Inner alignment via circuit simplification

Abstract

(Same as in the submission form)

The idea consists of a way of creating potentially scalable explanations ("approximate proofs", like what ARC was trying to do) of why the circuit formed by a learned world model + utility function + RL agent has the output distribution it has. Then the explanation and the agent would be jointly optimized to be able to show that the agent achieves a high expected return in a set of situations much more varied than the distribution of the training data.

It would start by taking the temporally unrolled circuit mapping random variables to utilities (for now for a finite number of time steps, but this seems possible to fix). Then on top of it, it would construct a series of progressively simpler circuits, until it gets to one that is small enough that it can be exhaustively analysed. Each circuit maps from different sets of random variables to the same utilities. Given a sampled assignment of values to the random variables of the base level (full circuit), we also have learned functions that map (abstract) a subset of the variables of the computation on that level to their corresponding values on the next simpler circuit. And this is repeated until we get to the simplest circuit. Each level is divided into potentially overlapping sub-circuits that correspond with bigger sub-circuits on the level below (bigger). The IO of the bigger subcircuit is mapped to the IO of the smaller one. The bigger subcircuit is still small enough that we can test exhaustively that as long as the abstraction on the input holds with the smaller one, the output abstraction also holds. And therefore the different levels are equivalent.

Which implies that as long as the inputs to the small subcircuits are within the distribution they have received in the past (which can happen far from the global data distribution), the AI will behave well.

If this is confusing, there are drawings in the attached document showing how this could work for tabular agents in MDPs, consequentialist agents, and "hierarchical actors".

Potential problems with this idea include: looking like a hard optimization problem at which we can't simply throw gradient descent, needing to have a world model and utility function (more on this in the document), assuming that the environment has a certain structure, and limiting the set of possible agents to those that can be explained by this method.

This is a resubmission. If you've read the original, there isn't much new here, only a few edits, sorry. I spent too much time thinking about unrelated ideas that ended up going nowhere instead of working on this, so it's still very unfinished and far from being a concrete algorithm.

Table of Contents

Introduction

If we are in a situation where:

- The agents we create that work by understandable means aren't powerful enough
- Powerful enough agents we create perform the hard part of their optimization inside magical black boxes
- These black boxes are complex enough that we can't simply use some clever trick to cleanly isolate their utility function and substitute it with our own[1]

and we want to come close to *guaranteeing* alignment, we are probably forced to automatically verify them.

For a subset of agents, including those entirely written by humans, we do have actual explanations of why they work, that vary from one to the next (even if for most of them it is some variation of search). Those explanations are ***the*** reason why they work, and I don't think we should expect there to be another one. If the way we went about verifying agents was by, say, designing better SAT solvers, we would need to make sure that for at least the understandable cases our SAT solver is capable of doing something that roughly follows the known explanations. Otherwise, with no other

1 There are other approaches, you could, for instance, have the model query an external utility function and train it with different goals so that it relies on it, but things along these lines don't actually guarantee that the model does what we want, it's still mostly a magical black box

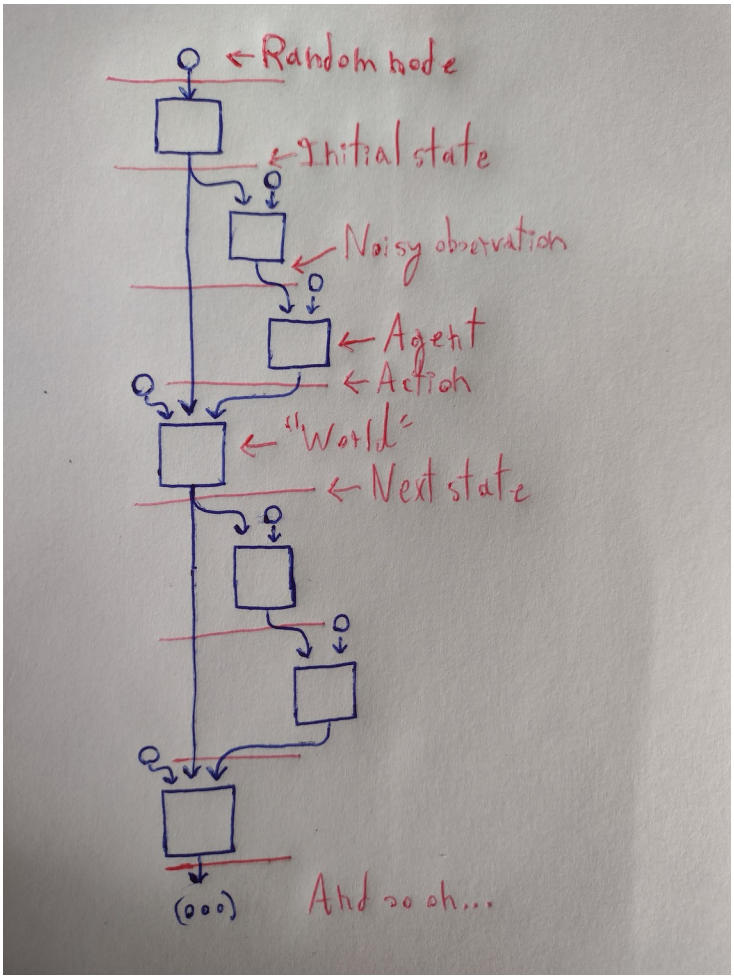
structure to exploit, we will be basically performing brute force search, and for big enough systems we shouldn't expect that to work.

So the way I tried to solve this problem is by looking at understandable algorithms and trying to find a unified method of automatically explaining why they work, hoping it generalizes to all the agents we are interested in.

MDPs

Imagine we have a tabular agent (table from observations to potentially stochastic actions) in a small MDP (with an explicit transition function). As the system evolves through time, it can go through an exponential number of possible trajectories, but since the state space is small, different ones lead to the same states. And in order to compute the expected return it's sufficient to keep track of the probability distribution over states at each time step[2].

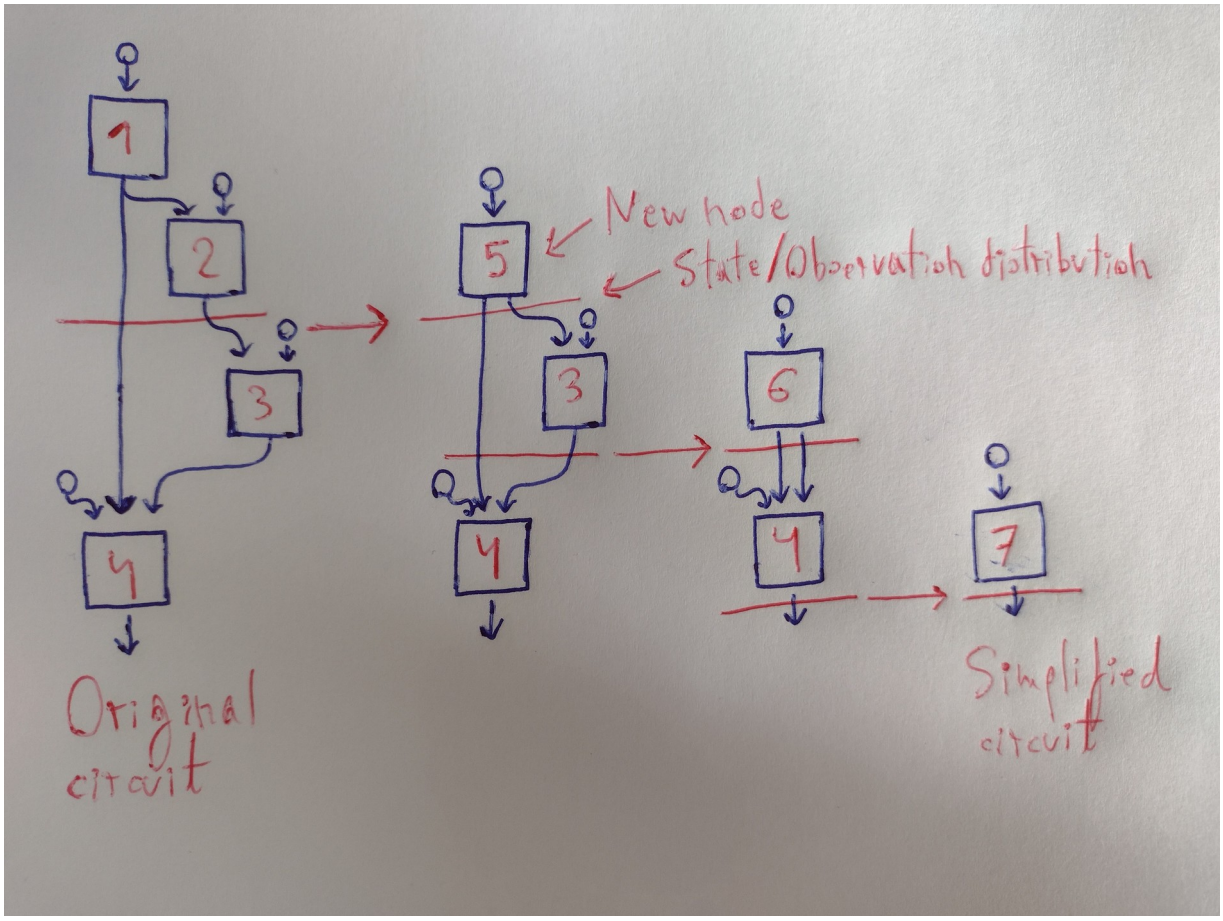
Graphically, we are moving a Markov blanket along the computational graph of the system, where there exists an exact way of representing the probability distribution of the variables in each blanket and a way of checking that each step is valid.



The boxes are deterministic functions. The circles are independent sources of randomness. The red horizontal lines are the blankets.

2 And if the total reward is a linear combination of the individual rewards, we are also using the linearity of expectation in order to avoid having to store the full joint distribution over state rewards

The way I'm dealing with this simple case in a way that generalizes to more complex ones is by considering each step in this process as a different circuit that we are simplifying until we get to a trivial one:

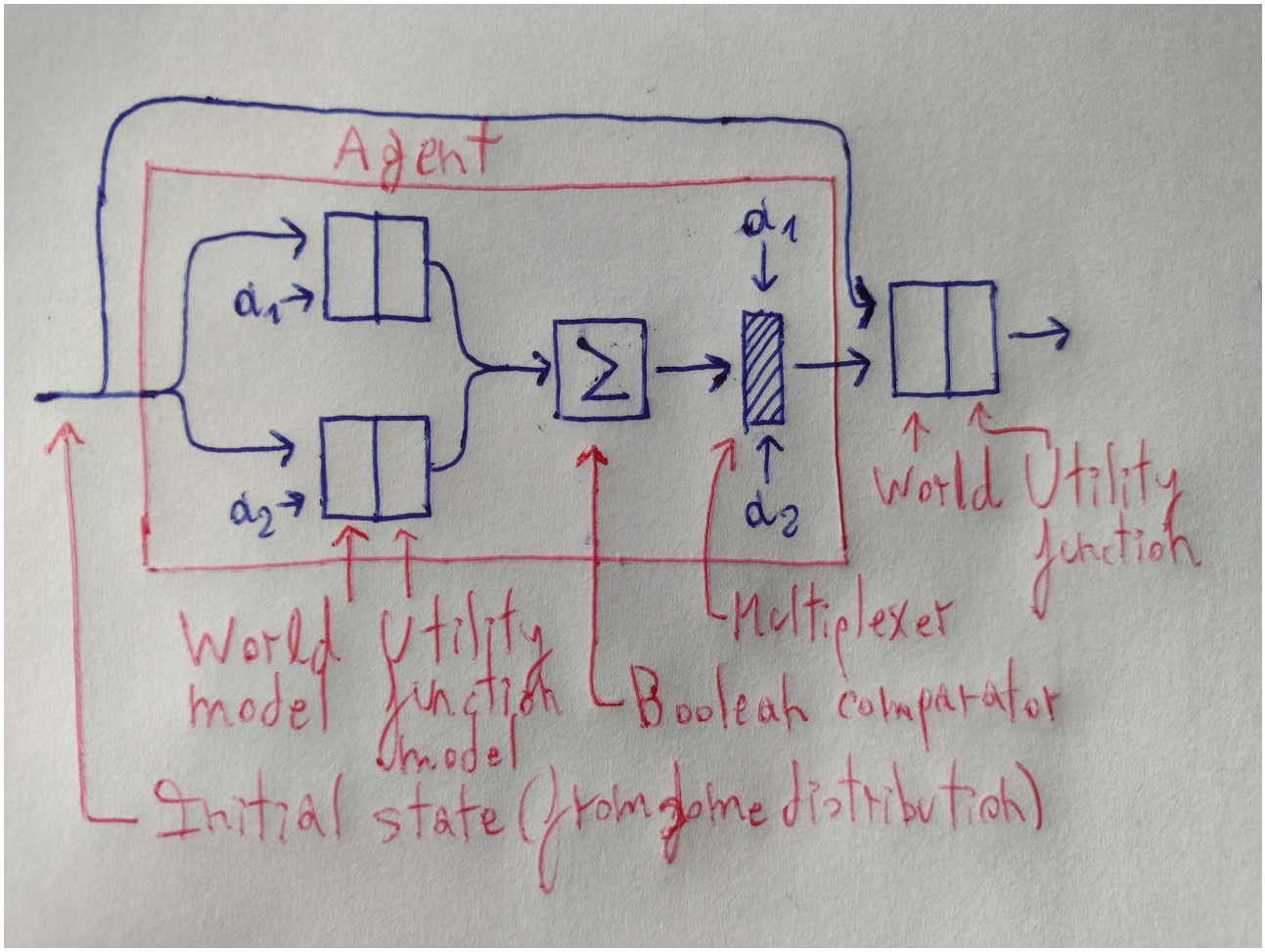


The basic idea is to take the initial circuit, along with a series of initially random circuits, and a final explicit probability distribution over the variable we care about (the reward[3]), and to optimize them so that the transformation between consecutive ones is as close to verifiably valid as possible. More details on this later.

3 The final circuit in the drawing is a model of the state distribution, but if there's a reward associated with each state it's easy to turn it into a reward distribution. A caveat is that this is the reward at a given time step, not the total one, but that too can be fixed one way or another

Consequentialist agents

Consider this simple system:



There's an agent that takes the state of the world as input, computes an action, and sends it to the world. The world takes the initial state and the action as input, performs a series of deterministic computations (that might include future instances of the agent), and gets a final result that is passed to a utility function that returns the output of the whole system.

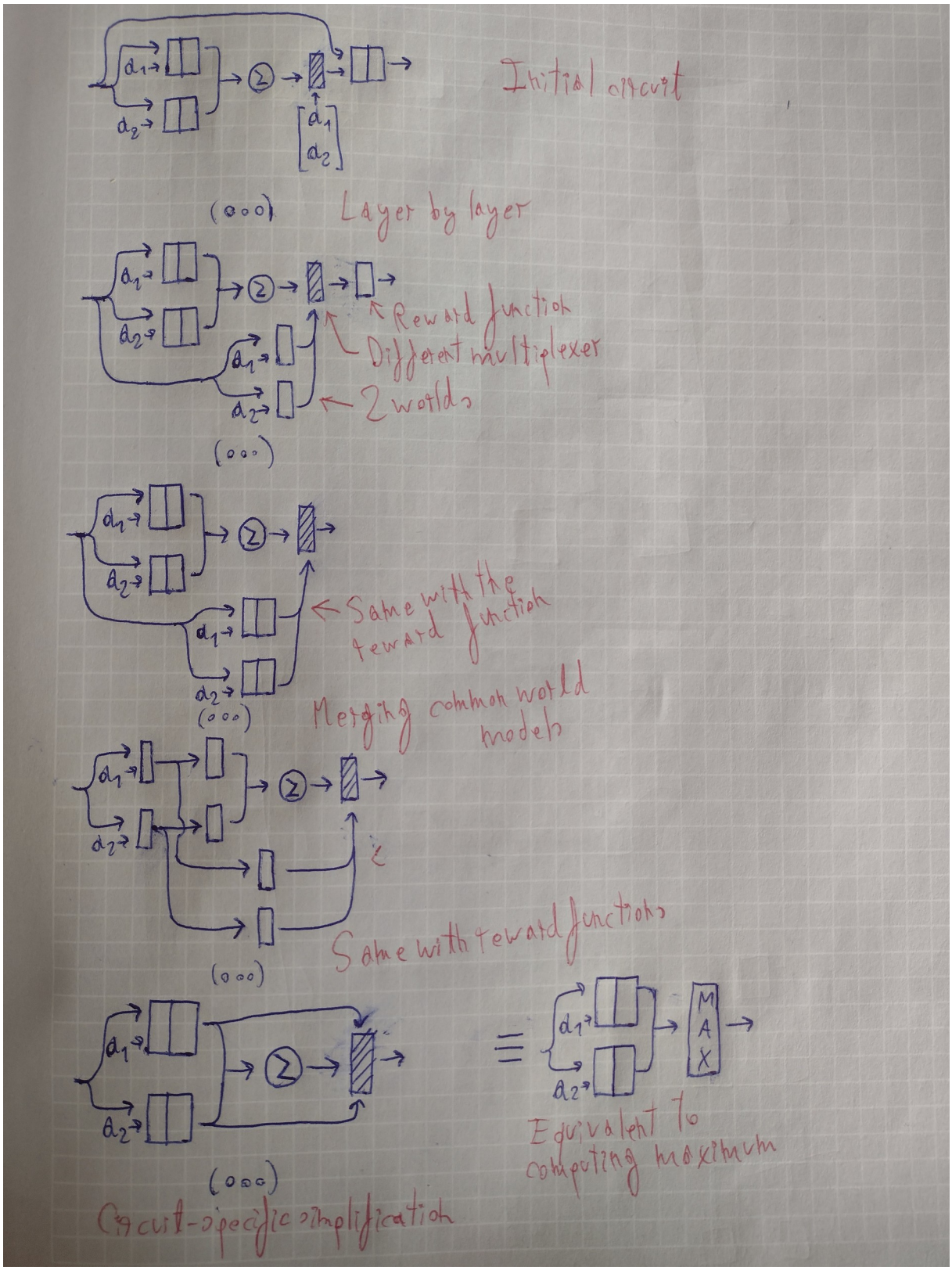
The agent works by considering two actions, simulating what would happen if it took each one, computing the total reward in each case, seeing which is greater, and passing that boolean to a multiplexer[4] that switches between the same two actions.

This agent gets a higher expected return than a random policy because of these facts:

- Its world model and internal utility function correspond with the real ones
- So the outcomes it predicts given each action correspond with the real ones
- Since the actions it chooses between are the same two it considers, the real reward corresponds to the predicted one
- It effectively chooses the maximum of two possible final values, and given *even a very poor model of the joint distribution*, it is possible to tell that this is a relatively high value

4 It's not the standard symbol because that would have made the drawing uglier

It turns out that this reasoning also corresponds with a sequence (sequences actually, it's not unique) of gradual transformations:



What matters about this sequence is that the validity of each step save the last one (getting the distribution of the maximum of two values) can be checked ***locally*** (we only need to consider small regions of the circuits) and maybe even exhaustively without exponential blow-up.

Now imagine that we do this:

We jointly optimize the part of the base circuit we can control (each instance of the agent) and the full sequence of transformations, so that:

- The agent gets a high expected return according to the final circuit
- Each step is as close to verifiably valid as possible, i.e. we show correspondences between small subcircuits
- The final circuit is very small

And imagine that the agent we found was the one drawn above.

The “real” and internal utility functions are merged in the effort of trying to simplify the circuit.

And to make the steps valid, this merged utility function must correspond to both base utility functions.

So the internal utility function will match the external one.

And the system will be inner aligned.

Local validity checks are meant to work something like this[5]: imagine we have two identical boolean circuits with many inputs, and that we want to verify they are identical. We might have a lot of test cases and they behave identically in all of them, but the input space is so big that there is no way of actually knowing that this is always the case. But we could notice (or have a program notice) that there is a way of matching the logic gates (or bigger subcircuits, but still smaller than the whole) of one circuit with those of the other so that the topologies match. And given each subcircuit pair we can check for every single input vector that assuming the inputs are the same, the outputs are also the same. And this implies that the circuits as a whole are identical. It would be a bit trickier if we are working with continuous values and linear combinations, etc. but the basic idea is the same.

We could get the test vectors for sub-circuits empirically or by any other method we want, and our system would remain inner aligned as long as the individual sub-circuits receive inputs that are in that distribution, which can happen far from the global distribution of network activations.

Abstraction and hierarchical acting

This aspect of intelligence is handled even more naturally by circuit simplification.

The space of all possible action sequences is too big to search directly, so the trick I’m almost certain people use internally is to plan hierarchically[6]. For instance, if someone is sleepy and goes to fetch some coffee:

- They will notice that they are sleepy and that getting some coffee will help
- They will remember where the coffee is and plan their rough trajectory around obstacles to get there

5 But not actually like this, more details in the next section

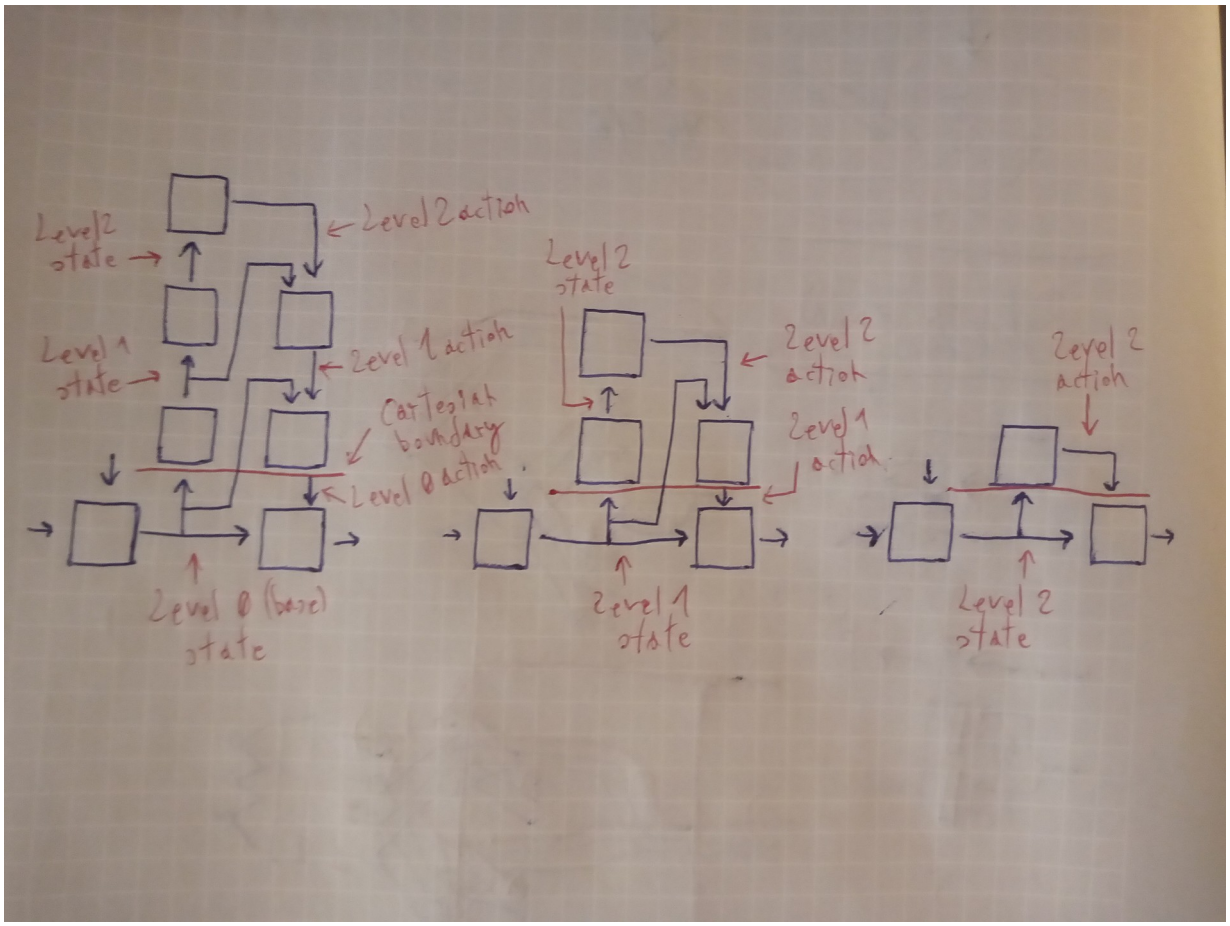
6 I say hierarchical acting, which might sound a bit weird but hierarchical stuff doesn’t necessarily involve planning, it can be reactive

- For each segment of the trajectory, they will decide to take some steps
- Each step is made by sending signals to the muscles to move accordingly

Each level except the last one works because the relevant dynamics of the world are well approximated by a simpler abstracted model:

- If they are in a sleepy state, they can take an action (“getting coffee”) that when sent to some world model (that doesn’t need to be explicit) will result in them getting to the state of not being sleepy. And this is approximately independent of the low-level details of reality: getting coffee will usually result in not being sleepy regardless of what is actually done to get the coffee.
- Going from one point to another can be done by modelling the world as a graph where we can take actions (“moving to this nearby point”) that when executed will result in being in that next point. And we usually don’t need to think about individual steps when doing this.
- And the same happens when taking those steps.

This can be well represented as a series of computational graph simplification steps:



An agent that models the world as having 3 levels of abstraction. With modules that abstract the world and others that take actions. Notice the Cartesian boundary and how the external world becomes more abstracted at each step.

And so as long as we can show that each step of abstraction is valid, we understand how the agent is capable of achieving its goals (and what kind of goals this kind of agent can even have[7]).

Fetching coffee is just an example, but I believe this is the mechanism behind practically everything we do. Another interesting example is language: we can predict what will happen next in a novel, and dialogue between characters is meaningful, because there is a level of abstraction of the dynamics of the world (the one used in language) that works despite all the details not included in the novel.

This is also necessary for realistic planners and agents based on table lookup, e.g. the simulator of the consequentialist agent described earlier might work at a different level of abstraction than the external world model, so we might need to abstract the external model first.

Viewing each step as an abstraction (even if they don't always look like normal abstractions[8]) makes it clear what we need to check: in the deterministic case, if we have a base subcircuit f_base and an abstracted subcircuit f_abs, we need to make sure that for every input X to f_base, the previous subcircuits guarantee that there is a (learned) function t_in such that the input to f_abs is t_in(X), and f_abs(t_in(X)) = t_out(f_base(X)), with t_out being the preconditions for the next subcircuits, which for the output to the whole circuit we force to be the identity.

Extending this in such a way that:

- It works with stochastic functions and we get a single global loss function
    - Or chaotic situations that need to be modelled as random
- Is compatible with some optimization algorithm and scales well to big circuits
- Considers and tries the reduce the "size" of the inputs to the functions
    - So that we can get closer to exhaustive checks
    - This might even include optimizing the world model so that it becomes sparse or modular

is probably the hardest part of all this, but there's probably, somehow, a way of doing it or something similar.

Other kinds of agents

It was a bit surprising to find that the first 3, apparently very different, mechanisms I was analysing could probably be explained using the same method. Now, these are just 3 and they are very simple, so I could be seeing a pattern where there is none. And there are many other mechanisms that agents can use to act coherently in the world. To name just a few:

- Using sampling to estimate value functions
- Using domain-specific heuristics, like in chess
- Memory and learning

7 I didn't draw a utility function connected to the outside world, but notice that if I had, it would need to be able to be abstracted into taking high-level inputs in order to survive until the last step

8 People mean many different but related things by the word "abstraction", here I mean it in the same sense as used in state abstraction in RL or as described in this post

- Exploration
- Hierarchical vision, from edges to parts to objects
- Arbitrary reasoning using language
- Combining multiple of these methods at the same time

Some of these are very complex or mysterious, which makes it hard to think clearly about how far my idea actually generalizes (who knows how language processing works?). But it's not hard to imagine how, for instance, we could abstract the dynamics of the world at the same time we discard the corresponding low-level features of the AI's visual system[9], or how there could be some way of extracting explicit circuits for algorithms that are indirectly encoded in language, that we could then simplify. In any case, while being able to explain the simple cases might not be sufficient, it is probably necessary for any verification-based AI and gives us information about what a more general system could look like. Unless it turns out that the thing we need to prove is arbitrary theorems[10], and that we can't restrict that space without making the prover useless. If that's true I don't think there's much we can do. Let's just hope that the space of real-life intelligent systems is more limited than that, and that we can extend my idea into some kind of theory of compute-bounded agency without needing to add too much additional machinery.

AIXI vs CoinRun

You might argue that even if we could somehow prove alignment with perfect mathematical rigour, it wouldn't completely address goal misgeneralization, because our explanation is only as good as our world model, and it could be very wrong out of distribution. For instance, if we had a world model trained on CoinRun, it would always place the coin to the right, and we could prove that the agent will perform well if this is true, but if the goal position is randomized, the assumptions, and therefore the proof, fail. I expect[11] that if it were possible to magically extend circuit simplification to create exact proofs in complex systems, it would perform well in practice, because when proving local equivalence, we are also showing subcircuits are equivalent for inputs that might be impossible to get. But for the sake of the argument let's assume that fails, then what do we do?

AIXI is optimal almost by definition, if we were to put it in CoinRun and after some time randomize the coin locations, it might take some time to discover the actual rules, but it would perform well. It's possible to know this because:

- Optimality is proven with respect to some distribution of environments, not some concrete environment. Or, equivalently, with respect to the predictions made by a Solomonoff inductor conditioned on the observations it generates.
- The rewards it gets are well-defined in any possible situation.

So if we want to prove our AIs are aligned, we need to include the learning dynamics in our analysis. Thankfully for us, LLMs have shown that we can expect in-context learning to happen by default in world models (and probably in big agents too), by being implemented in their memories. We would need to extend simplification algorithms to work with variable-size circuits (long-term

9 IIRC finding correspondences like this was part of ARC's plan with their heuristic explanations

10 Or even worse! Arbitrary approximate theorems!

11 Note written on the final revision before submitting: I don't know why past me wrote that, and now I'm too sleepy to think about it so I defer to him, but I don't think it matters much either way

memory), but that seems possible to do. This also seems good for dealing with the definition of goals in new situations: when a transformer learns something by keeping it in its memory, the rest of its circuitry stays the same, so if we had some goodness evaluator attached to it, it would probably keep working. And it looks like the same reason why humans don't go through ontological crises: learning new facts doesn't actually update the inner workings of the mind (or at least whatever world-model-like thing we have), it just gets processed and stored like any other piece of data.

Potential problems

Need for explicit world model and utility function

Needing an explicit world model and utility function is additional complexity that other approaches to goal misgeneralization might not need[12], but unless we somehow exploit some peculiarity of human values (as I think, but I'm not sure, the shard theory people are trying to do, for instance), it seems inevitable in satisfactory solutions[13]. If we want an AI to be inner-aligned, there should probably be some explicit external value system to align it to: human feedback, for instance, isn't explicit and so it doesn't make much sense to talk about some system being inner-aligned to it. And the only way I can imagine of directly computing that system is by somehow detecting agents (or some other class that separates humans from most other stuff in the world) within a world model[14] and then getting their values.

I have some hope that if we can find some relatively simple[15] structure representing why the actions an agent takes tend to move the world in some direction, we will be closer to being able to compute what it's trying to do. So instead of needing a separate outer alignment algorithm, it might be possible to extend circuit simplification to do it for us, maybe with a definition of agents and values along these lines: Given a world model, (±?[16]) values are simple functions of the model that non-trivially (by unification instead of ignoring) compress large parts of it around an agent.

But I still haven't thought much about this. Please don't just implement something like this[17] into your AGI, you will die.

Limiting the set of possible agents to those explainable by the algorithm, and environments that can't be explained

If there is some agentic mechanism that works but that this method isn't capable of explaining, it might not be able to use it; and if some part of the environment has some property that can't be explained, it might not be able to exploit it, for instance:

12 And if you are paranoid enough you might worry that now we have to deal with 2 AIs: the agent and the world model

13 Well, there might be some way of proving that some kind of transformation of a human model makes it smarter while keeping alignment. Or some general way of taking two agents and directly checking whether they are aligned with each other, but I doubt it.

14 Let's ignore potential mindcrimes for now

15 Sub-exponential on the size of our graph. Hopefully linear.

16 Or simple functions of values? I need to think more about this

17 Or any existing value learning system, for that matter

- It’s not obvious how to make the AI learn to use language and its contents (for communication or thought) by mainly reading or hearing it, without mathematically showing it is useful first.
- Like AIXI, it might have trouble with non-cartesian reasoning.
- Thoughts about abstract things like mathematics will need to be backed up by explanations of why they are actually useful in physical reality.
- It might think that it is a good idea to go against agents that are smarter than itself because according to its imperfect world model, they will take actions that are worse than the ones they will actually choose.
- If trying some strategy and seeing it works doesn’t update its world model (directly or using in-context learning), then the AI won’t know that it is useful to try the same thing again.

This doesn’t mean that verification-based AIs can’t work, only that we are probably limiting the set of possible agents and that we will need to let empiricism in to some degree. Knowing how big of a problem this is will have to wait until we have an actual algorithm and implementation that we can test.

Capabilities externalities

Alternatively, it might also be possible that this makes AIs much smarter by directly optimizing individual modules and by making sure they generalize better. This is a possible way of solving agent misgeneralization in general, not only goal misgeneralization.

Computational difficulty

Other than looking like a hard optimization problem, there’s also the problem that these circuits will be very big.

In the examples I gave, the abstraction functions are often the identity, so we might be able to exploit this, maybe by somehow merging all the circuits into one.

Then temporally unrolling the world model makes it even bigger. This seems possible to fix, maybe by making a verifier that optimizes a big but finite circuit that takes the initial state and outputs a sample of the final total reward to show that it is equivalent to first running a world+agent time step, computing the local reward and adding it to a temporally discounted sample from the same big circuit when given the new world state (à la Bellman equation but for sampled returns instead of expected values). And then working with either of these two circuits.

But worrying about this before fixing more basic issues seems like premature optimization.

Other things about the proposal

That you don’t need to read in order to understand it.

Relationship with interpretability

Having better interpretability tools might help us understand agents better and therefore know what kind of mechanisms we need to explain. But since circuit simplifications are basically automatic explanations of the relationship between an agent and the environment, they might also serve as tools for mechanistic interpretability.

Two related ideas I tried

The other obvious way of generalizing the Markov process algorithm is to try to propagate more complex data structures along with the blankets. I tried to do it, extending them to try to represent things like “world state and action such that when giving them to this world model it computes a high reward” but it didn’t look like it could scale well and ended up just being an unnecessarily convoluted way of performing circuit simplification. This might be related to something called abstract interpretation, but I didn’t look much into it.

I also tried to see whether I could discover some way of showing optimality regardless of the reward distribution. This seemed like a potentially good idea since expectimax, for instance, is basically optimal by design, but computing the expected return is harder, so it seemed like the natural thing to do. Sadly, trying to think about how close arbitrary systems are to optimality without dealing with actual rewards is hard, and I couldn’t make any progress.

Multiple agents

If we have a system with two agents we could collapse the system around either of them. But the world model of one might approximate reality better than the other. I wonder if there might exist some way of making an abstract ensemble of these models, weighted by the quality of the simplification, and therefore formalize the idea that smarter agents get to choose the future.

Relationship with corrigibility

Some approaches to corrigibility might rely on being able to determine what the user locally (instrumentally?) wants. Think, for instance, about how when a user wants the AI to be shut down we want the AI to follow that even though the user might not have wanted it to happen just a few moments before.

The idea is that if we can get something roughly following the outer alignment strategy described a few sections back, we might (somehow, I don’t know how) be able to compute the intended consequences behind some action, at a level of explanation below that of the whole agent/system.

There are a few things you could try to do with this, but I don’t think any of them will work and I’m not sure they even make sense:

- You could optimize the AI, as part of the computational graph of the world, so that reality (including the user) follows some level of the intended world dynamics, and therefore whenever the user tries to have something happen, the AI will make sure that the intended consequences of their actions happen.
- If you are able to get these local preferences as some sort of temporally varying utility functions, and additionally you are somehow able to compute something like user

preferences about their own future state, you might be able to optimize some weighted sum of the chain of utility functions that starts with the preferences of the present user over their future mental state, and so on recursively, and the preferences of each future user state over the state of the world in their short term futures. This wouldn't actually be corrigible, and the AI will be incentivized to optimize the user, but future user states will be able to try to make the AI do things different from what it was trying to do before, even after fully updating.

Things about which I won't write because this is already taking way too long

This is just a disorganized list of things about which I thought I might write a proper section but in the end didn't. It's probably better than nothing, but I don't promise that any of it makes sense.

- What happens when abstractions break
- How to integrate RL into this
- Details of how to do adversarial training
  - Now it's a lot easier to compute adversarial situations because we've reduced by a lot the exponent of the size of the set of possibilities to test
- OOD avoidance by default
- The consequentialist agent example worked well because we were thinking about a deterministic world and exact correspondences between the external and internal utility functions. If we want some principled way of computing a loss function we will need to deal with things like estimating the value a utility function gives to a situation optimized by another utility function, considering optimization power and without actually performing the calculation for all cases; or with value function verification.
- Frequencies of local vectors at each level of abstraction
- Related work
  - ARC's heuristic explanations were what inspired this proposal
  - Work on causal abstraction and state abstraction
  - Work on formal verification and NN verification in particular
  - Davidad's Open Agency Architecture and Stuart Russell's provably beneficial AI
- Details, benefits, and drawbacks of detecting misgeneralization using this
- Abstraction in cellular automata and physical reality (not enough progress to report so far)
  - If real-life physical abstract circuits "exist" and can be checked using something similar to what is described here, the computational graphs probably need to be sparse. And they might be: the base one due to the locality of physics and the abstract ones because it looks like it's possible to predict some features of the results of interactions of physical objects using only a few of their features. E.g. we can use common sense to know that on a hot day a snowman will turn liquid, knowing only that the snowman is made of snow and that the air is hot, regardless of many features like colour, size, or shape. And we can make a good guess about which of two animals would win a fight, knowing only a few of their attributes like their size and species. If you think about it this is kind of impressive, natural language descriptions of the initial situations refer to an enormous space of possible very big mathematical objects (sidenote: and visual input is compatible with an enormous number of possible low-level states, but the uncertainty about high-level states is lower, possibly making intelligence possible), and yet we can intuitively guess the values of functions of their interactions doing relatively little computation. This deserves some explanation and there is probably something clever you can say about whatever process makes this possible to such a big extent.
- Computing "what the network expects" using the high-level circuits

- Comparison with other methods of avoiding goal misgeneralization
- More about how and to what extent having to deal with hierarchical planning and abstraction in general would force other methods to resemble this one
- To what extent this is an approximate proof, it doesn't even compute bounds
  - What matters isn’t actually verifying the agent, but showing that all the optimization pressure it exerts is in the direction we want
- More about how close it is possible to get to proving things about something as big and unpredictable as reality
- I’m calling this circuit simplification instead of something like alignment via causal abstraction verification because we really are transforming the circuits in arbitrary ways just like we would when simplifying a mathematical expression. I came up with this when I was thinking about the graph of the consequentialist agent, thought that keeping staring at it wouldn’t allow me to progress further, and wrote it down as pseudocode, which I then started manipulating like it was a system of equations and then realized that I was actually simplifying a mathematical expression and that this could be extended naturally to the other 2 cases I was considering at the time.
- Appearance and disappearance of random variables
- The relationship between reality and state estimates might be represented by simplified joint probability distributions
- Exploitation of bugs in the world model
- Detailed circuit abstraction examples and the problems with them:

(moving additional layers of binary operations is a little different, but we only need to check inputs of size up to 5 regardless of the number of values the circuit processes in parallel)