Agent Foundations

Daniel C

19th April 2026

Why Agent Foundations?

Aligning a system that doesn't exist yet

- Alignment may eventually require us to steer a system far more capable than anything we have built, under conditions unlike anything we have yet observed
- Agent foundations studies the properties that capable agents tend to share *in general*, rather than the details of any one system – giving us a way to reason about highly capable agents *in advance* of being able to study them directly
- At a sufficiently dangerous capability level, we may need to get key safety properties right on the *first critical try*: a serious failure could be unrecoverable, leaving no room for the trial-and-error that science normally relies on
- Even if we learn about capable agents empirically from weaker systems, we still need a theory of *which* properties continue to hold as capability increases – and supplying that theory is itself the work of agent foundations

Reflective Stability

Self-modification and the stability of goals

- A sufficiently capable agent may be able to: rewrite its own source code, build a more capable successor, and radically revise its world model (e.g. learning new physics, ontology shifts)
- A property is *reflectively stable* if it remains invariant under all such self-modifications
- A safety property we instil only matters if it is reflectively stable: a guardrail is of little use if the agent can self-modify to remove it, or build a successor that lacks it
- For a sufficiently capable goal-directed agent, the working assumption is that **a property persists only if there is some reason it has to** – architectures, constraints, and conceptual schemes are all candidates for being optimised away under enough pressure
- So to guarantee a safety property holds (e.g. "this system does not cause a catastrophe"), it is not enough to instil it once; we need it to be *invariant under self-modification*, and we need to understand what kinds of properties *can* have that invariance

Two Pathways of Impact: Modelling vs. Implementation

Modelling

- Build an *abstract mathematical model* of an idealised, highly capable agent, and use it to make safety claims about how such an agent would behave
- The model need not be runnable; its value is that it lets us reason about such agents *in advance*, and in particular show *why* a given alignment proposal would fail

Implementation

- Develop a theory to the point where it could be turned into an actual algorithm or running system that we could build and inspect
- On this pathway, researchers often favour *modular* architectures – a separate, inspectable world model, a planning module, and an explicit representation of the goal – rather than one opaque network, so that each part can be checked and controlled on its own
- A clean factorisation like this is what would let us point at "the goal" or "the world model" inside the system and verify or edit it directly

(Distinction following C. Wyeth, *Modeling versus Implementation*.)

Reasoning About Ideal Intelligence

Coherence theorems, and their limits

- Coherence theorems (money-pump / Dutch-book arguments) give *some* justification for modelling a capable agent as an **expected-utility maximiser**: an agent with incoherent preferences can be money-pumped, so optimisation pressure favours coherence
- The claim is almost **vacuous** as stated: anything can be cast as a utility maximiser. A rock is an expected-utility maximiser whose utility is maximised by sitting on the floor
- A frame that fits every system equally makes no predictions. For predictive content we must specify what the agent is coherent **over**

Reasoning About Ideal Intelligence

Coherence over resources makes concrete predictions

- Utility maximisation becomes *predictive* once a physical system is modelled as a maximiser over **specific resources** (money, energy, free energy)
- Operational preference: the system **prefers** state A to state B if it expends resources to move from B to A
- Coherence then makes a **falsifiable** prediction: an *efficient* agent does not spend resources to go $A \to B$ and also spend resources to go $B \to A$
- This is **checkable** for a given system and resource. Example: does a cell spend energy converting compound $X \to Y$ and also energy converting $Y \to X$?

Why care about resources?

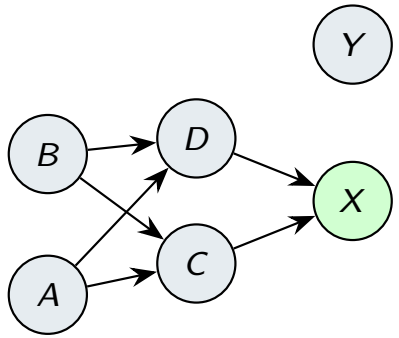
Instrumental convergence

- **Instrumental convergence:** some resources (energy, money, compute) are useful across a *wide variety* of goals
- So accumulating such resources is good by the agent's own lights, whatever its terminal goal
- **Resource theories** in physics formalise a resource as a *monotone*: a state function (e.g. free energy, negentropy) that cannot increase under the free operations
- More of the resource gives more **optionality**: a wider set of future states the agent can still reach

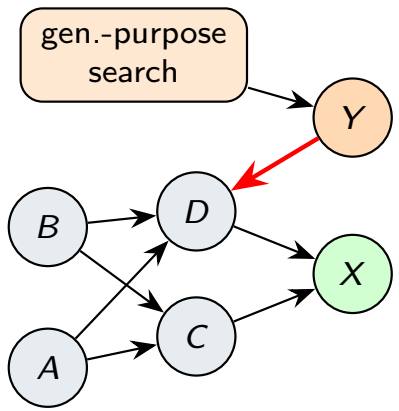
General-purpose search

Why intelligence is "general purpose"

- Why expect a capable mind to be *general-purpose* – able to steer towards a *wide variety* of goals, not just one fixed task?
- Suppose you hold a terminal goal X, but your world model is **incomplete**: some variable Y causally influences X, yet you are not (yet) aware of that pathway
- As you learn and update your world model, you discover the path $Y \to X$ – i.e. Y is revealed to be an **instrumental subgoal** for X. Many variables may "turn out" this way
- Now suppose the agent runs a **general-purpose search**: an internal algorithm that (1) takes *any* goal description and (2) returns a plan to achieve it. Then the moment a new instrumental subgoal is discovered, the agent can optimise for it **on the fly** – it need not have anticipated it



known pathways into X; Y unlinked

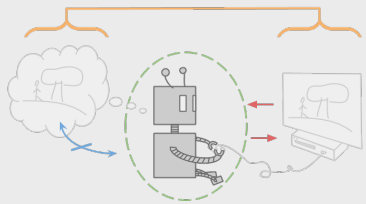


learn $Y \to D \to X$; search targets Y

Dualistic agents

The cybernetic picture: Alexei plays a video game

- In the standard ("cybernetic") picture, an agent learns by building a *model* of its environment, and uses that model to plan towards a goal. This is Alexei, playing a video game:

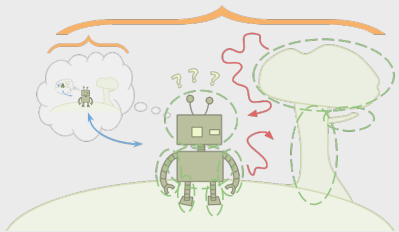


- **Clear input & output channels** between the agent and the environment
- The agent is **"larger" than the environment**: it can hold a full copy of the game inside its own mind
- The agent is **"outside" the environment**: it need not reason about itself, only about optimising the environment
- Standard idealised models of agency – an expected-utility maximiser, or any agent that keeps a full internal model of the world and plans within it – assume this dualistic, "large agent" picture
- But can we still reason about an agent that is *smaller* than its environment, and contained inside it?

Embedded agents

Embedded agents: Emmy plays real life

- This is Emmy. Emmy is playing real life:



- The environment **contains her**: Emmy is just one part of the environment, made of the same pieces as everything else
- **No well-defined input/output channel** cleanly separating agent from environment
- She **cannot hold the environment inside her own head** – the environment is bigger than she is

Embedded agents

Why embeddedness is hard

- The environment may contain **copies of her**, and other equally capable agents that model her while she models them
- She may be **modified, destroyed, or copied** (by herself or by others) between interaction steps
- Being inside the environment she manipulates, she is also capable of **self-improvement**
- Dualistic models struggle to represent this: a model that treats the environment as simpler than the agent itself cannot contain a copy of the agent, so it cannot describe an agent that is part of its own environment
- We would like a theory of consequentialist agency that is consistent with this embedded setting

Self-Modification and Vingean Reflection

Studying self-modification in a simplified setting

- We want to study **self-modification**: a sufficiently capable agent A_1 may rewrite its own code or build a successor A_0 more capable than itself (self-improvement is a special case of building a successor)
- Strategy: reduce this to a simpler setting we can analyse cleanly, but whose lessons should carry over to richer settings
- **Vingean uncertainty.** Here "more capable" means A_0 makes more accurate predictions, or selects more effective actions towards a goal, than A_1. So A_1 *cannot* predict exactly what A_0 will do – if it could, it could simply take those actions itself and be equally effective, and A_0 would not be more capable after all (cf. Deep Blue's designers knowing it was "trying to win" without knowing its moves)
- **Implication.** A_1 must establish, in an *abstract* sense, that building A_0 is safe – without simulating A_0 step by step

The tiling principle

- We want A_1 to be able to approve building a successor *similar to itself* – e.g. the same algorithm running on different hardware
- This substrate-independence is the point: we want a *robust* abstract argument ("constructing this successor is safe"), not a fragile rule that only blesses agents with identical source code

Self-Modification: Why a Logical Framework?

Stripping away empirical uncertainty

- A real-world agent also has **empirical uncertainty**: which environment am I in? how will my actions affect it?
- To isolate the self-trust problem, we *strip this away*: assume the agent already knows its environment and how its actions influence it
- For simplicity, assume both the agent and the environment can be represented as **programs**

Why proofs about programs are the right tool

- **Concrete behaviour becomes provable.** If a program halts and returns an output, L can prove it – the proof just lists the execution trace
- **But proofs also certify general behaviour.** One can prove a program never halts, or prove a universally quantified claim ("for all inputs, the output satisfies X") – things no single finite trace can show
- So once agents and environments are programs, proofs are *strictly more general* than simulation: they let us certify a property *in the abstract*, without inspecting the execution step by step – exactly what Vingean uncertainty demands

How can A_1 trust A_0?

From "A_0 acts on its proofs" to "A_0's actions are safe"

- A_1 and A_0 share a satisficing goal $\mathcal{G}$, and A_0 only takes actions it can *prove* achieve $\mathcal{G}$
- A_1 inspects A_0's source code and verifies: if A_0 is constructed, then any action A_0 takes was proved (in A_0's proof system) to achieve $\mathcal{G}$:

$$A^1 \Vdash \bar{A}^0 \rightarrow (\forall b_i \in Acts^0 : \bar{b}_i \rightarrow \Box_0[\bar{b}_i \rightarrow \mathcal{G}]) \qquad (2.3)$$

($\bar{A}^0$ = "A_0 is constructed", $\Box_0[\phi]$ = "a proof of ϕ exists in A_0's system")

- But this only tells A_1 that A_0 *acts on what its proof system certifies* – not that what it certifies is actually *true*
- To close this gap, A_1 needs to trust A_0's proof system: whenever A_0 proves something, it is actually true

$$A^1 \Vdash \forall b_i : (\Box_0[\bar{b}_i \rightarrow \mathcal{G}] \rightarrow (\bar{b}_i \rightarrow \mathcal{G})) \qquad (2.4)$$

- Combining (2.3) and (2.4): $A^1 \Vdash \bar{A}^0 \rightarrow (\forall b_i \in Acts^0 : (\bar{b}_i \rightarrow \mathcal{G}))$
  "If A_0 is constructed, then any action A_0 takes achieves the goal"

The Löbian Obstacle

The problem of self-trust

- When A_1 and A_0 use the *same* proof system L, (2.4) becomes $L \vdash \forall \phi : (\Box \phi \rightarrow \phi)$: "whatever I can prove is true" – L vouching for its own proofs
- This conflates two distinct things. $L \vdash P$ means a concrete proof of P is in hand. $\Box P$ is the *internal* statement "a proof of P exists", which L can establish abstractly (e.g. by reasoning about a proof-search program) *without exhibiting* any such proof
- **Löb's theorem.** For any statement C: if $L \vdash \Box C \rightarrow C$, then $L \vdash C$. The only statements L can vouch for in the abstract are ones it can already prove outright
- E.g. self-trust would give $L \vdash \Box(2{+}2{=}5) \rightarrow (2{+}2{=}5)$, and Löb's theorem then forces $L \vdash 2{+}2{=}5$ – impossible for a consistent L
- So a consistent system cannot trust its own proofs *in the abstract*; it can only trust a proof once it has *witnessed it directly*

The Löbian Obstacle

Consequence for tiling agents

- One workaround: A_1 can trust A_0's proofs if A_0 uses a *strictly weaker* proof system than A_1 (a stronger system can trust a weaker one)
- But then each successor in a chain must use a weaker system than its predecessor – a "telomere" of logical strength that shortens with each generation
- Eventually the chain runs out of trust, so indefinitely long chains of safe self-modification seem blocked. The *tiling agents* programme studies how, and whether, this obstacle can be overcome

Embedded Subproblem: Logical Uncertainty

Empirical vs. logical uncertainty

- **Empirical uncertainty:** "will it rain tomorrow?" – we lack *evidence* about the world; more observations would help
- **Logical uncertainty:** "is the millionth digit of π a 7?" – the answer is already fixed by the axioms of mathematics. We are not missing observations; we are missing *computation*
- Other examples: the output of a program whose source code you already have, or whether a specific large number is prime

Why this matters for embedded agents

- Standard Bayesian reasoning assumes *logical omniscience*: the agent instantly knows all consequences of its beliefs, at no computational cost
- A bounded, embedded agent cannot: it may know every axiom of arithmetic yet still be uncertain about what they imply
- We need a principled way to assign probabilities to *logical* facts, and to refine them as we spend more computation

A Theory of Logical Uncertainty

Analogy with Bayesian probability

- Bayesian probability gives us a *model* of how an idealised agent should hold beliefs and make predictions under empirical uncertainty – even though no real agent ever performs exact Bayesian inference
- We would like the same for *logical* uncertainty: a clean account of how a bounded agent should assign and update probabilities over logical facts

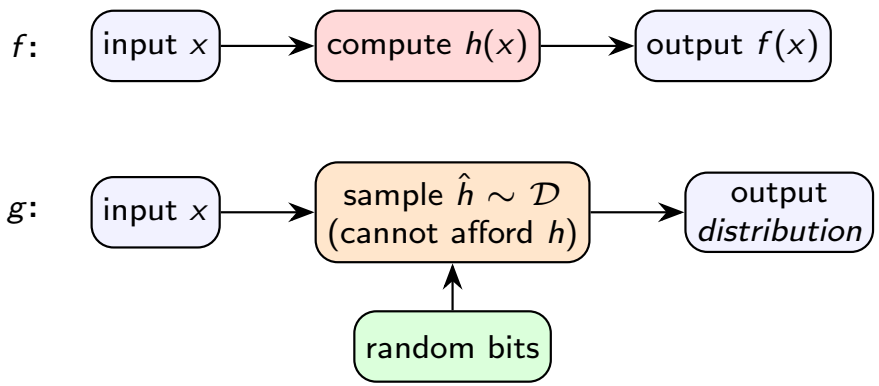
Modelling, then implementation

- Recall the modelling-vs-implementation split: a good *theory* of logical uncertainty (the model) can then guide *algorithms* that approximate it
- This mirrors how the theory of Bayesian inference guides practical, approximate inference algorithms (variational methods, MCMC, . . .)

A Model of Logical (Computational) Uncertainty

Ground truth vs. a time-bounded approximator

- **Ground truth:** a computable function f from inputs to outputs (or output distributions)
- **Approximator:** a program g that receives the input *and* a string of random bits, and must answer within a fixed **time bound**. The random bits induce a distribution over g's outputs; its quality is how close that distribution is to f
- **The tradeoff.** If computing f needs a subroutine h that g cannot afford in time, a sensible move is to use the random bits to *sample* a value $\hat{h}$ from a distribution over h's output, and propagate that uncertainty forward
- As the time bound grows, g can just compute f directly and the random bits become irrelevant – *uncertainty shrinks as we spend computation*



Logical Uncertainty: Open Problems (I)

An algorithm approximating itself

- Give one algorithm A several possible time budgets. With a short budget, A can try to approximate *"what would I output if I had a larger budget?"*, using random bits in place of the computation it cannot afford
- So the same machinery models an agent reasoning about its own future, more-computed conclusions

Randomness as logical counterfactuals

- In the subroutine picture, the random bits build a distribution over a subroutine's output: each draw is a different *possibility* for what that output "could" be
- We need these possibilities precisely because we lack the compute to pin down the actual value – so different settings of the random bits behave like different **logical counterfactuals**

Logical Uncertainty: Open Problems (II)

Which representations are useful?

- Suppose the approximator may also *condition* on some side information about the input. Useful side information lets it reduce its logical uncertainty
- Handing it the output of an expensive subroutine saves that computation, freeing compute to reduce uncertainty elsewhere
- Handing it an *encryption* of the input – carrying the same Shannon / algorithmic information – helps not at all
- So this gives a handle on which *representations* of the same information are actually useful to a bounded reasoner, and on when "post-processing" a representation is worthwhile

Aside: Logical Induction

One concrete realisation of these ideas

- **Logical induction** (Garrabrant et al.) gives a computable way to assign probabilities to logical sentences that improve as more is proved. Picture a *market* pricing each sentence φ in $[0, 1]$ (its probability); a φ-share pays \$1 once φ is proved
- **The logical induction criterion:** no "efficient" (polynomial-time) trader can make unbounded profit against the market
- (This weakens the classical Dutch-book argument – "*no* trader can exploit you $\Rightarrow$ your beliefs obey the probability axioms" – to "no *efficient* trader can exploit you"; that single condition already implies many good properties.)
- In the limit it yields: **coherence** (probability 1 to theorems, 0 to contradictions, monotone under implication); **statistical patterns** (it prices "the nth digit of π is 7" near 10% without computing it); and **self-trust** (current credence = weighted average of its expected future credences)

Descriptive Agent Foundations

From normative to descriptive

- **Normative (ideal) agent foundations:** characterizes how a perfectly rational agent *should* reason and act in principle – starts from desiderata of ideal agents and derives what follows
- **Descriptive agent foundations:** asks what agents we actually encounter in the wild actually *look like* – given any physical system (a neural network, a bacterium, a future AI), can we identify its goals, world-model, and decision-making structure?
- The aim is to develop concepts that apply to a wide range of intelligences, from bacteria to superintelligences

World-centric framing and robustness to scale

- Normative agency starts from the agent's subjective viewpoint (beliefs, preferences, choice); embedded agency asks how that picture maps onto a world where the agent is just another physical subsystem
- Descriptive foundations starts from the other direction: from properties of the *world* (modularity, selection pressures, resource constraints), it asks what processes reliably steer the world into narrow targets
- **Scaling up:** our concepts must not break when the agent becomes vastly more capable – a safety property that fails under extreme optimisation pressure is useless
- **Scaling down:** the same framework should also degrade gracefully when applied to simpler systems

Descriptive Agent Foundations

Physical constraints shape agent structure

- Descriptive foundations emphasize selection pressures, constraints from the physical world, and computational boundedness – these determine what kinds of agents actually arise
- For instance, a key property of the world is **modularity**: the world decomposes into subsystems that interact sparsely. This should make both planning and world-modelling easier for any agent that exploits it
- **World-modelling:** Bayesian networks exploit modularity – instead of maintaining a full joint hypothesis over every possible world, updates about variables are only propagated locally along edges in the graph
- **Planning:** General-purpose search exploits modularity – if the world is modular, an agent can pursue decoupled subgoals independently, rather than having to plan everything jointly

Optimization and Thermodynamics

Optimization as local entropy reduction

- Powerful agents reliably steer the world towards narrow regions of target configurations – outcomes that would be extremely unlikely to arise under any random process
- This is a form of *local entropy reduction*: concentrating probability mass from a broad initial distribution onto a narrow final distribution around a target
- Irreversible transitions: funnelling many initial states into the same targets
- Existing results in stochastic thermodynamics such as fluctuation theorems (Second law of thermodynamics), cost of information erasure and information thermodynamics directly constrains the shape of optimization processes and hold far from equilibrium

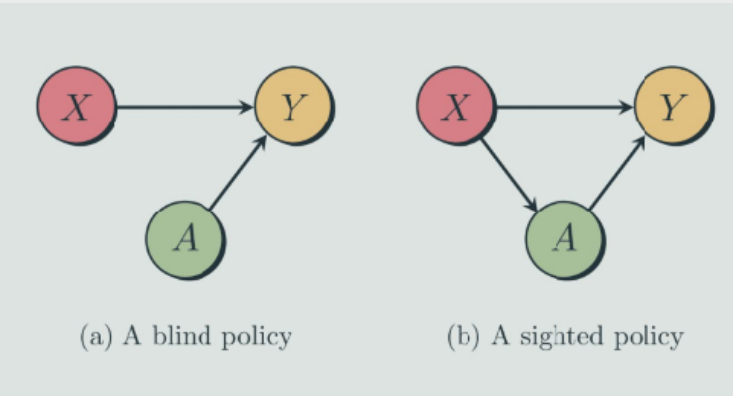
Optimization and Thermodynamics

Even a pure predictor should attend to optimizers

- Even if you only want to predict the system's end state, it is useful to model the optimizers acting on it
- Without an optimizer: you must know the initial condition in detail and integrate the dynamics forward, and prediction error grows over time
- With an optimizer: the initial condition becomes irrelevant (it drives any starting state to its target), and the target itself is the prediction, robust to perturbations
- Paying attention to optimizers is efficient in terms of the information you have to learn vs what you get to predict

Optimization and Thermodynamics

Steering costs information (Touchette & Lloyd)



(a) A blind policy (b) A sighted policy

- A *sighted* agent observes X and chooses action A accordingly; a *blind* agent acts independently of X. The entropy reduction a sighted agent can achieve over the blind baseline is bounded:

$$\Delta H \leq \Delta H_{\text{Blind}}^{\text{Max}} + I(X; A)$$

- The degree to which an agent can steer outcomes beyond what a blind policy achieves is paid for by the mutual information between its observations and its actions – optimization requires information

Optimization and Thermodynamics

From ensembles to individual states

- Traditional thermodynamics defines entropy in terms of probability distributions over ensembles – but where does the distribution come from?
- **Algorithmic thermodynamics** (Ebtekar & Hutter) replaces Shannon entropy H with Kolmogorov complexity K, yielding thermodynamic laws that apply to *individual physical states* rather than ensembles
- Intuitively, a state in which all particles are concentrated in one location would have low entropy, because the repeated coordinates can be printed by a short program.

Knowledge as an endogenous physical resource

- In the Gibbs-Shannon framework, a distribution μ is an *exogenous* parameter specifying what the agent knows. But an embedded agent's knowledge must be *physically encoded* in its memory
- In algorithmic thermodynamics, knowledge becomes *endogenous*: it manifests as **algorithmic mutual information** between the agent's memory and the environment
- This plays the same role as the mutual information in Touchette & Lloyd: it is the resource an embedded agent can spend to steer the world into narrow target configurations
- So an agent that *knows more* about its environment can perform *more* optimization – knowledge is the budget for steering

Optimization and Thermodynamics

Why "knowing more" formally means "optimizing more"

- Common intuition: an agent that understands its environment better can do more to it. Two bridges make this precise
- **Beliefs → mutual information.** If the agent's memory a encodes a belief q over environments, the true state s has a Shannon codeword of length $\approx \log \frac{1}{q(s)}$, so its description given a is short: $K(s \mid a) \leq \log \frac{1}{q(s)}$, hence

$$I(a; s) = K(s) - K(s \mid a) \;\geq\; K(s) - \log \frac{1}{q(s)}$$

- **Mutual information → optimization.** By (the algorithmic) Touchette & Lloyd, the agent can reduce the environment's entropy by up to $I(a; s)$
- **Together:** assigning more probability to the *true* environment ⇒ shorter description ⇒ more mutual information ⇒ more optimization. *Knowing more is being able to do more* – now a theorem, with a fixed exchange rate