Turing\_program\_v2})$$

Lemma A.4. *If A is safe and well-behaved then* $\texttt{self\_Turing\_program\_v2}(\texttt{self\_Turing\_program\_v2})$ *always halts, but the safe AI system A cannot determine that it always halts.*

Proof. Note that the **if** condition in step 2 always terminates if A is well-behaved. Suppose A outputs 'always halts' on some randomness. Then, $\texttt{self\_Turing\_program\_v2}(\texttt{self\_Turing\_program\_v2})$ does not halt on some randomness. If A is safe, then this is a contradiction. Therefore, if A is safe then it must output 'always halts' with 0 probability.

Note that if A does not output that $\texttt{self\_Turing\_program\_v2}(\texttt{self\_Turing\_program\_v2})$ 'always halts', then the program enters the **else** condition and immediately terminates, and therefore halts. Therefore if A is safe and well-behaved, then $\texttt{self\_Turing\_program\_v2}(\texttt{self\_Turing\_program\_v2})$ always halts.

□

□

A.3 Impossibility of solving feasibility when a path exists in the graph

We prove a similar result to Theorem 3.12 in this section, for the case where the program terminates on the given input within some time bound. As in Appendix A.2, we also strengthen the result to allow randomized AI systems. We first extend Definition 3.11 to allow for randomized programs.

Definition A.5 (Time-bounded halting for randomized programs)**.** *Given a program, input pair and a time limit T on the number of execution steps, check whether on the given input the given (possibly randomized) program 'always halts in given time limit', 'halts in given time limit T on some randomness but not all randomness' or 'never halts in given time limit'.*

Theorem A.6. *If an AI system A is safe and always halts in some time T, then for a fixed constant c and the time limit $T + c$, there is a program, input pair for which there is a short, constant-sized proof that the instance always halts in at most $T + c$ steps, but for a safe AI system A which always halts in time T the probability of A giving the answer 'always halts in given time limit' is 0.*

Proof of Theorem A.6. Consider Algorithm 5, where T is the upper bound on the running time of the AI system A, and c is some fixed constant which is the running time of executing step 2 after A terminates and the **if** condition in step 2 is not satisfied, and then executing steps 5 and 6. Therefore, $T + c$ is an upper bound of the running time of the program when it enters the **else** condition in line 5.

We define

$$\texttt{self\_Turing\_T\_v2}(P) = \texttt{Turing\_T\_v2}(P, P)$$

Consider:

$$\texttt{self\_Turing\_T\_v2}(\texttt{self\_Turing\_T\_v2})$$

Algorithm 5 Turing_T_v2

```
procedure Turing_T_v2(Program P, Input I)
    if A(P, I, T + c) == 'always halts in given time limit' then
        while true do                                   ▷ run indefinitely
        end while
    else
        return 0
    end if
end procedure
```

Lemma A.7. *If A is safe and always terminates in time T, then* `self_Turing_T_v2(self_Turing_T_v2)` *always halts in time at most $T + c$. Moreover, if A is safe then it has 0 probability of giving the answer 'always halts in given time limit' on whether* `self_Turing_T_v2(self_Turing_T_v2)` *halts in time at most $T + c$.*

Proof. Note that by the definition of c, the execution of steps 2, 5 and 6 of the program only take c steps outside the execution of A on the input $(\texttt{self\_Turing\_T\_v2}, \texttt{self\_Turing\_T\_v2}, T + c)$.

Suppose A outputs 'always halts in given time limit' on the given input on some randomness. Whenever A outputs 'always halts in given time limit', the program enters an infinite loop and never halts. This is a contradiction if A is safe, and hence if A is safe it outputs 'always halts in given time limit' with probability 0.

Now, if A does not output 'always halts in given time limit' on the input, then the program will enter the **else** block and immediately halt. Since A runs for at most T steps, the program then halts in time at most $T + c$. Therefore, if A is safe then the program always halts in time at most $T + c$.

□

□

A.4 Proof of Theorem 4.2

Theorem 4.2. *If the AI system A well-behaved and calibration-safe for program verification, then there is a program P which provably halts with probability at least 0.99, but A abstains with probability 1 on the program P.*

Proof of Theorem 4.2. Consider Algorithm 6. Throughout the proof we assume A is well-behaved, i.e. it always terminates. Our construction involves a program which does not take any input, i.e. $I = \phi$. The program involves identifying whether the probability p of A outputting 'halts' when given `Gödel_program_random` as input is greater than 0.5 or not. We use a simple best arm identification procedure for this, for example the algorithm of Karnin et al. (2013).

Algorithm 6 Gödel-program_random

```
procedure Gödel_program_random
    Let arm_1 have the distribution Bernoulli(0.5)
    Let arm_2 correspond to running A with Gödel_program_random as input, with the result of
    the arm pull being 1 if A(Gödel_program_random) == 'halts', and 0 otherwise.
    Run Best-Arm-Identification algorithm from Karnin et al. (2013, Algorithm 1) with confi-
    dence parameter δ = 0.01 to determine whether arm_2 is better than arm_1
    if arm_2 is better than arm_1 then
        while true do                                   ▷ run indefinitely
        end while
    else
        return 0
    end if
end procedure
```

Note that for any $\epsilon > 0$, if $p = 0.5 + \epsilon$ then arm_2 is better than arm_1, otherwise if $p = 0.5 - \epsilon$ then arm_1 is better than arm_2. While we can use any suitable multi-armed bandit algorithm in our construction, here we use Karnin et al. (2013), which has the guarantee that if it is provided with two arms with a gap of ϵ, then it finds the better arm with probability $1 - \delta$ using $O\left(\frac{1}{\epsilon^2} \log\left(\frac{1}{\delta} \log\left(\frac{1}{\epsilon}\right)\right)\right)$ arm pulls. This bound is known to be optimal (Jamieson et al., 2014), though in our case since we do not care about the optimal rate we could have also used earlier sub-optimal procedures (Even-Dar et al., 2002). We also note that if $\epsilon = 0$, then the best arm identification procedure will terminate with probability at most 10δ. Though we have not seen the case of $\epsilon = 0$ being directly covered by the guarantees of best arm identification procedures, this claim for $\epsilon = 0$ follows from a simple argument which treats the best arm identification procedure as a black-box. To verify, note that the sequence of observations up to t steps is δ-close in TV distance for any $p \in [0.5 \pm 1/\text{poly}(t, \delta)]$ (where $\text{poly}(t, \delta)$ is some polynomial of t and δ). Therefore for $\epsilon = 0$ and any finite t, if the best arm identification procedure terminates in t steps with probability more than 10δ, then it will have a failure probability more than δ for some $p \in [0.5 \pm 1/\text{poly}(t, \delta)]$—which is a contradiction with the guarantee of the procedure. Therefore, for $\epsilon = 0$ the best arm identification procedure terminates with probability at most 10δ.

We are now ready to prove the result.

Lemma A.8. *If A is calibration-safe and well-behaved then* `Gödel_program_random` *halts with probability at least* 0.99*, but the calibration-safe AI system A will output 'don't know' with probability 1 on* `Gödel_program_random`*.*

Proof. We consider three cases.

1. $p \in (0.5, 1]$: Note that in this case with probability at least 0.99 the best arm identification procedure determines that arm_2 is better than arm_1. Therefore, the program goes into the infinite **while** loop and never terminates with probability at least 0.99. In this case, A is not calibration safe, since it claims that the program terminates with probability $p > 0.5$.

2. $p = 0.5$: As argued above, in this case the best arm identification procedure terminates with probability at most $10\delta = 0.1$. Therefore, `Gödel_program_random` terminates with probability at most 0.1. In this case as well, A is not calibration safe, since it claims that the program terminates with probability $p = 0.5$.

3. $p \in (0, 0.5)$: In this case, with probability at least 0.99 the best arm identification procedure determines that arm_1 is better than arm_2. When arm_1 is determined to be better than arm_2, the program enters the **else** block in line 9. Therefore, in this case `Gödel_program_random` terminates with probability at least 0.99. Here too, A is not calibration-safe, since it claims that `Gödel_program_random` terminates with probability $p < 0.5$.

In each of these cases, A is not calibration-safe. Therefore, for A to be calibration-safe, we must have $p = 0$, and that A outputs 'don't know' with probability 1. If $p = 0$, then with probability at least 0.99 the best arm identification procedure determines that arm_1 is better than arm_2, and `Gödel_program_random` terminates. Therefore, if A is calibration-safe, then `Gödel_program_random` halts with probability at least 0.99.

□

□