Finding causal theories quickly enough

Building a responsive Apperception Engine

Jean-François Cloutier
Robotics and Embodied project
Active Inference Institute

Last updated: 2023-11-08

Abstract

We describe the implementation of an Apperception Engine designed a distributed cognitive architecture targeting Lego robots. An Apperception Engine applies symbolic machine learning to produce causal theories from sequences of observations. The causal theories are small, human-readable logic programs that model an environment's generative processes and predict incoming observations. Finding these causal theories is a difficult problem largely because of the enormous size of the search space. We built an Apperception Engine that is designed to infer good enough causal theories quickly enough to be useful to a robot actively engaging with its environment.

The role of an Apperception Engine

We implemented an Apperception Engine as part of an ongoing project to animate Lego robots via a distributed model of basal cognition aligned with the principles of Active Inference. The principal goal of the project is to enable robots to learn, without human supervision, how to engage survivably with their physical environment.

In our implementation of a cognitive architecture, we eschew the expected probabilistic approaches of Active Inference and instead employ symbolic machine learning, particularly an Apperception Engine, to realize predictive processing capabilities.

An Apperception Engine is software that examines a history of sensory inputs (observations) and discovers good enough causal theories. A causal theory is a human-readable logic program that makes sense of these observations and can infer future observations from the latest observations.

The cognitive model making use of the Apperception Engine adopts a Society of Mind[1] architecture whereby a robot grows and evolves a collective of self-organizing "cognition actors". The cognition actors are processes under constant threat of extinction and thus

[1] Minsky, Marvin (1986). The Society of Mind. New York: Simon & Schuster. ISBN 0-671-60740-5

motivated to collectively become competent at discovering and extracting the resources they need to survive[2]. To that end, each cognition actor continually strives to make sense of what causes its sensory inputs (observations) and how it should act to bring about desirable sensations/observations.

Each cognition actor is a separate process with its own limited scope of perception and enaction. It senses a small set of signals emitted by its environment which is either the outer world inhabited by the robot-as-autonomous-agent or the inner world populated by other cognition actors. A detailed description of our implementation of a society of mind will be the subject of a future publication[3].

Each cognition actor needs to make sense of its observations (sensory inputs) in order to develop effective action policies to beneficially exploit and explore its environment. A cognition actor calls upon the Apperception Engine to derive and revise a causal theory that explains its more recent observations. Note that any action prompted by a cognition actor also becomes a signal it observes. This causal theory is enlisted to predict incoming signals and also to predict the consequences of possible actions.

To be usable, the Apperception Engine must be able to produce good enough results quickly or else cognition actors might be constantly saddled with obsolete causal theories, possibly leading to their extinction. The Apperception Engine need not discover the best possible causal theories but it *must* operate in "useful time", meaning the causal theories it looks for must still be relevant by the time they are found.

---

[2] A simulated metabolism is implemented in which computations progressively deplete the robot's initial and simulated energy stores. The robot can replenish its stores by positioning itself over different colored areas of the floor representing food (parking itself over one color is not enough). If energy stores become too low, cognition actors are pruned from the robot's society of mind to reduce the computational energy draw. If and when all cognition actors are decimated, the robot effectively dies. The robot survives by learning, before it is too late, how to find the kinds of "food" it needs. Learning is done by evolving a society of mind that is both effective and frugal at perceiving and engaging the robot's outer and inner world. This experimental setup will be detailed in a future publication.

[3] Here's a quick overview: Cognition actors interact with each other by emitting and receiving predictions, prediction errors and action requests. Cognition actors also emit the beliefs they synthesize from their perceptions. These are signals that other cognition actors can sense and make predictions about to form their own, more abstract beliefs. Cognition actors thus form abstraction hierarchies.

Cognition actors also emit "cognitive signals" about fluctuations in their learning and sense making. "Meta-cognition actors" can sense these signals and derive from them beliefs about the "cognitive aptitudes" of other cognition actors. They can then act upon these beliefs to influence the cognitive functions of the individual cognition actors that they sense. Cognitive acts include the retiring of cognition actors and the instantiating of new ones. This cognition about cognition, we argue, realizes a form of basal consciousness. Meta-cogniton actors themselves emit cognition signals that other meta-cognition actors can sense, thus forming "meta hierarchies". Since all cognition actors can form abstraction hierarchies, meta-cognition actors can participate in both meta-hierarchies and abstraction hierarchies. This cognitive architecture is under active research. It is hoped that such a self-organizing society of mind will, through active engagement, be capable of acquiring behaviors of increasing competency needed to survive.

The Apperception Engine, here described, is based on the one documented in “Making sense of sensory input” (2020) by Richard Evans et al[4]. Ours is a re-implementation in Prolog and CHR. Our objective was to achieve good-enough response times so that individual cognition actors can use the Apperception Engine to make sense of their environment *as they interact with it*. Our Apperception Engine was successfully tested on simple yet non-trivial apperception tasks.

What an Apperception Engine does

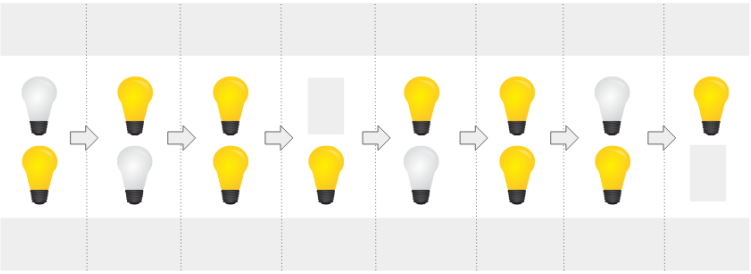
Succinctly put, an Apperception Engine is a program that takes a temporal sequence of observed states and searches for a logic program that, given one state, can infer the next one. The logic program thus embodies a causal theory for the observations; it is a model of the latent generative process at work.

An observed state is a set of simultaneous observations about objects. An observation can be a property of an object or a relationship between two objects. States are snapshots that form a discrete (step-wise) temporal sequence.

To be considered a *valid* causal theory, the logic program must obey a number of unity constraints borrowed from Immanuel Kant’s epistemology. These unity constraints ensure that the logic program “hangs together” and has explanatory power.

To help make our discussion concrete, we will be using the LEDs example from the Evans paper[5].

Consider two visible LED lights. We’ll call them A and B. An LED is either on or off whenever observed. In any given state taken from a sequence of observations, A and B will either be both on, both off, or one on and the other off. If we observe the blinkingLEDs long enough, we notice that they go through a repeating blinking sequence.



Observed states may be incomplete; data might be missing. The state of an LED might, once in a while, be unknown, or there might be other LEDs that are occluded and never observed, or

[4] “Making sense of sensory input”, Richard Evans, Jose Hernandez-Orallo, Johannes Welbl, Pushmeet Kohli, Marek Sergot. arxiv.org/abs/1910.02227
[5] In the paper, the LEDs are called “sensors”. We find LEDs more illuminating.

there might be relationships between the LEDs that are also occluded (for example, they might be wired together out of sight.)

The task of the Apperception Engine is to discover a causal theory that explains why the LEDs blink the way they do. A good theory will identify the LEDs in play (observed or not), the relevant properties of these LEDs (observed or not) and the relevant relationships between the LEDs (observed or not). It will also articulate rules in terms of these properties and relationships, rules that not can predict incoming observations *but also* make sense of them[6].

There are three types of rules in a causal theory produced by an Apperception Engine: Static rules, causal rules and conceptual constraints.

Static rules dictate what conjunctions of object properties and relationships are simultaneously allowed. An example might be that an LED must always be on if the one to its right is off. Static rules check if an inferred state is possible. They can also be used to infer missing observations in an incompletely observed state.

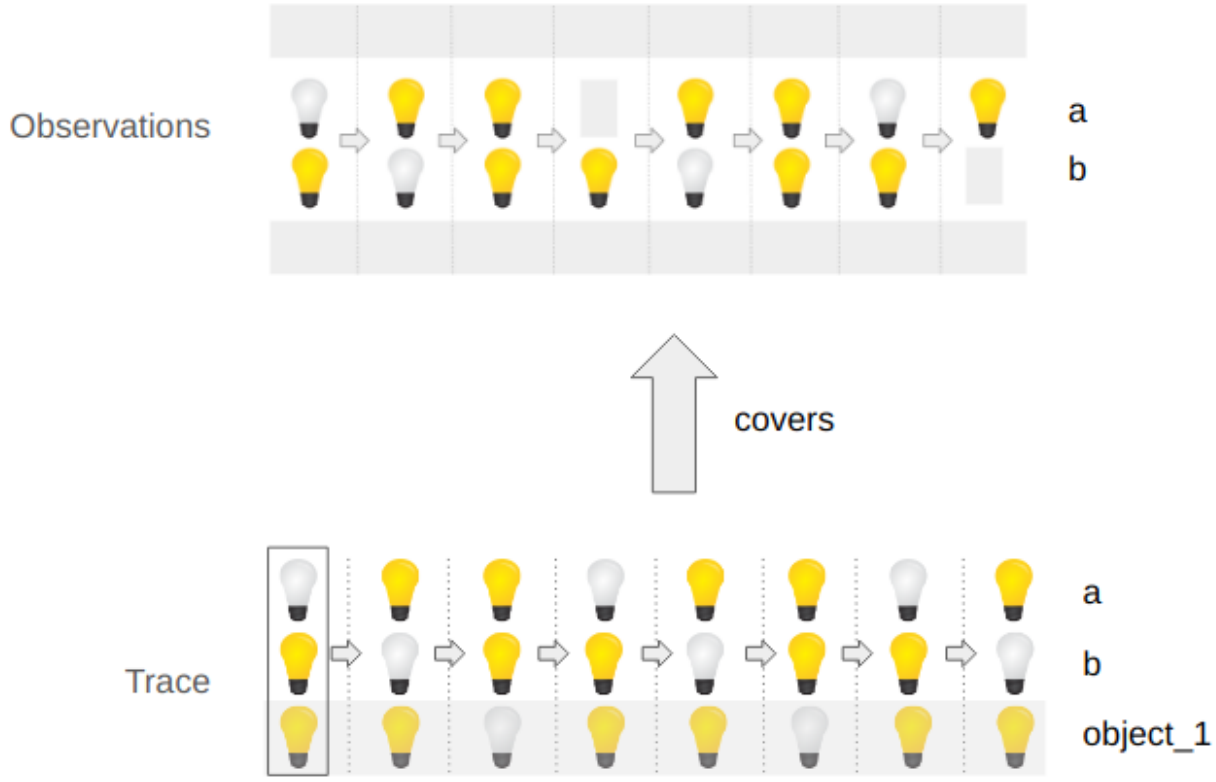
Causal rules dictate what the next observed state will be given a current state. An example might be that an LED will turn on if the one to its left is currently off. Causal rules are used to infer a next state from a current state. If applied to the last observations, they jointly predict incoming observations. If applied to a past observed state, they can backfill the missing state that follows it. So causal rules can both predict and retrodict.

Conceptual constraints are rules on relationships and properties. An example might be that any LED must have one and only one LED to its left, or that no LED can be simultaneously on and off[7].

In addition to rules and constraints, a causal theory identifies initial conditions from which the rules are applied to create a trace. A trace is a sequence of states that, if the causal theory is 100% correct, recreates the observed states, possibly adding missing or unobserved relations and properties that themselves may involve unobserved objects.

A fully successful theory will create a trace that completely *covers* observed states. A not so successful theory will create a trace that only partially covers observed states.

[6] Predicting incoming observations is not, in and of itself, the same as making sense of them.
[7] In our implementation, mutual exclusion constraints on properties are implied.



A causal theory is rated by how well it overlays the observed states and by how complex it is. The fewer and shorter the rules, the less complex the theory is.

The Apperception Engine is looking for a theory that maximizes coverage and minimizes complexity (Occam's razor).

The rules, conceptual constraints, observations and initial conditions of a causal theory are all expressed in terms of a vocabulary of

- object types (e.g. LED),
- objects (e.g. LED 1, LED 2, LED 3),
- property types (e.g. on(some LED, some value),
- relation types (e.g. left-of(some LED, some other LED) and in-front-of(some LED, some other LED)

A "base" vocabulary is implied in the observations that are given as input to the Apperception Engine. For example, if all that is observed are the on status of two LEDs, then the two LEDs and the "on" property make up the base vocabulary.

The Apperception Engine may need to extend this base vocabulary. For example, it might want to add an unobserved LED (*object_1* in the illustration above) and an unobserved relation in order to craft a causal theory that successfully explains observations even though these observations are expressed in a more limited vocabulary.

Even if the Apperception Engine finds a causal theory with perfect coverage and minimal complexity, it will nevertheless reject it if the theory is not *unified*.

A causal theory (initial conditions, static and causal rules, conceptual constraints) is unified if it meets spatial, conceptual, static and temporal unity conditions. The unity conditions are borrowed from Kant's Synthetic Unity of Apperception[8].

Conceptual unity: Every type of predicates (predicates express properties and relations) must be referenced in at least one conceptual constraint. For example, a constraint on the relational "left of" predicate type might be that one and only one LED can be left of another. As for property types of predicates, they are always constrained in our implementation: An object's property can take only one value at once. For example, an LED's "on" property can be true or false but never both simultaneously[9].

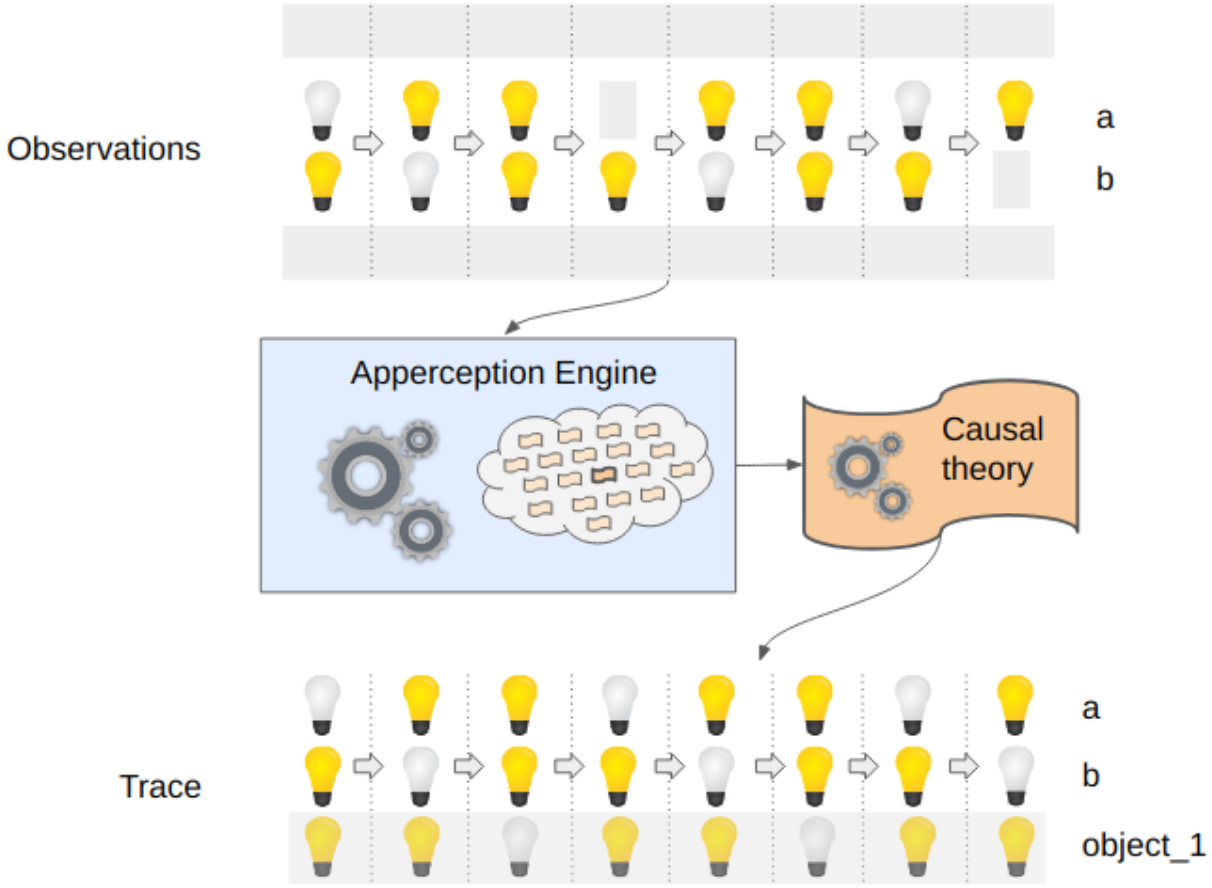
Spatial unity: Each object must be connected transitively to all others via relations. For example, if, in every state of the trace produced by the theory, the LEDs form a connected circle, then each one is related, directly or indirectly, to all others.

Static unity: Observations in a state (in a trace or from observations) must obey all static rules. For example, if a static rule states that an LED must be on whenever the LED to its right is off, then LED A can not be off when LED B to its right is off.

Temporal unity: All states in a trace must be producible by first applying the static rules on the prior state, and then carrying over what remains unchanged. For example, consider a state containing both on/off observations and left-of observations. If causal rules only change on/off states, the left-of observations would be immutable and thus would be carried unchanged throughout a temporal sequence of states.

To restate, the task of the Apperception Engine is, given a sequence of observations (possibly incomplete), to find a unified causal theory composed of initial conditions, static rules, causal rules and conceptual constraints such that the causal theory-as-logic-program produces a trace from its initial conditions that covers as closely as possible the sequence of observations while maximizing coverage and minimizing complexity.

[8] See The Apperception Engine for an in-depth discussion.
[9] The quantum realm is off-limits to the Apperception Engine.



Because the Apperception Engine does not restrict the causal theory strictly to observed objects, properties and relation, a successful theory might involve properties and relationships of hidden objects. For example, it is possible to set up a sequence of observations that only contain the on/off status of 2 LEDs such that articulating a successful causal theory will require the introduction of a hidden third LED plus a hidden relationship between the LEDs.

Requirements

While the functionality of our Apperception Engine is essentially the same as the Apperception Engine described in Evans et al., our performance requirements differ sharply.

The overall goal of the Evans implementation was to demonstrate that an inductive and symbolic approach to machine learning can outperform artificial neural networks on some classes of problems, both in its economy (very little training data is required) and in the predictive capabilities demonstrated after the learning phase.

While some efforts were made to optimize their implementation, it was not a requirement that a causal theory be produced quickly. Finding a 100% successful causal theory with their Apperception Engine might require many hours of computation, perhaps even days[10].

Our requirements are different. Cognition actors will call upon the Apperception Engine while engaging with their environment. They can not afford to wait long to replace obsolete causal theories with ones that better fits the latest observations. Our Apperception Engine needs to find good enough causal theories in useful time, which means seconds, not minutes, and certainly not hours or days.

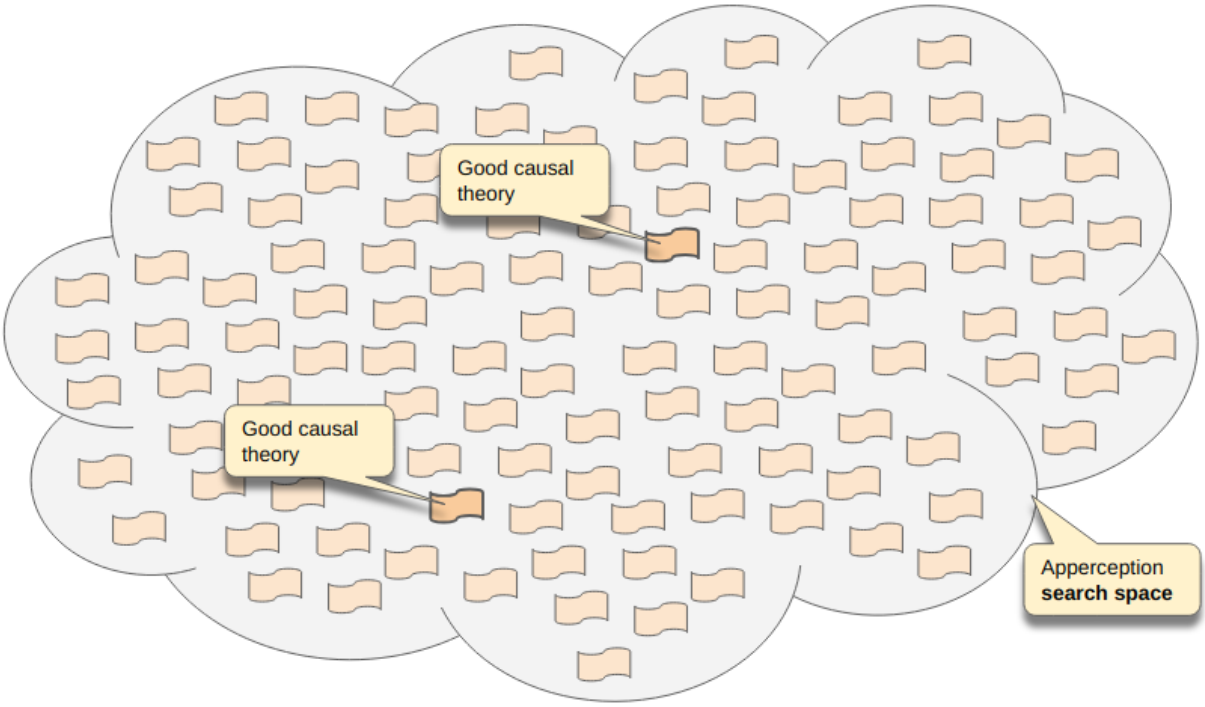
It is important to note that a cognition actor will be restricted to a relatively small “umwelt”; the number of observed objects will be kept small and so will the variety of observed properties and relations. Furthermore, restrictions will be imposed on how many unobserved objects and object types, as well as types of properties and relationships, a cognition actor can imagine (abduce) when formulating a causal theory.

In short, our Apperception Engine is designed to find good enough solutions to relatively simple apperception problems in a matter of seconds.

Apperceiving in useful time is hard

Implementing a program that finds/builds logic programs that regenerates its input[11] is not, in and of itself, a trivial endeavor. But what makes implementing a functional Apperception Engine a truly hard problem is the sheer number of logic programs that are candidates for causal theories and that must be sifted through. This is the proverbial needle in a haystack.

[10] According to documentation in their GitHub repository.
[11] What an Apperception Engine does in generic terms.



If there was an algorithm that could, from a sequence of observations, directly compose step by step a causal theory, then the Apperception Engine could efficiently go directly from observations to causal theory.

There is, to our knowledge, no such algorithm. What we are left with is a search problem.

A simplistic implementation would generate every possible causal theory in turn and test it for validity (unity), correctness (how well the trace it produces covers the sequence of observations) and for complexity (how many rules it has and how big they are.)

The problem is that “every possible causal theory” is a ridiculously large set. Let’s examine why.

A causal theory is composed of initial conditions (a set of simultaneous observations, that is, one observed state), a set of static rules, a set of causal rules, and a set of conceptual constraints.

There are typically a large number of possible permutations of object properties and inter-object relations that can compose the initial conditions of a causal theory.

For every possible initial conditions, there are very many ways in which the rules and constraints making up the rest of a causal theory can be composed.

Rules, whether static or causal, are of the form

```
head if body
```

where the head is either a property or a relation conditionally true of all objects, and the body is one or more properties/relations describing these objects and expressing the conditions.

For example,

```
on(led(X), true) if (left-of(led(X), led(Y)) and on(led(Y), false)
```

How many different individual rule formulations are possible given one vocabulary? A great many. How many combinations of such rules are possible? Many more.

There can be many formulations of conceptual constraints about the relation types in the vocabulary.

For example, every (LED) object has one and only one other (LED) object to its left:

```
one_related(left-of)
```

Or an (LED) object can be either left-of or in-front-of another, but not both:

```
one_relation([left-of, in-front-of])
```

A causal theory could have many different sets of conceptual constraints.

The number of possible causal theories given a vocabulary is roughly the number of possible initial conditions X the number of possible static rules X the number of possible causal rules X the number of possible conceptual constraints.

Clearly, the space of all possible causal theories grows exponentially with the size of a vocabulary and quickly becomes too large to be enumerated and validated one by one.

This is bad enough for one vocabulary. The Apperception Engine searches for a variety of vocabularies, each one extending the base vocabulary (implied in the observations) by adding any number of “imagined” (unobserved) object types, objects and property and relation types.

The search space of all possible causal theories formulated using all possible vocabularies is, for all intents and purposes, infinite.

The generation of a candidate causal theory requires significant computational effort. So does the validation of a candidate theory; a trace must be produced by applying the rules on the initial conditions and then the trace must be checked against the input sequence of observation. And then the causal theory as well as the trace it produces must be checked for unity (spatial, conceptual and static and temporal.)

A simple “generate then validate" implementation would clearly take forever to systemically go though all possible candidate causal theories. Consequently, the odds of finding a valid theory early in the process would be very small indeed.

One strategy employed by the Evans et al. implementation and borrowed in our own is to limit and partition the search space, and to restrict the search to partitions that would produce reasonably small theories.

The Apperception Engine partitions the search space into regions[12], where each region imposes lesser or greater limits on how “imaginative” a causal theory can be, namely how many unobserved objects, properties and relations can be added to the vocabulary implicit in the observations, vocabulary from which causal theories are formulated.

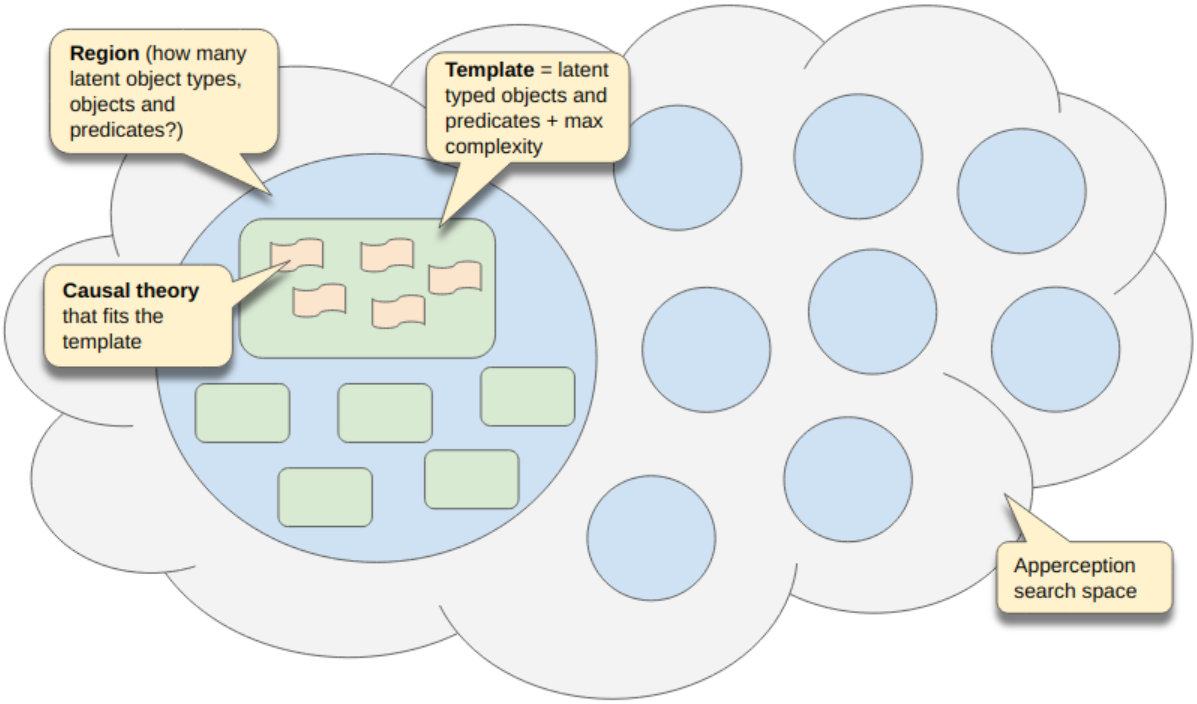
The Apperception Engine does not explore all possible regions into which the search space is partitioned, only those that require not too much imagination (i.e. minimal extensions to the base vocabulary). For example, in the LEDs example, imagining one unobserved LED is sensible but imagining ten probably is not.

There can be many alternate vocabularies that fit within the “limits on imagination” set by a region of the search space, and so each region is subdivided into “causal theory templates”. Each template has its own vocabulary, extended from the base vocabulary, but within the limits set by its region.

A template also sets limits on the number and size of static and causal rules. These limits are based on the vocabulary’s richness.

Each template is searched for successful causal theories expressed in the template’s vocabulary and that do not exceed a maximum complexity.

[12] Regions are called “tuples” in Evans et al.



The benefit of this partitioning and delimiting of the search space is that searches will now terminate well before the thermal death of the universe. The downside is that a satisfactory causal theory might not be found in the time allowed within the allowed regions of an otherwise infinite search space.

Partitioning and restricting the search space helps but it is not enough. We needed to do a lot more if our implementation of the Apperception Engine was to have a reasonable chance of finding good enough causal theories within seconds.

Implementation

Evans et al.'s implementation uses a variant of Datalog (a simple logic programming language) together with a constraint solver library written in Haskell. The emphasis appears to have been on keeping the code as simple as possible, which is desirable from a research perspective where performance is not a core concern.

In our implementation[13] performance is a central concern.

We implemented our Apperception Engine *de novo* in SWI-Prolog[14], augmented by a Constraint Handling Rules[15] (CHR) library.

[13] https://github.com/jfcloutier/andy_karma/tree/main/code/apperception
[14] https://www.swi-prolog.org/
[15] See https://en.wikipedia.org/wiki/Constraint_Handling_Rules

The Prolog programming language, with its declarative, deductive computing model, and its use of backtracking, is particularly well-suited for searching complex solution spaces.

Backtracking is a way for a program to go back to a previous point in its execution where it made a choice that, in retrospect, needs to be changed before restarting there. With this capability, exploration of complex search spaces can be programmed in relatively few lines of code without sacrificing efficiency (quite the contrary.)

SWI-Prolog is an open-source Prolog with support for concurrency on multicore hardware. It enables our implementation to explore in parallel as many templates as the CPU has cores. This greatly accelerates the search.

CHR is a “committed-choice rule-based language embedded in Prolog”. In effect, it adds abductive reasoning to Prolog’s deductive reasoning. It enables a program, here the Apperception Engine, to make assumptions and hold on to them unless and until further assumptions cause CHR rules to refute them.

For example, the Apperception Engine will make the assumption that a causal theory under construction is unified. The assumption holds until a rule is added that contradicts a condition of unification (written in CHR). An assumption, once contradicted, causes a failure that forces the program to backtrack to a point in its execution where the assumption was still held. The program can then resume from that point and go on to add a different rule that hopefully will not break the unity assumption.

This use of abductive reasoning, combined with bactacking on failure, enables the program to fail as soon as possible, pruning searches early, thus avoiding following a search path to a deadend long after it was doomed to fail.

The Apperception Engine entangles three kinds of logical inference.

Prolog programs solve problems by logically *proving* that there are solutions. They do so by applying *deductive reasoning,* from known or inferable facts, using the logic rules that compose these programs. CHR adds *abductive reasoning* to Prolog; it allows programs to assume, during their execution, that some facts are true until ruled otherwise.

The Apperception Engine, written in Prolog + CHR thus uses deductive and abductive reasoning to do what it does. And what it does is *inductive reasoning*; it builds causal theories out of observations (inferring rules from facts).

To more fully appreciate the entanglement, consider that the causal theories built using these three kinds of logic are themselves logic programs that can deduce incoming observations from prior observations.

Tactics and heuristics

What an Apperception Engine does is carry out “apperception tasks”. An apperception task is expressed as a sequence of observations for which a valid, unified causal theory is to be found. Finding a causal theory means searching a vast space of candidate causal theories for one that “works” and is unified, and thus makes sense of the observations given as input.

As mentioned earlier, a naive implementation, one that systematically generates each possible candidate causal theory and then tests it for accuracy and unity, would be grossly inefficient.

In order to find good enough causal theories in useful time (in seconds), we needed to employ a number of tactics and heuristics. The shared objective of all tactics and heuristics is to increase the odds that exploration of the search space will yield an acceptable solution in a small amount of time.

Fail early

If a path taken through the search space is going to fail, it is best to figure this out as soon as possible and abandon that path before any more time and computing resources are wasted.

Our implementation makes use of CHR to check that validity assumptions are contradicted as soon as possible in the process of building (i.e. searching for) a causal theory.

For example, one validity constraint is that a static rule must not be recursive because that makes them tautological. A recursive rule would say something like “An **LED X** must be **on** if there is an LED Y to its left and **LED X** is **on**”. This failure condition is checked while each rule is being built and can be triggered before a rule is done being built. The implementation does not wait for all of a theory’s static and causal rules to be built, or even a single one, to check for recursive static rules, because a lot of work would then need to be undone.

Lower the bar

Looking for the perfect causal theory can be prohibitively expensive whereas good enough theories might be plentiful and thus much faster to find.

Our implementation allows apperception parameters to specify what a good enough causal theory is. For example, an apperception task might be launched such that it is good enough for a causal theory to have a coverage of 80%, meaning the trace it produces covers 80% of the sequence of observations given as input.

As soon as a good enough causal theory is found in some template, the overall search is stopped and the successful causal theory is produced.

By lowering the bar for success, the search can terminate quickly with a useful, if imperfect, causal theory.

Avoid repetition

When traversing a search space, there is a lot of potential for duplication of work. Retracing one's steps is a surefire way of wasting time and resources.

For example, in a causal theory, the static and causal rules are restricted to conjunctions and are of the form "A if B and C and D". If care is not taken, another path through the search space might try the alternate rule "A if D and C and B". Both rules are logically equivalent so only one variant need be tried.

A lot of work went into making our implementation avoid repetitive work.

Throw the dice

There are a lot of choice points when building/searching for a causal theory. Which property or relation to use first in a rule? How many predicates should appear in the rule? How many static rules should the causal theory have? Etc. When faced with a choice, instead of trying options in the same order every time, our implementation randomizes the order in which different options are tried.

Running the same apperception task multiple times does not yield the causal theories in the same order. Sometimes a particularly apt causal theory will be found early in the search, sometimes late. This is better than always late.

Set time limits

The Apperception Engine divides the search space into regions and those into templates. A lot of time could be spent searching in one template, leaving little to no time to explore others. To avoid this, our implementation (and Evans et al.'s as well) time boxes searching each template[16].

In our implementation, the time limit is a function of the size of the vocabulary attached to a template.

When the time allocated to a template is exhausted, the best causal theories found in it are gathered to compete against the theories found in other templates.

An apperception task is also given an overall time limit. Once that time limit is reached without finding a good enough causal theory, the overall search is terminated and the runner-up theories found in the explored search space are produced as a best-effort solution.

[16] Our time limits are expressed in seconds, not hours.

Be selective

A number of heuristics fall under this category. The aim is not to look everywhere in a sprawling search space, but instead to look where, if a solution could be found, it might be found quickly.

Our implementation, and Evans', favors simple causal theories over more complicated ones. For one thing, they are cheaper to build and evaluate. Since the Apperception Engine favors **simplicity** (Occam's razor), it is best to search for simple theories first. This means favoring templates with small vocabularies and with severe restrictions on the allowed number and sizes of rules composing the causal theories.

Furthermore, if a search in a template comes up empty after a short amount of time, our implementation presumes that spending more time in it is a fool's errand, becomes **impatient**, and moves on to another template.

Our implementation does not try every possibility when building/searching for a causal theory. it instead **samples** only a few. For example, when assembling initial conditions (from which a trace will be built by applying the theory's rules), there are a large number of permutations of initial observations to choose from. Our implementation tries only a few under the assumption that they are likely to all produce equivalent traces.

Another way in which our implementation is selective in its search is the use of **iterative deepening**. If the overall time budget is not exhausted after the initial search across templates, the more promising templates are retained, and are searched again, this time with more time allocated to each one. The hope is that templates that produced the better causal theories might produce even better ones if given more time. And because luck is involved (see "Throw the dice"), a new iteration could get lucky and find in a promising template a great theory that eluded prior iterations.

Apperception parameters

An apperception task is accompanied by parameters that limit the search in terms of its scope, success criteria and timeliness.

The parameters are:

- How much time is allocated to the apperception task overall.
- The minimum coverage of a satisfactory causal theory; the search stops when the first satisfactory theory is found.
- How many runner-ups causal theories should be produced if no satisfactory theory is found in the time allowed.
- What are the limits imposed on imagination (up to how many unobserved objects, object types, relation types and property types can the Apperception Engine invent to express a causal theory.)
- How quickly should a new iteration narrow down to the most promising templates when a search completes without success and there's still time left.

These parameters impact the likelihood of quickly finding a satisfactory causal theory. They can be tuned for different apperception tasks.

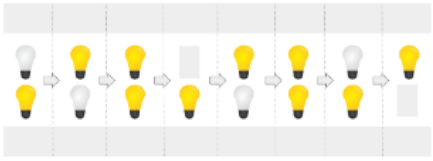
Sample apperception tasks

We present the results of running two apperception tasks. The first task is to explain the blinking pattern of two LED lights (the LED task). The other task is to find the rule(s) of an Elementary Cellular Automaton (the ECA task).

Each task defines search spaces with significantly different properties. The LED task involves few objects (two lights with perhaps another one that's unobserved), possibly connected by unobserved relations. The ECA task involves many more objects (11 cells, either on or off, all observed) connected by an observed relation (every cell has exactly one cell to its left, effectively forming a circle.)

LED task

The Apperception Engine is given a sequence of observations expressed as predicates.



```
sensed(on, [object(led, a), false], 1).
sensed(on, [object(led, b), true], 1).
sensed(on, [object(led, a), true], 2).
sensed(on, [object(led, b), false], 2).
sensed(on, [object(led, a), true], 3).
sensed(on, [object(led, b), true], 3).
sensed(on, [object(led, b), true], 4).
sensed(on, [object(led, a), true], 5).
sensed(on, [object(led, b), false], 5).
sensed(on, [object(led, a), true], 6).
sensed(on, [object(led, b), true], 6).
sensed(on, [object(led, a), false], 7).
sensed(on, [object(led, b), true], 7).
sensed(on, [object(led, a), true], 8).
```

It was then asked to prove that a causal theory exists with these parameters:

```
MaxSignatureExtension = max_extension{max_object_types:1, max_objects:1,
max_predicate_types:1},
```

```
ApperceptionLimits = apperception_limits{max_signature_extension:
MaxSignatureExtension, good_enough_coverage: 100, keep_n_theories: 3, funnel:
3-2, time_secs: 60},
```

With *max_extension{max_object_types:1, max_objects:1, max_predicate_types:1}*, the Apperception Engine is allowed to imagine up to one hidden object type, object and relation or property.

With *good_enough_coverage: 100*, it is asked to be satisfied by a fully accurate causal theory, else to return the 3 runner ups (*keep_n_theories: 3*). The search is given a maximum of 60 seconds to find a perfect causal theory (*time_secs: 60*).

This is the successful causal theory the Apperception Engine typically finds first:

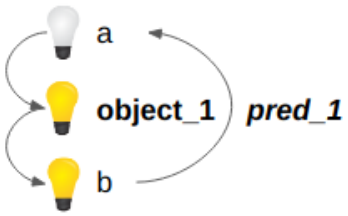
```
static_constraints:[one_related(pred_1)]

static_rules:[on(_A, true)-[pred_1(_B, _A), on(_B, false)]]

causal_rules:[on(_C, false)-[pred_1(_C, _D), on(_D, false)]]

initial_conditions:[pred_1(object_1, b), pred_1(b, a), pred_1(a, object_1), on(object_1, true), on(b, true)
on(a, false)]
```

Interpreting the output a bit demonstrates that the causal theory does provide a cogent explanation of the blinking sequence given as input.



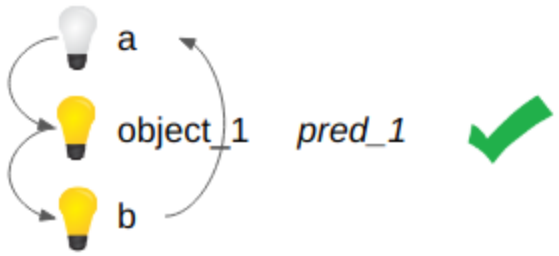
A light is related to one and only one other light via pred_1

A light is on whenever another that is pred_1 to it is off

A light turns off if it is pred_1 to another light that was off

In order to make sense of the observations, the Apperception Engine imagined a hidden LED (it named it *object_1*) as well as an unobserved relationship (it named it *pred_1*) that connects all LEDS together.

The causal theory is unified spatially, conceptually, statically and temporally.



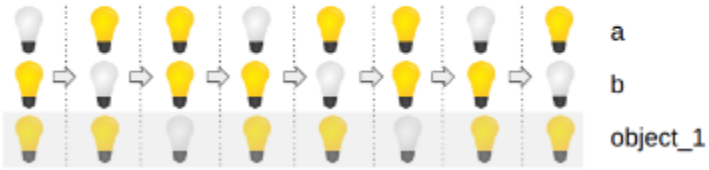
Spatially

All objects are connected directly or transitively via the *pred_1* relation

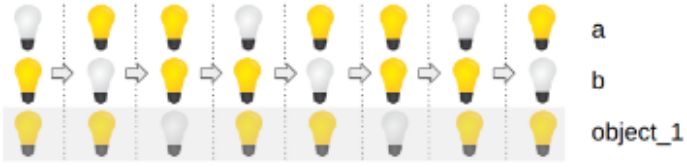
Conceptually

The *pred_1* relation is constrained in that only one LED can be *pred_1* to another (the one_related conceptual constraint). The *on* property is implicitly constrained in our implementation in that an LED's *on* property can only take one value (true or false) at a time.

Statically A light is on whenever another that is pred_1 to it is off



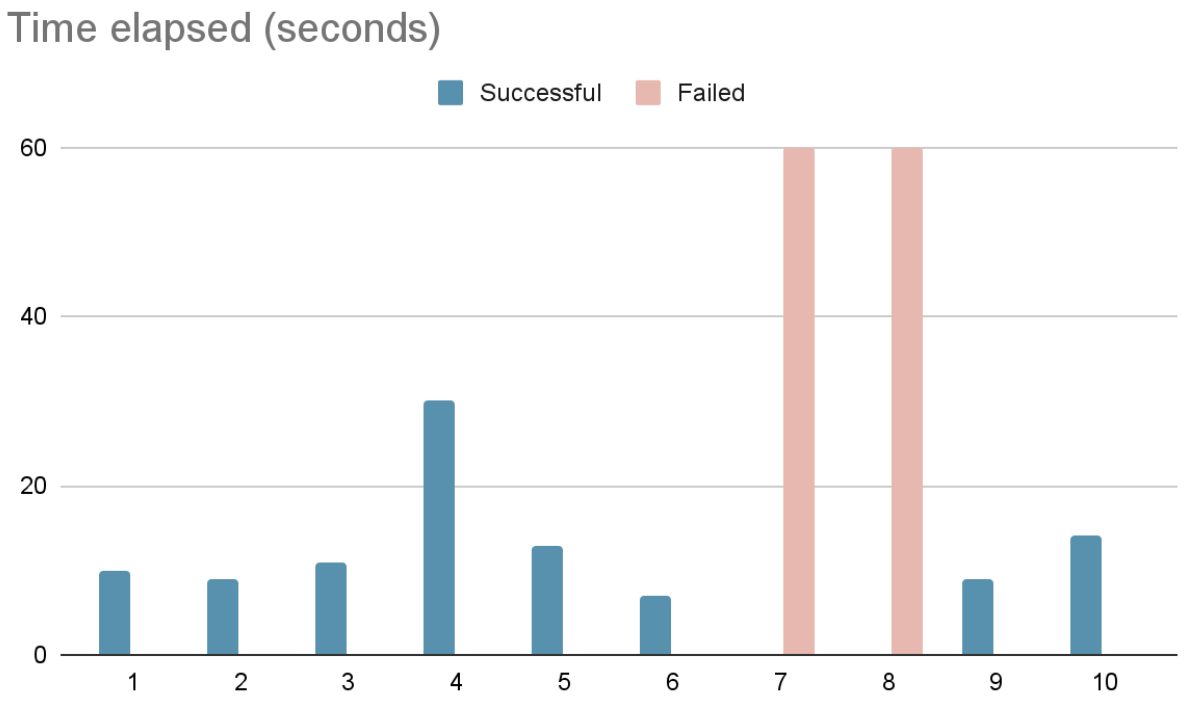
The static rule is respected in every traced state.



Temporally A light turns off if it is pred_1 to another light that was off

The causal rule describes correctly what makes LEDs turn on or off in the trace.

Ten consecutive searches were run. The 100% coverage causal theory shown above was found 8 out of 10 times. Two runs failed to find a perfect theory after 60 seconds; they only found runner-up theories with 75% coverage. These however were found in under 1 second.



ECA task

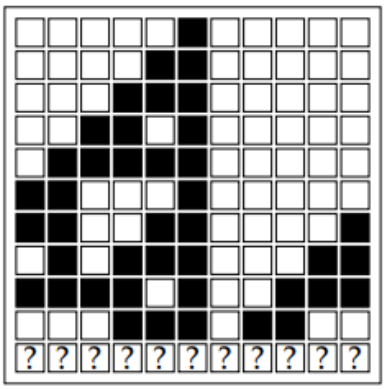
The task is to uncover the rule (or rules) that drive an Elementary Cellular Automaton[17] (ECA).

*"an **elementary cellular automaton** is a one-dimensional cellular automaton where there are two possible states (labeled 0 and 1) and the rule to determine the state of a cell in the next generation depends only on the current state of the cell and its two immediate neighbors"*

While an ECA is depicted as rows where each row succeeds the other in time, they are in fact circles with the first cell considered to be to the right of the last cell, that is, cell 2 is to the right of cell 1, …, cell 11 is to the right of cell 10, and cell 1 is to the right of cell 11.

The Apperception Engine is given these observed rows of an ECA as input:

[17] See https://en.wikipedia.org/wiki/Elementary_cellular_automaton



https://arxiv.org/pdf/1910.02227.pdf (p.34)

Nothing is left to the imagination in this apperception task: All objects are observed (11 cells), their on/off states are all accounted for in the observations, the only relation is *right_of* and all the *right_of* relations between cells are observed.

What is asked of the Apperception Engine this time is to find not a perfect solution (it would take much too long) but a solution that is 90% correct.

The key parameters given to the apperception task are `max_extension{max_object_types:0, max_objects:0, max_predicate_types:0}` (nothing left to the imagination), `good_enough_coverage: 90` (90% accuracy is acceptable), and `time_secs: 30` (time allocated is 30 seconds).

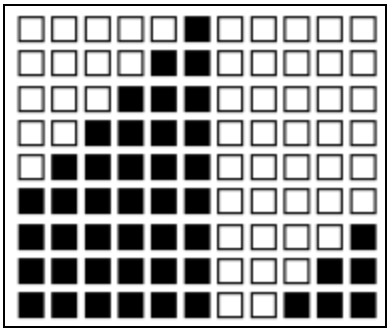
The Apperception Engine finds a causal theory that predicts the behavior of the ECA with 93% accuracy. It contains one causal rule and no static rules.

The causal rule is `on(_A, true)-[right_of(_B, _A), on(_B, true)` which translates to " A cell will turn on if the cell to its right is on"[18].

Ten consecutive searches were run. A causal theory with 93% accuracy was found reliably after 1 to 3 seconds of searching.

The trace produced by this good enough causal theory is

[18] The two causal rules of an 100% accurate theory are "A cell will turn on if it is off and the cell to its right is on" and "A cell will turn off if it is on and the cells to its right is on and left are on"



It approximates the observations given as input while not quite reproducing them. However, the causal theory was found consistently in a handful of seconds.

Future work

Further optimizations to our Apperception Engine implementation will be made as the need for them arises.

We expect this will be the case as we move to implementing cognition actors in Prolog + CHR and have them use the Apperception Engine to make sense of their environment, as they perceive and engage with it.