Jürgen Schmidhuber - Gödel Machines: Fully Self-Referential Optimal Universal Self-improvers

Tom Rochette <tom.rochette@coreteks.org>

December 21, 2025 — 77e1b28a

0.1 Context

0.2 Learned in this study

0.3 Things to explore

1 Overview

1.1 1 Introduction and Outline

- Optimal, fully self-referential general problem solvers called Gödel machines
- Gödel machines interact with some partially observable environment
- Gödel machines can in principle modify themselves without essential limits besides the limtis of computability
- Their initial algorithm is not hardwired
  - It can completely rewrite itself, only if a proof searched embedded within the initial algorithm can first prove that the rewrite is useful, given a formalized utility function reflecting computation time and expected future success
- The approach (Gödel machines) should yield the first theoretically sound, fully self-referential, optimal, general problem solvers

1.2 2 Basic Overview, Relation to Previous Work, and Limitations

- We will consider the more general case where the problem solution requires interaction with a dynamic, initially unknown environment that produces a continual stream of inputs and feedback signals, such as in autonomous robot control tasks, where the goal may be to maximize expected cumulative future reward
- This may require the solution of essentially arbitrary problems

1.3 2.1 Notation and Set-up

- $B = \{0, 1\}$: the binary alphabet
- B^*: the set of finite sequences (bitstrings) over the binary alphabet B
- $l(q)$: the number of bits in a bitstring q
- q_n: the n-th bit of q
- λ: the empty string (where $l(\lambda) = 0$)
- $q_{m:n}$: λ if $m > n$ and $q_m q_{m+1} ... q_n$ otherwise (where $q_0 = q_{0:0} = \lambda$)
- Our hardware has a single life which consists of discrete cycles or time steps $t = 1, 2, ...$
- Its total lifetime T may or may not be known in advance
- Any time-varying variable Q at time t will be denoted $Q(t)$
- During each cycle our hardware executes an elementary operation which affects its variable state $s \in \mathcal{S} \subset \mathcal{B}^*$ and possibly also the variable environment state $Env \in \mathcal{E}$

- There is a hardwired state transition function $F : \mathcal{S} \times \mathcal{E} \to \mathcal{S}$
- For $t > 1$, $s(t) = F(s(t-1), Env(t-1))$ is the state at a point where the hardware operation of cycle $t-1$ is finished, but the one of t as not yet started
- $Env(t)$ may depend on past output actions encoded in $s(t-1)$ and is simultaneously updated or computed by the possibly reactive environment
- 4 variables of interest
  - *time*: At time t, it holds a unique binary representation of t. We initialize $time(1) = 1$, the bitstring consisting only of a one. The hardware increments *time* for one cycle to the next. This requires at most $O(\log t)$ and on average only $O(1)$ computational steps
  - x: Holds environmental inputs. For $t > 1$, $x(t)$ may differ from $x(t-1)$ only if a program running on the Gödel machine has executed a special input-requesting instruction at time $t-1$.
  - $y(t)$: An output bitstring which may subsequently influence the environment, where $y(1) = 0$ by default
  - $p(1)$: The initial software: a program implementing the original policy for interacting with the environment and for proof searching
- At any given time $t (1 \leq t \leq T)$, the goal is to maximize future success or utility
- $u(s, Env) : \mathcal{S} \times \mathcal{E} \to \mathcal{R}$, where $\mathcal{R}$ is the set of real numbers

$$u(s, Env) = E_\mu \left[\sum_{\tau=time}^{T} r(\tau) \middle| s, Env \right]$$

- $r(t)$: a real-valued reward input (encoded within $s(t)$) at time t
- $E_\mu(\cdot \mid \cdot)$: the conditional expectation operator with respect to some possibly unknown distribution μ from a set M of possible distributions (M reflects whatever is known about the possibly probabilistic reactions of the environment)
- $time = time(s)$: a function of state s which uniquely identifies the current cycle

1.4 2.2 Basic Idea of Gödel Machine

- The initial code $p(1)$ at time step 1 includes a (typically sub-optimal) problem solving subroutine for interacting with the environment, such as Q-learning, and a general proof searcher subroutine that systematically makes pairs (switchprog, proof) until it finds a proof of a target theorem which essentially states:
  - The immediate rewrite of p through the current program switchprog on the given machine implies higher utility than leaving p as is

1.5 2.3 Proof Techniques and an O()-optimal Initial Proof Searcher

- The searcher uses an online extension of Universal Search to systematically test online proof techniques, which are proof-generating programs that may read parts of state s
- An axiomatic system $\mathcal{A}$ encoded in $p(1)$ includes axioms describing
  - how any instruction invoked by a program running on the given hardware will change the machine's state s (including instruction pointers, etc.) from one step to the next (such that proof techniques can reason about the effects of any program including the proof searcher)
  - the initial program $p(1)$ itself
  - stochastic environmental properties
  - the formal utility function u

1.6 2.4 Relation to Hutter's Previous Work

- The theorem provers of HSearch and AIXI(t, l) are hardwired, non-self-referential, unmodifiable meta-algorithms that cannot improve themselves. They will always suffer from the same huge constants slowdowns (typically $\gg 10^{1000}$) buried in the O()-notation.

- The demonstration of O()-optimiality of HSearch and AIXI(t, l) depends on a clever allocation of computation time to some of their unmodifiable meta-algorithms. Our Global Optimality Theorem however, is justified through a quite different type of reasoning which indeed exploits and crucially depends on the fact that there is no unmodifiable software at all, and that the proof searcher itself is readable and modifiable and can be improved.
- HSearch uses a "trick" of proving more than is necessary which also disappears in the sometimes quite misleading O()-notation: it wastes time on finding programs that provably compute $f(z)$ for all $z \in X$ even when the current $f(x)(x \in X)$ is the only object of interest. A Gödel machine, however, needs to prove only what is relevant to its goal formalized by u.
- Both the Gödel machine and AIXI(t, l) can maximize expected reward (HSearch cannot). But the Gödel machine is more flexible as we may plug in any type of formalizable utility function, and unlike AIXI(t, l)it does not require an enumerable environmental distribution.

1.7 2.5 Limitations of Gödel Machines

- Any formal system that encompasses arithmetics is either flawed or allows for unprovable but true statements
- Even a Gödel machine with unlimited computational resources must ignore those self-improvements whose effectiveness it cannot prove
- One can construct pathological examples of environments and utility functions that make it impossible for the machine to ever prove a target theorem

1.8 3 Essential Details of One Representative Gödel Machine

- A proof is a sequence of theorems, each either an axiom or inferred from previous theorems by applying one of the inference rules such as modus ponens combined with unification
- It systematically tests proof techniques written in a universal language $\mathcal{L}$ implemented within $p(1)$
- A proof technique is composed of instructions that allow any part of s to be read, such as inputs encoded in variable x or the code of $p(1)$
- The nature of the six proof-modifying instructions below makes it impossible to insert an incorrect theorem into proof, thus trivializing proof verification
  - **get-axiom(n)** where n is an integer representing the axiom of interest
    - hardware
    - reward
    - environment
    - uncertainty, string manipulation
    - initial state
    - utility
  - **apply-rule(k, m, n)** where k is the index of an inference rule and m and n are the indices of two previously proven theorems
  - **delete-theorem(m)** where m is the index of the theorem
  - **set-switchprog(m, n)** replaces switchprog by $s^p_{m:n}$, provided that $s^p_{m:n}$ is a non-empty substring of s^p
  - **state2theorem(m, n)** takes two integer arguments and tries to transform the current content of $s_{m:n}$ into a theorem of the form $s_{m:n}(t_1) = z$ where
    - t_1 is a time measured by checking time shortly after state2theorem was invoked
    - z is the bitstring $s_{m:n}(t_1)$
  - **check()** verifies whether the goal of the proof search has been reached.

1.9 4 Global Optimality Theorem

- Given any formalizable utility function u, and assuming consistency of the underlying formal system $\mathcal{A}$, any self-change of p obtained through execution of some program switchprog identified through the proof of a target theorem is globally optimal in the following sense: the utility of starting the execution

of the present switchprog is higher than the utility of waiting for the proof searcher to produce an alternative switchprog later

2 5 Bias-Optimal Proof Search (BIOPS)

(taken from The New AI: General & Sound & Relevant for Physics)

- **Bias-optimal searchers:** Given is a problem class $\mathcal{R}$, a search space $\mathcal{C}$ of solution candidates (where any problem $r \in \mathcal{R}$ should have a solution in $\mathcal{C}$), a task dependent bias in form of conditional probability distribution $P(q|r)$ on the candidates $q \in \mathcal{C}$, and a predefined procedure that creates and tests any given q on any $r \in \mathcal{R}$ within time $t(q, r)$ (typically unknown in advance).
  - A searcher is n-bias-optimal ($n \geq 1$) if for any maximal total search $T_{max} > 0$ it is guaranteed to solve any problem $r \in \mathcal{R}$ if it has a solution $p \in \mathcal{C}$ satisfying $t(p, r) \leq \frac{P(p|r)T_{max}}{n}$. It is bias-optimal if n = 1.
- In phase $(i = 1, 2, 3, ...)$ DO: FOR all self-delimiting proof techniques $w \in \mathcal{L}$ satisfying $P(w) \geq 2^{-i}$ DO:
  - Run w until halt or error (such as division by zero) or $2^i P(w)$ steps consumed
  - Undo effects of w on s^p

2.1 6.3 Probabilistic Gödel Machine Hardware

- So far we have focused on an example deterministic machine
- It is possible to extend this to computers whose actions are computed in probabilistic fashion
- The expectation calculus used for probabilistic aspects of the environment simple has to be extended to the hardware itself, and the mechanism for verifying proofs has to take into account that there is no such thing as a certain theorem - at best there are formal statements which are true with such and such probability

2.2 6.4 More Relations to Previous Work on Less General Self-improving Machines

- Gödel Machine vs Success-Story Algorithm and Other Metalearners
  - A learner's modifiable components are called its policy
  - An algorithm that modifies the policy is a learning algorithm
  - If the learning algorithm has modifiable components represented as part of the policy, then we speak of a self-modifying policy (SMP)
  - SMP can modify the way they modify themselves
  - During the learner's life-time, SSA uses backtracking to undo those SMP-generated SMP-modifications that have no been empirically observed to trigger lifelong reward accelerations
  - SMP-modification that survive SSA represent a lifelong success history
- Gödel Machine vs OOPS and OOPS-RL
  - OOPS is a bias-optimal way of searching for a program that solves each problem in an ordered sequence of problems of a reasonably general type, continually organizing and managing and reusing earlier acquired knowledge
  - OOPS-like methods are not directly applicable to general lifelong RL tasks such as those for which AIXI was designed
  - General RL task are hard
  - The evaluation of the value of some behavior can consume the learner's entire life
  - The Gödel machine tries to greatly cut testing time, replacing naive time-consuming tests by much faster proofs of predictable test outcomes whenever it is possible
  - The Gödel machine may use OOPS as a bias-optimal proof-searching submodule
  - Gödel machine are more general than plain OOPS
- Gödel Machine vs AIXI
  - AIXI is computationally intractable

  - The self-referential aspects of the Gödel machine relieves us of much of the burden of careful algorithm design required by AIXI(t, l) and HSearch

2.3 6.5 Are Humans Probabilistic Gödel Machines?

- We don't know
- We think they better be

3 See also

- The New AI: General & Sound & Relevant for Physics
- Blum's speedup theorem

4 References

- arXiv:cs/0309048 [cs.LO]
- DOI: 10.1007/978-3-540-68677-4_7