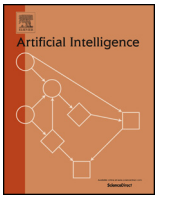
Artificial Intelligence 293 (2021) 103438

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint



Making sense of sensory input

Richard Evans [a,b,*], José Hernández-Orallo [c,d], Johannes Welbl [a], Pushmeet Kohli [a], Marek Sergot [b]

[a] DeepMind, London, United Kingdom
[b] Imperial College London, United Kingdom
[c] Universitat Politècnica de València, Spain
[d] CFI, University of Cambridge, United Kingdom

ARTICLE INFO

Article history:
Received 21 June 2019
Received in revised form 23 September 2020
Accepted 13 December 2020
Available online 5 January 2021

Keywords:
Learning dynamical models
Unsupervised program synthesis

ABSTRACT

This paper attempts to answer a central question in unsupervised learning: what does it mean to "make sense" of a sensory sequence? In our formalization, making sense involves constructing a symbolic causal theory that both explains the sensory sequence and also satisfies a set of unity conditions. The unity conditions insist that the constituents of the causal theory – objects, properties, and laws – must be integrated into a coherent whole. On our account, making sense of sensory input is a type of program synthesis, but it is *unsupervised* program synthesis.

Our second contribution is a computer implementation, the Apperception Engine, that was designed to satisfy the above requirements. Our system is able to produce interpretable human-readable causal theories from very small amounts of data, because of the strong inductive bias provided by the unity conditions. A causal theory produced by our system is able to predict future sensor readings, as well as retrodict earlier readings, and impute (fill in the blanks of) missing sensory readings, in any combination. In fact, it is able to do all three tasks simultaneously.

We tested the engine in a diverse variety of domains, including cellular automata, rhythms and simple nursery tunes, multi-modal binding problems, occlusion tasks, and sequence induction intelligence tests. In each domain, we test our engine's ability to predict future sensor values, retrodict earlier sensor values, and impute missing sensory data. The Apperception Engine performs well in all these domains, significantly out-performing neural net baselines. We note in particular that in the sequence induction intelligence tests, our system achieved human-level performance. This is notable because our system is not a bespoke system designed specifically to solve intelligence tests, but a *general-purpose* system that was designed to make sense of *any* sensory sequence.

1. Introduction

Imagine a machine, equipped with various sensors, that receives a stream of sensory information. It must, somehow, *make sense* of this stream of sensory data. But what does it mean, exactly, to "make sense" of sensory data? We have an intuitive understanding of what is involved in making sense of the sensory stream – but can we specify precisely what is involved? Can this intuitive notion be formalized?

* Corresponding author at: Imperial College London, United Kingdom.
E-mail address: richardevans@google.com (R. Evans).

https://doi.org/10.1016/j.artint.2020.103438

One approach is to treat the sensory sequence as the input to a supervised learning problem[1]: given a sequence $x_{1:t}$ of sensory data from time steps 1 to t, maximize the probability of the next datum x_{t+1}. This family of approaches seeks to maximize $p(x_{t+1} \mid x_{1:t})$. More generally, as well as training the system to predict future sensor readings, we may also train it to *retrodict* past sensor readings (maximizing $p(x_1 \mid x_{2:t})$), and to *impute* missing intermediate values (maximizing $p(x_i \mid x_{1:i-1}, x_{i+1:t})$).

We believe there is more to "making sense" than prediction, retrodiction, and imputation. Predicting the future state of one's photoreceptors may be *part* of what is involved in making sense – but it is not on its own sufficient. The ability to predict, retrodict, and impute is a sign, a surface manifestation, that one has made sense of the input. We want to define the underlying mental model that is constructed when one makes sense of the sensory input, and to show how constructing this mental model *ipso facto* enables one to predict, retrodict, and impute.

In this paper, we assume that making sense of sensory input involves constructing a symbolic theory that explains the sensory input [9–12]. A number of authors, including Lake [13] and Marcus [14], have argued that constructing an explanatory theory is a key component of common sense. Following Spelke and others [15,16], we assume the theory must posit objects that persist over time, with properties that change over time according to general laws. Further, we assume, following John McCarthy and others [17–19], that making sense of the surface sensory perturbations requires positing *latent objects*: some sensory sequences can only be made intelligible by hypothesizing an underlying reality, distinct from the surface features of our sensors, that makes the surface phenomena intelligible. The underlying reality consists of latent objects that causally interact with our sensors to product the sensory perturbations we are given as input. Once we have constructed such a theory, we can apply it to predict future sensor readings, to retrodict past readings, or to impute missing values.

Now constructing a symbolic theory that explains the sensory sequence is necessary for making sense of the sequence. But it is not, we claim, sufficient. There is one further additional ingredient that we add to our characterisation of "making sense". This is the requirement that our theory exhibits a particular form of *unity*: the constituents of our theory – objects, properties, and atoms – must be integrated into a coherent whole. Specifically, our unity condition requires that the objects are interrelated via chains of binary relations, the properties are connected via exclusion relations, and the atoms are unified by jointly satisfying the theory's constraints. This extra unity condition is necessary, we argue, for the theory to achieve good accuracy at prediction, retrodiction, and imputation.[2]

This paper makes two main contributions. The first is a formalization of what it means to "make sense" of the stream of sensory data. According to our definition, making sense of a sensory sequence involves positing a symbolic causal theory – a set of objects, a set of concepts, a set of initial conditions, a set of rules, and a set of constraints – that together satisfy two conditions. First, the theory must explain the sensory readings it is given. Second, the theory must satisfy a particular type of unity. Our definition of unity involves four conditions. (i) *Spatial unity*: all objects must be unified in space via a chain of binary relations. (ii) *Conceptual unity*: all concepts must be unified via constraints. (iii) *Static unity*: all propositions that are true at the same time must jointly satisfy the set of constraints. (iv) *Temporal unity*: all the states must be unified into a sequence by causal rules.

Our second contribution is a description of a particular computer system, the Apperception Engine,[3] that was designed to satisfy the conditions described above.[4] We introduce a causal language, Datalog$^\ni$, that was designed for reasoning about infinite temporal sequences. Given a sensory sequence, our system synthesizes a Datalog$^\ni$ program that, when executed, generates a trace that both explains the sensory sequence and also satisfies the four conditions of unity. This can be seen as a form of *unsupervised program synthesis* [22]. In traditional supervised program synthesis, we are given input/output pairs, and search for a program that, when executed on the inputs, produces the desired outputs. Here, in unsupervised program synthesis, we are given a sensory sequence, and search for a causal theory that, when executed, generates a trajectory that both respects the sensory sequence and also satisfies the conditions of unity.

The Apperception Engine has a number of appealing features. (1) Because the causal theories it generates are symbolic, they are human-readable and hence verifiable. We can understand precisely how the system is making sense of its sensory data.[5] (2) Because of the strong inductive bias (both in terms of the design of the causal language, Datalog$^\ni$, but also in terms of the unity conditions that must be satisfied), the system is data-efficient, able to make sense of the shortest and scantiest of sensory sequences.[6] (3) Our system generates a causal model that is able to accurately predict future sensory input. But *that is not all it can do*; it is also able to retrodict previous values and impute missing sensory values in the middle

[1] See, for example, [1–5]. See also the Predictive Processing paradigm: [6–8].

[2] We show, in the ablation experiments of Section 5, that without these unity conditions, our computer implementation is much less accurate at prediction, retrodiction, and imputation.

[3] "Apperception" comes from the French 'apercevoir'. The term is introduced by Leibniz in the *New Essays Concerning Human Understanding* [20]. Apperception, as we use it in this paper, is the process of assimilating sensory information into a coherent unified whole. See Dewey: "*Apperception* is the relating activity which combines the various sensuous elements presented to the mind at one time into a *whole*, and which *unites* these wholes, recurring at successive times, into a continuous mental life, thereby making psychical life intelligent" [21].

[4] Our code and datasets are publicly available at https://github.com/RichardEvans/apperception.

[5] Human readability is a much touted feature of Inductive Logic Programming (ILP) systems, but when the learned programs become large and include a number of invented auxiliary predicates, the resulting programs become less readable (see [23]). But even a large and complex machine-generated logic program will be easier to understand than a large tensor of floating point numbers.

[6] Our sensory sequences are less than 300 bits. See Table 4.

of the sensory stream. In fact, our system is able to predict, retrodict, and impute simultaneously.[7] (4) The Apperception Engine has been tested in a diverse variety of domains, with encouraging results. The five domains we use are elementary cellular automata, rhythms and nursery tunes, "Seek Whence" and C-test sequence induction intelligence tests [24], multi-modal binding tasks, and occlusion problems. These tasks were chosen because they require cognition rather than mere classificatory perception, and because they are simple for humans but not for modern machine learning systems, e.g. neural networks.[8] The Apperception Engine performs well in all these domains, significantly out-performing neural net baselines. These results are significant because neural systems typically struggle to solve the binding problem (where information from different modalities must somehow be combined into different aspects of one unified object) and fail to solve occlusion tasks (in which objects are sometimes visible and sometimes obscured from view).

We note in particular that in the sequence induction intelligence tests, our system achieved human-level performance. This is notable because the Apperception Engine was not designed to solve these induction tasks; it is not a bespoke hand-engineered solution to this particular domain. Rather, it is a *general-purpose*[9] system that attempts to make sense of *any* sensory sequence. This is, we believe, a highly suggestive result [25].

In ablation tests, we tested what happened when each of the four unity conditions was turned off. Since the system's performance deteriorates noticeably when each unity condition is ablated, this indicates that the unity conditions are indeed doing vital work in our engine's attempts to make sense of the incoming barrage of sensory data.

1.1. Related work

A human being who has built a mental model of the world can use that model for counterfactual reasoning, anticipation, and planning [26–28]. Similarly, computer agents endowed with mental models are able to achieve impressive performance in a variety of domains. For instance, Lukasz Kaiser et al. [29] show that a model-based reinforcement learning agent trained on 100K interactions compares with a state-of-the-art model-free agent trained on tens or hundreds of millions of interactions. David Silver et al. [30] have shown that a model-based Monte Carlo tree search planner with policy distillation can achieve superhuman level performance in a number of board games. The tree search relies, crucially, on an accurate model of the game dynamics.

When we have an accurate model of the environment, we can leverage that model to anticipate and plan. But in many domains, we do not have an accurate model. If we want to apply model-based methods in these domains, we must *learn* a model from the stream of observations. In the rest of this section, we shall describe various different approaches to representing and learning models, and show where our particular approach fits into the landscape of model learning systems.

Before we start to build a model to explain a sensory sequence, one fundamental question is: what form should the model take? We shall distinguish three dimensions of variation of models (adapted from [31]): first, whether they simply model the observed phenomena, or whether they also model latent structure; second, whether the model is explicit and symbolic or implicit; and third, what type of prior knowledge is built into the model structure.

We shall use the hidden Markov model (HMM)[10] [32,33] as a general framework for describing sequential processes. Here, the observation at time t is x_t, and the latent state is z_t. In a HMM, the observation x_t at time t depends only on the latent (unobserved) state z_t. The state z_t in turn depends only on the previous latent state z_{t-1}.

The first dimension of variation amongst models is whether they actually use latent state information z_t to explain the observation x_t. Some approaches [34–39] assume we are *given* the underlying state information $z_{1:t}$. In these approaches, there is no distinction between the observed phenomena and the latent state: $x_i = z_i$. With this simplifying assumption, the only thing a model needs to learn is the transition function. Other approaches [40,2,41] focus only on the observed phenomena $x_{1:t}$ and ignore latent information $z_{1:t}$ altogether. These approaches predict observation x_{t+1} given observation x_t without positing any hidden latent structure. Some approaches take latent information seriously [42–44,5,45]. These jointly learn a perception function (that produces a latent z_t from an observed x_t), a transition function (producing a next latent state z_{t+1} from latent state z_t) and a rendering function (producing a predicted observation x_{t+1} from the latent state z_{t+1}). Our approach also builds a latent representation of the state. As well as positing latent properties (unobserved properties that explain observed phenomena), we also posit latent *objects* (unobserved objects whose relations to observed objects explain observed phenomena).

The second dimension of variation concerns whether the learned model is explicit, symbolic and human-readable, or implicit and inscrutable. In some approaches [42–44,5], the latent states are represented by vectors and the dynamics of the model by weight tensors. In these cases, it is hard to understand what the system has learned. In other approaches

[7] See Example 3 for a case where the Apperception Engine jointly predicts, retrodicts, and imputes.

[8] Fig. 7 shows how neural baselines struggle to solve these tasks.

[9] Although the algorithm is general-purpose (the exact same code is applied to many different domains), domain-specific knowledge can be injected as needed. For example, in the sequence induction tasks (Section 5.1.4), the successor relation on letters is provided to the system. But this information is arguably part of the problem formulation, rather than part of the solution.

[10] Many systems predict state dynamics for partially observable Markov decision processes (POMDPs), rather than HMMs. In a POMDP, the state transition function depends on the previous state z_t and the action a_t performed by an agent. See Jessica Hamrick's paper for an excellent overview [31] of model-based methods in deep learning that is framed in terms of POMDPs. In this paper, we consider HMMs. Adding actions to our model is not particularly difficult, but is left for further work.

[46–49], the latent state is represented symbolically, but the state transition function is represented by the weight tensor of a neural network and is inscrutable. We may have some understanding of what state the machine thinks it is in, but we do not understand why it thinks there is a transition from this state to that. In some approaches [11,12,50–53], both the latent state and the state transition function are represented symbolically. Here, the latent state is a set of ground atoms[11] and the state transition function is represented by a set of universally quantified rules. Our approach falls into this third category. Here, the model is fully interpretable: we can interpret the state the machine thinks it is in, and we can understand the reason why it believes it will transition to the next state.

A third dimension of variation between models is the amount and type of prior knowledge that they include. Some model learning systems have very little prior knowledge. In some of the neural systems (e.g. [2]), the only prior knowledge is the spatial invariance assumption implicit in the convolutional network's structure. Other models incorporate prior knowledge about the way objects and states should be represented. For example, some models assume objects can be composed in hierarchical structures [47]. Other systems additionally incorporate prior knowledge about the type of rules that are used to define the state transition function. For example, some [51–53] use prior knowledge of the event calculus [54]. Our approach falls into this third category. We impose a language bias in the form of rules used to define the state transition function and also impose additional requirements on candidate sets of rules: they must satisfy the four unity conditions introduced above (and elaborated in Section 3.3 below).

To summarize, in order to position our approach within the landscape of other approaches, we have distinguished three dimensions of variation. Our approach differs from neural approaches in that the posited theory is explicit and human readable. Not only is the representation of state explicit (represented as a set of ground atoms) but the transition dynamics of the system are also explicit (represented as universally quantified rules in a domain specific language designed for describing causal structures). Our approach differs from other inductive program synthesis methods in that it posits significant latent structure in addition to the induced rules to explain the observed phenomena: in our approach, explaining a sensory sequence does not just mean constructing a set of rules that explain the transitions; it also involves positing a type signature containing a set of latent properties and a set of latent *objects*. Our approach also differs from other inductive program synthesis methods in the type of prior knowledge that is used: as well as providing a strong language bias by using a particular representation language (a typed extension of datalog with causal rules and constraints), we also inject a substantial inductive bias: the unity conditions, the key constraints on our system, represent domain-*independent* prior knowledge. Our approach also differs from other inductive program synthesis methods in being entirely unsupervised. In contrast, OSLA and OLED [51,52] are supervised, and SPLICE [53] is semi-supervised. See Section 7 for detailed discussion.

1.2. Paper outline

Section 2 introduces basic notation. Section 3 presents the main definition of what it means for a theory to count as a unified interpretation of a sensory sequence. Section 4 describes a computer system that is able to generate unified interpretations of sensory sequences. Section 5 describes our experiments in five different types of task: elementary cellular automata, rhythms and nursery tunes, "Seek Whence" sequence induction tasks, multi-modal binding tasks, and occlusion problems. In Section 6, we show how our system is extended to robustly handle noise. Related work is discussed in Section 7.

2. Background

In this paper, we use basic concepts and standard notation from logic programming [55–57]. A function-free atom is an expression of the form $p(t_1, ..., t_n)$, where p is a predicate of arity $n \geq 0$ and each t_i is either a variable or a constant. We shall use $a, b, c, ...$ for constants, $X, Y, Z, ...$ for variables, and $p, q, r, ...$ for predicate symbols.

A **substitution** σ is a mapping from variables to terms. For example $\sigma = \{X/a, Y/b\}$ replaces variable X with constant a and replaces variable Y with constant b. We write $\alpha\sigma$ for the application of substitution σ to atom α, so e.g. $p(X, Y)\sigma = p(a, b)$.

A **Datalog clause** is a definite clause of the form $\alpha_1 \wedge ... \wedge \alpha_n \rightarrow \alpha_0$ where each α_i is an atom and $n \geq 0$. It is traditional to write clauses from right to left: $\alpha_0 \leftarrow \alpha_1, ..., \alpha_n$. In this paper, we will define a Datalog interpreter implemented in another logic programming language, ASP (answer-set programming). In order to keep the two languages distinct, we write Datalog rules from left to right and ASP clauses from right to left. A **Datalog program** is a set of Datalog clauses.

A key result of logic programming is that every Datalog program has a unique subset-minimal least Herbrand model that can be directly computed by repeatedly generating the consequences of the ground instances of the clauses [58].

We turn now from Datalog to normal logic programs under the answer set semantics [59]. A **literal** is an atom α or a negated atom $not\ \alpha$. A **normal logic program** is a set of clauses of the form:

$$\alpha_0 \leftarrow \alpha_1, ..., \alpha_n$$

[11] A ground atom is a logical atom that contains no variables.

where α_0 is an atom, $\alpha_1, ..., \alpha_n$ is a conjunction of literals, and $n \geq 0$. Normal logic clauses extend Datalog clauses by allowing functions in terms and by allowing negation by failure in the body of the rule.

Answer Set Programming (ASP) is a logic programming language based on normal logic programs under the answer set semantics. Given a normal logic program, an ASP solver finds the set of answer sets for that program. Modern ASP solvers can also be used to solve optimization problems by the introduction of weak constraints [60]. A **weak constraint** is a rule that defines the cost of a certain tuple of atoms. Given a program with weak constraints, an ASP solver can find a preferred answer set with the lowest cost.

3. A computational framework for making sense of sensory sequences

What does it mean to make sense of a sensory sequence? In this section, we formalize what this means, before describing our computer implementation. We assume that the sensor readings have already been discretized into ground atoms of first-order logic, so a sensory reading featuring sensor a can be represented by a ground atom $p(a)$ for some unary predicate p, or by an atom $r(a, b)$ for some binary relation r and unique value b.[12]

Definition 1. An **unambiguous symbolic sensory sequence** is a sequence of sets of ground atoms. Given a sequence $S = (S_1, S_2, ...)$, every **state** S_t in S is a set of ground atoms, representing a *partial* description of the world at a discrete time step t. An atom $p(a) \in S_t$ represents that sensor a has property p at time t. An atom $r(a, b) \in S_t$ represents that sensor a is related via relation r to value b at time t. If $\mathcal{G}$ is the set of all ground atoms, then $S \in \left(2^{\mathcal{G}}\right)^*$.

Example 1. Consider, the following sequence $S_{1:10}$. Here there are two sensors a and b, and each sensor can be either *on* or *off*.

$$S_1 = \{\} \quad S_2 = \{off(a), on(b)\} \quad S_3 = \{on(a), off(b)\} \quad S_4 = \{on(a), on(b)\} \quad S_5 = \{on(b)\}$$
$$S_6 = \{on(a), off(b)\} \quad S_7 = \{on(a), on(b)\} \quad S_8 = \{off(a), on(b)\} \quad S_9 = \{on(a)\} \quad S_{10} = \{\}$$

There is no expectation that a sensory sequence contains readings for all sensors at all time steps. Some of the readings may be missing. In state S_5, we are missing a reading for a, while in state S_9, we are missing a reading for b. In states S_1 and S_{10}, we are missing sensor readings for both a and b. ◁

The central idea is to make sense of a sensory sequence by *constructing a unified theory that explains that sequence*. The key notions, here, are "theory", "explains", and "unified". We consider each in turn.

3.1. The theory

Theories are defined in a new language, Datalog$^{\ni}$, designed for modelling dynamics. In this language, one can describe how facts change over time by writing a causal rule stating that if the antecedent holds at the current time-step, then the consequent holds at the *next* time-step. Additionally, our language includes a frame axiom allowing facts to persist over time: each atom remains true at the next time-step unless it is overridden by a new fact which is incompatible with it. Two facts are incompatible if there is a *constraint* that precludes them from both being true. Thus, Datalog$^{\ni}$ extends Datalog with causal rules and constraints.

Definition 2. A **theory** is a four-tuple (ϕ, I, R, C) of Datalog$^{\ni}$ elements where:

- ϕ is a type signature specifying the types of constants, variables, and arguments of predicates
- I is a set of initial conditions
- R is a set of rules describing the dynamics
- C is a set of constraints

We shall consider each element in turn, starting with the type signature.

Definition 3. Given a set $\mathcal{T}$ of types, a set $\mathcal{O}$ of constants representing individual objects, and a set $\mathcal{P}$ of predicates representing properties and relations, let $\mathcal{G}$ be the set of all ground atoms formed from $\mathcal{T}$, $\mathcal{O}$, and $\mathcal{P}$. Given a set $\mathcal{V}$ of variables, let $\mathcal{U}$ be the set of all unground atoms formed from $\mathcal{T}$, $\mathcal{V}$, and $\mathcal{P}$.

A **type signature** is a tuple (T, O, P, V) where $T \subseteq \mathcal{T}$ is a finite set of types, $O \subseteq \mathcal{O}$ is a finite set of constants representing objects, $P \subseteq \mathcal{P}$ is a finite set of predicates representing properties and relations, and $V \subseteq \mathcal{V}$ is a finite set of variables. We write $\kappa_O : O \to T$ for the type of an object, $\kappa_P : P \to T^*$ for the types of the predicate's arguments, and $\kappa_V : V \to T$ for the type of a variable.

[12] We restrict our attention to unary and binary predicates. This restriction can be made without loss of generality, since every k-ary relationship can be expressed as $k+1$ binary relationships [61].

Now some type signatures are suitable for some sensory sequences, while others are unsuitable, because they do not contain the right constants and predicates. The following definition formalizes this:

Definition 4. Let $G_S = \bigcup_{t \geq 1} S_t$ be the set of all ground atoms that appear in sensory sequence $S = (S_1, ...)$. Let G_ϕ be the set of all ground atoms that are well-typed according to type signature ϕ. If $\phi = (T, O, P, V)$ then $G_\phi = \{p(a_1, ..., a_n) \mid p \in P, \kappa_P(p) = (t_1, ..., t_n), a_i \in O, \kappa_O(a_i) = t_i \text{ for all } i = 1..n\}$. A type signature ϕ is **suitable** for a sensory sequence S if all the atoms in S are well-typed according to signature ϕ, i.e. $G_S \subseteq G_\phi$.

Next, we define the set of unground atoms for a particular type signature.

Definition 5. Let U_ϕ be the set of all **unground** atoms that are well-typed according to signature ϕ. If $\phi = (T, O, P, V)$ then $U_\phi = \{p(v_1, ..., v_n) \mid p \in P, \kappa_P(p) = (t_1, ..., t_n), v_i \in V, \kappa_V(v_i) = t_i \text{ for all } i = 1..n\}$. Note that, according to this definition, an atom is unground if *all* its terms are variables. Note that "unground" means more than simply not ground. For example, $p(a, X)$ is neither ground nor unground.

Example 2. One suitable type signature for the sequence of Example 1 is (T, O, P, V), consisting of types $T = \{s\}$, objects $O = \{a{:}s, b{:}s\}$, predicates $P = \{on(s), off(s)\}$, and variables $V = \{X{:}s, Y{:}s\}$. Here, and throughout, we write $a{:}s$ to mean that object a is of type s, $on(s)$ to mean that unary predicate on takes one argument of type s, and $X{:}s$ to mean that variable X is of type s. The unground atoms are $U_\phi = \{on(X), off(X), on(Y), off(Y)\}$. There are, of course, an infinite number of other suitable signatures. ◁

Definition 6. The **initial conditions** I of a theory (ϕ, I, R, C) is a set of ground atoms from G_ϕ representing a partial description of the facts true at the initial time step.

The initial conditions are needed to specify the initial values of the latent unobserved information. Some systems (e.g. LFIT [12]) define a predictive model without using a set I of initial conditions. These systems are able to avoid positing initial conditions because they do not use latent unobserved information. But any system that does invoke latent information beneath the surface of the sensory stimulations must also define the initial values of the latent information.

The rules define the dynamics of the theory:

Definition 7. There are two types of rule in Datalog$^\ni$. A **static rule** is a definite clause of the form $\alpha_1 \wedge ... \wedge \alpha_n \rightarrow \alpha_0$, where $n \geq 0$ and each α_i is an *unground* atom from U_ϕ consisting of a predicate and a list of variables. Informally, a static rule is interpreted as: if conditions $\alpha_1, ...\alpha_n$ hold at the current time step, then α_0 also holds at that time step. A **causal rule** is a clause of the form $\alpha_1 \wedge ... \wedge \alpha_n \ni \alpha_0$, where $n \geq 0$ and each α_i is an unground atom from U_ϕ. A causal rule expresses how facts change over time. Rule $\alpha_1 \wedge ... \wedge \alpha_n \ni \alpha_0$ states that if conditions $\alpha_1, ...\alpha_n$ hold at the current time step, then α_0 holds at the *next* time step.

All variables in rules are implicitly universally quantified. So, for example, $on(X) \ni off(X)$ states that for all objects X, if X is currently *on*, then X will become *off* at the next-time step.

The constraints rule out certain combinations of atoms[13]:

Definition 8. There are three types of constraint in Datalog$^\ni$. A **unary constraint** is an expression of the form $\forall X, p_1(X) \oplus ... \oplus p_n(X)$, where $n > 1$, meaning that for all X, exactly one of $p_1(X), ..., p_n(X)$ holds. A **binary constraint** is an expression of the form $\forall X, \forall Y, r_1(X, Y) \oplus ... \oplus r_n(X, Y)$ where $n > 1$, meaning that for all objects X and Y, exactly one of the binary relations hold. A **uniqueness constraint** is an expression of the form $\forall X, \exists! Y{:}t_2, r(X, Y)$, which means that for all objects X of type t_1 there exists a *unique* object Y such that $r(X, Y)$.

Note that the rules and constraints are constructed entirely from *unground* atoms. Disallowing constants prevents special-case rules that apply to particular objects, and forces the theory to be general.[14]

3.2. Explaining the sensory sequence

A theory explains a sensory sequence if the theory generates a trace[15] that covers that sequence. In this section, we explain the trace and the covering relation.

[13] Note that exclusive disjunction between atoms $p_1(X), ..., p_n(X)$ is different from *xor* between the n atoms. The *xor* of n atoms is true if an *odd* number of the atoms hold, while the exclusive disjunction is true if *exactly one* of the atoms holds. We write $p_1(X) \oplus ... \oplus p_n(X)$ to mean exclusive disjunction between n atoms, not the application of $n - 1$ *xor* operations.

[14] This restriction also occurs in some ILP systems [12,62].

[15] In [12], the trace is called the *orbit*.

Definition 9. Every theory $\theta = (\phi, I, R, C)$ generates an infinite sequence $\tau(\theta)$ of sets of ground atoms, called the **trace** of that theory. Here, $\tau(\theta) = (A_1, A_2, ...)$, where each A_t is the smallest set of atoms satisfying the following conditions:

- $I \subseteq A_1$
- If there is a static rule $\beta_1 \wedge ... \wedge \beta_m \rightarrow \alpha$ in R and a ground substitution σ such that A_t satisfies $\beta_i\sigma$ for each antecedent β_i, then $\alpha\sigma \in A_t$
- If there is a causal rule $\beta_1 \wedge ... \wedge \beta_m \ni \alpha$ in R and a ground substitution σ such that A_{t-1} satisfies $\beta_i\sigma$ for each antecedent β_i, then $\alpha\sigma \in A_t$
- *Frame axiom*: if α is in A_{t-1} and there is no atom in A_t that is incompossible with α w.r.t constraints C, then $\alpha \in A_t$. Two ground atoms are **incompossible** if there is some constraint c in C and some substitution σ such that the ground constraint $c\sigma$ precludes both atoms being true.

The frame axiom is a simple way of providing inertia: a proposition continues to remain true until something new comes along which is incompatible with it. Including the frame axiom makes our theories much more concise: instead of needing rules to specify all the atoms which remain the same, we only need rules that specify the atoms that change.

Note that the state transition function is deterministic: A_t is uniquely determined by A_{t-1}.

Theorem 1. *The trace of every theory repeats after some finite number of steps. For any theory θ, there exists a k such that $\tau(\theta) = (A_1, ..., A_{k-1}, A_k, A_{k+1}, ...)$ and for all $i \geq 0$, $A_i = A_{k+i}$.*

Proof. Since the set G_ϕ of ground atoms is finite, there must be a k such that $A_1 = A_k$. The proof proceeds by induction on i. If $i = 0$, the proof is trivial. When $i > 0$, note that the trace function τ satisfies the Markov condition that the next state A_{t+1} depends only on the current state A_t, and not on any earlier states. Hence if $A_i = A_{i+k}$, then $A_{i+1} = A_{i+k+1}$. □

One important consequence of Theorem 1 is:

Theorem 2. *Given a theory θ and a ground atom α, it is decidable whether α appears somewhere in the infinite trace $\tau(\theta)$.*

Proof. Let $\tau(\theta)$ be the infinite sequence $(A_1, A_2, ...)$. From Theorem 1, the trace must repeat after k time steps. Thus, to check whether ground atom α appears somewhere in $\tau(\theta)$, it suffices to test if α appears in $A_1, ..., A_k$. □

Next we define what it means for a theory to "explain" a sensory sequence.

Definition 10. Given finite sequence $S = (S_1, ..., S_T)$ and (not necessarily finite) S', $S \sqsubseteq S'$ if $S' = (S'_1, S'_2, ...)$ and $S_i \subseteq S'_i$ for all $1 \leq i \leq T$. If $S \sqsubseteq S'$, we say that S is **covered by** S', or that S' covers S. A theory θ **explains** a sensory sequence S if the trace of θ covers S, i.e. $S \sqsubseteq \tau(\theta)$.

In providing a theory θ that explains a sensory sequence S, we make S intelligible by placing it within a bigger picture: while S is a scanty and incomplete description of a fragment of the time-series, $\tau(\theta)$ is a complete and determinate description of the whole time-series.

Example 3. We shall provide a theory to explain the sensory sequence S of Example 1.

Consider the type signature $\phi = (T, O, P, V)$, consisting of types $T = \{s\}$, objects $O = \{a{:}s, b{:}s\}$, predicates $P = \{on(s), off(s), p_1(s), p_2(s), p_3(s), r(s, s)\}$, and variables $V = \{X{:}s, Y{:}s\}$. Here, ϕ extends the type signature of Example 2 by adding three unary predicates p_1, p_2, p_3, and one binary relation r.[16]

Consider the theory $\theta = (\phi, I, R, C)$, where:

$$I = \left\{ \begin{array}{l} p_1(b) \\ p_2(a) \\ r(a,b) \\ r(b,a) \end{array} \right\} \quad R = \left\{ \begin{array}{l} p_1(X) \ni p_2(X) \\ p_2(X) \ni p_3(X) \\ p_3(X) \ni p_1(X) \\ p_1(X) \rightarrow on(X) \\ p_2(X) \rightarrow on(X) \\ p_3(X) \rightarrow off(X) \end{array} \right\} \quad C = \left\{ \begin{array}{l} \forall X{:}s,\ on(X) \oplus off(X) \\ \forall X{:}s,\ p_1(X) \oplus p_2(X) \oplus p_3(X) \\ \forall X{:}s,\ \exists! Y{:}s\, r(X, Y) \end{array} \right\}$$

[16] Extended type signatures are generated by the machine, not by hand. Our computer implementation searches through the space of increasingly complex type signatures extending the original signature. This search process is described in Section 4.1.

The infinite trace $\tau(\theta) = (A_1, A_2, ...)$ for theory θ begins with:

$$A_1 = \{on(a), on(b), p_2(a), p_1(b), r(a,b), r(b,a)\} \quad A_2 = \{off(a), on(b), p_3(a), p_2(b), r(a,b), r(b,a)\}$$
$$A_3 = \{on(a), off(b), p_1(a), p_3(b), r(a,b), r(b,a)\} \quad A_4 = \{on(a), on(b), p_2(a), p_1(b), r(a,b), r(b,a)\}$$
$$\ldots$$

Note that the trace repeats at step 4. In fact, it is always true that the trace repeats after some finite set of time steps.

Theory θ explains the sensory sequence S of Example 1, since the trace $\tau(\theta)$ covers S. Note that $\tau(\theta)$ "fills in the blanks" in the original sequence S, both predicting final time step 10, retrodicting initial time step 1, and imputing missing values for time steps 5 and 9. ◁

3.3. Unifying the sensory sequence

Next, we proceed from explaining a sensory sequence to "making sense" of that sequence. In order for θ to make sense of S, it is *necessary* that $\tau(\theta)$ covers S. But this condition is not, on its own, *sufficient*. The extra condition that is needed for θ to count as "making sense" of S is for θ to be *unified*. We require that the constituents of the theory are integrated into a coherent whole. A trace $\tau(\theta)$ of theory θ is a (i) sequence of (ii) sets of ground atoms composed of (iii) predicates and (iv) objects. For the theory θ to be unified is for unity to be achieved at each of these four levels[17]:

Definition 11. A theory θ is **unified** if each of the four conditions hold:

1. Objects are united in space (see Section 3.3.1)
2. Predicates are united via constraints (see Section 3.3.2)
3. Ground atoms are united into states by jointly respecting constraints and static rules (see Section 3.3.3)
4. States are united into a sequence by causal rules (see Section 3.3.4)

3.3.1. Spatial unity

Definition 12. A theory θ satisfies **spatial unity** if for each state A_t in $\tau(\theta) = (A_1, A_2, ...)$, for each pair (x, y) of distinct objects, x and y are connected via a chain of binary atoms $\{r_1(x, z_1), r_2(z_1, z_2), ...r_n(z_{n-1}, z_n), r_{n+1}(z_n, y)\} \subseteq A_t$.

If this condition is satisfied, it means that given any object, we can get to any other object by hopping along relations. Everything is connected, even if only indirectly.

Note that this notion of spatial unity is rather abstract: the requirement is only that every pair of objects are indirectly connected via some chain of binary relations. Although some of these binary relations might be spatial relations (e.g. "left-of"), they need not all be. The requirement is only that every pair of objects are connected via some chain of binary relations; it does not insist that each binary relation has a specifically "spatial" interpretation.

3.3.2. Conceptual unity

A theory satisfies conceptual unity if every predicate is involved in some constraint, either exclusive disjunction ($\oplus$) or unique existence ($\exists!$). The intuition here is that constraints combine predicates into clusters of mutual incompatibility.

Definition 13. A theory $\theta = (\phi, I, R, C)$ satisfies **conceptual unity** if for each unary predicate p in ϕ, there is some xor constraint in C of the form $\forall X{:}t, p(X) \oplus q(X) \oplus ...$ containing p; and, for each binary predicate r in ϕ, there is some xor constraint in C of the form $\forall X{:}t_1, \forall Y{:}t_2, r(X,Y) \oplus s(X,Y) \oplus ...$ or some $\exists!$ constraint in C of the form $\forall X{:}t, \exists! Y{:}t_2, r(X,Y)$.

To see the importance of this, observe that if there are no constraints, then there are no exhaustiveness or exclusiveness relations between atoms. An xor constraint e.g. $\forall X{:}t, on(X) \oplus off(X)$ both rules out the possibility that an object is simultaneously *on* and *off* (exclusiveness) and also rules out the possibility that an object of type t is neither *on* nor *off* (exhaustiveness). It is exhaustiveness which generates states that are *determinate*, in which it is guaranteed every object of type t is e.g. either *on* or *off*. It is exclusiveness which generates *incompossibility* between atoms, e.g. that $on(a)$ and $off(a)$ are incompossible. Incompossibility, in turn, is needed to constrain the scope of the frame axiom (see Definition 9 above). Without incompossibility, *all* atoms from the previous time-step would be transferred to the next time-step, and the set of true atoms in the sequence $(S_1, S_2, ...)$ would grow monotonically over time: $S_i \subseteq S_j$ if $i \leq j$, which is clearly unacceptable. The purpose of the constraint of conceptual unity is to collect predicates into groups, to provide determinacy in each state, and to ground the incompossibility relation that constrains the way information persists between states.[18]

[17] The unity conditions were inspired by Kant's discussion of the "synthetic unity of apperception" in the *Critique of Pure Reason* [63].

[18] A natural question to ask at this point is: why use exclusive disjunction to represent constraints? Why not instead represent constraints using strong negation and negation as failure [64], since an exclusive disjunction can always be converted into a set of extended clauses representing the predicates'

3.3.3. Static unity

In our effort to interpret the sensory sequence, we construct various ground atoms. These need to be grouped together, somehow, into states (sets of atoms). But what determines how these atoms are grouped together into states?

Treating a set A of ground atoms as a state is (i) to insist that A satisfies all the constraints in C and (ii) to insist that A is closed under the static rules in R:

Definition 14. A theory $\theta = (\phi, I, R, C)$ satisfies **static unity** if every state $(A_1, A_2, ...)$ in $\tau(\theta)$ satisfies all the constraints in C and is closed under the static rules in R.

Static unity is an uncontroversial requirement and is used in other ILP systems [67,68]. Note that, from the definition of the trace in Definition 9, all the states in $\tau(\theta)$ are automatically closed under the static rules in R.

3.3.4. Temporal unity

Given a set of states, we need to unite these elements in a *sequence*. According to the fourth and final condition of unity, the only thing that can unite states in a sequence is a set of *causal rules*. These causal rules are *universal* in two senses: they apply to all object tuples, and they apply at all times. A causal rule $\alpha_1 \wedge ... \wedge \alpha_n \exists \alpha_0$ fixes the temporal relation between the atoms $\alpha_1, ..., \alpha_n$ (which are true at t) and the atom α_0 (which is true at $t+1$):

Definition 15. A sequence $(A_1, A_2, ...)$ of states satisfies **temporal unity** with respect to a set $R_{\exists}$ of causal rules if, for each $\alpha_1 \wedge ... \wedge \alpha_n \exists \alpha_0$ in $R_{\exists}$, for each ground substitution σ, for each time-step t, if $\{\alpha_1\sigma, ..., \alpha_n\sigma\} \subseteq A_t$ then $\alpha_0\sigma \in A_{t+1}$.

Temporal unity is an uncontroversial requirement and is also used in other ILP systems such as LFIT [12]. Note that, from the definition of the trace in Definition 9, the trace $\tau(\theta)$ *automatically* satisfies temporal unity.

3.3.5. The four conditions of unity

To recap, the trace of a theory is a sequence of sets of atoms. The four types of element are objects, predicates, sets of atoms, and sequences of sets of atoms. Each of the four types of element has its own form of unity:

1. *Spatial unity*: objects are united in space by being connected via chains of relations
2. *Conceptual unity*: predicates are united by constraints
3. *Static unity*: atoms are united in a state by jointly satisfying constraints and static rules
4. *Temporal unity*: states are united in a sequence by causal rules

Since temporal unity is automatically satisfied from the definition of a trace in Definition 9, we are left with only three unity conditions that need to be explicitly checked: spatial unity, conceptual unity, and static unity. A trace partially satisfies static unity since the static rules are automatically enforced by Definition 9; but the constraints are not necessarily satisfied.

Note that both checking spatial unity and checking static unity require checking every time-step, and the trace is infinitely long. However, as long as the trace repeats at some point, Theorem 1 ensures that we need only check the finite portion of the trace until we find the first repetition (the first k such that $A_1 = A_k$ where $\tau(\theta) = (A_1, ...)$).

3.4. Making sense

Now we are ready to define the central notion of "making sense" of a sequence.

Definition 16. A theory θ **makes sense** of a sensory sequence S if θ explains S, i.e. $S \sqsubseteq \tau(\theta)$, and θ satisfies the four conditions of unity of Definition 11. If θ makes sense of S, we also say that θ is a **unified interpretation** of S.

Example 4. The theory θ of Example 3 satisfies the four unity conditions since:

1. For each state A_i in $\tau(\theta)$, a is connected to b via the singleton chain $\{r(a, b)\}$, and b is connected to a via $\{r(b, a)\}$.

exclusiveness, and one normal clause representing their exhaustiveness? The main reason we chose to add exclusive disjunction as a first-class language construct in Datalog$^{\exists}$, rather than adding negation as failure, is that it means we can avoid the complexities involved in the semantics if we added negation as failure to our target language. There are various semantics for normal logic programs that include negation as failure (e.g. Clark completion [65], stable model semantics [59], well-founded models [66]), but each of them introduces significant additional complexities when compared with the least model of a definite logic program: the Clark completion is not always consistent (does not always have a model), the stable model semantics assigns the meaning of a normal logic program to a *set* of models rather than a single model, and the well-founded model uses a 3-valued logic where atoms can be true, false, or undefined. Thus, the main reason for expressing constraints using exclusive disjunction (rather than using negation as failure) is to restrict the rules to definite rules and avoid the complexities of the various semantics of normal logic programs.

2. The predicates of θ are $on, off, p_1, p_2, p_3, r$. Here, on and off are involved in the constraint $\forall X{:}s,\ on(X) \oplus off(X)$, while p_1, p_2, p_3 are involved in the constraint $\forall X{:}s,\ p_1(X) \oplus p_2(X) \oplus p_3(X)$, and r is involved in the constraint $\forall X{:}s,\ \exists! Y{:}s\ r(X, Y)$.
3. Let $\tau(\theta) = (A_1, A_2, A_3, A_4, ...)$. It is straightforward to check that A_1, A_2, and A_3 satisfy each constraint in C. Observe that A_4 repeats A_1, thus Theorem 1 ensures that we do not need to check any more time steps.
4. Temporal unity is automatically satisfied by the definition of the trace $\tau(\theta)$ in Definition 9.

Hence, θ makes sense of sensory sequence S of Example 1, since $S \sqsubseteq \tau(\theta)$ (Example 3) and θ also satisfies the four conditions of unity. ◁

In our search for interpretations that make sense of sensory sequences, we are particularly interested in *parsimonious* interpretations. To this end, we define the cost of a theory[19]:

Definition 17. Given a theory $\theta = (\phi, I, R, C)$, the **cost** of θ is

$$|I| + \sum \left\{ n + 1 \mid \alpha_1 \wedge ... \wedge \alpha_n \circ \alpha_0 \in R, \circ \in \{\rightarrow, \Rsh\} \right\}$$

Here, $cost(\theta)$ is just the total number of ground atoms in I plus the total number of unground atoms in the rules of R.

The key notion of this section is the discrete apperception task.

Definition 18. The input to an apperception task is a triple (S, ϕ, C) consisting of a sensory sequence S, a suitable type signature ϕ, and a set C of (well-typed) constraints such that (i) each predicate in S appears in some constraint in C and (ii) S can be extended to satisfy C: there exists a sequence S' covering S such that each state in S' satisfies each constraint in C.

Given such an input triple (S, ϕ, C), the **discrete apperception task** is to find the lowest cost theory $\theta = (\phi', I, R, C')$ such that ϕ' extends ϕ, $C' \supseteq C$, and θ makes sense of S.

Note that the input to an apperception task is more than just a sensory sequence S. It also contains a type signature ϕ and a set C of constraints. A natural question at this point is: why not simply let the input to an apperception task be just the sequence S, and ask the system to produce some theory θ satisfying the unity conditions such that $S \sqsubseteq \tau(\theta)$? The reason that the input needs to contain types ϕ and constraints C to supplement S is that otherwise the task is severely under-constrained, as the following example shows.

Example 5. Suppose our sequence is $S = (\{on(a)\}, \{off(a)\}, \{on(a)\}, \{off(a)\}, \{on(a)\}, \{off(a)\})$. If we are not given any constraints (such as $\forall X : t, on(X) \oplus off(X)$), if we are free to construct any ϕ and any set C of constraints, then the following interpretation $\theta = (\phi, I, R, C)$ will suffice, where $\phi = (T, O, P, V)$, consisting of types $T = \{t\}$, objects $O = \{a{:}t\}$, predicates $P = \{on(t), off(t), p(t), q(t)\}$, and variables $V = \{X{:}t\}$, and suppose that I, R, C are defined as:

$$I = \left\{ \begin{array}{l} on(a) \\ off(a) \end{array} \right\} \quad R = \{\ \} \quad C = \left\{ \begin{array}{l} \forall X{:}t,\ on(X) \oplus p(X) \\ \forall X{:}t,\ off(X) \oplus q(X) \end{array} \right\}$$

Here we have introduced two latent predicates p and q which are incompatible with on and off respectively. But in this interpretation, on and off are not incompatible with each other, so the degenerate interpretation (where both on and off are true at all times) is acceptable. This shows the need for including constraints on the input predicates as part of the *task formulation*.

More generally, for *any* sensory sequence $(S_1, ..., S_T)$ featuring predicates $p_1, ..., p_n$, but no constraints between $p_1, ..., p_n$, we can always construct a degenerate interpretation by adding new predicates $q_1, ..., q_n$ with an xor constraint $\forall X : t, p_i(X) \oplus q_i(X)$ between each predicate p_i and the corresponding new predicate q_i. In the degenerate interpretation, the initial conditions I are $S_1 \cup ... \cup S_T$, and the rules R are empty. This shows that, without constraints on the predicates appearing in the initial sequence, the problem is underspecified. ◁

The apperception task can be generalized to the case where we are given as input, not a single sensory sequence S, but a set of m such sequences.

[19] Note that this simple measure of cost does not depend on the constraints in C or the type signature ϕ. There are various alternative more complex definitions of *cost*. We could, for example, use the Kolmogorov complexity [69] of θ: the size of the smallest program that can generate θ. Or we could use Levin complexity [70] and also take into account the log of the computation time needed to generate $\tau(\theta)$, up to the point where the trace first repeats.

Definition 19. Given a set $\{S^1, ..., S^m\}$ of sensory sequences, a type signature ϕ and constraints C such that each (S^i, ϕ, C) is a valid input to an apperception task as defined in Definition 18, the **generalized apperception task** is to find a lowest-cost theory $(\phi', \{\}, R, C')$ and sets $\{I^1, ..., I^m\}$ of initial conditions such that ϕ' extends ϕ, $C' \supseteq C$, and for each $i = 1..m$, (ϕ', I^i, R, C') makes sense of S^i.

3.5. Examples

In this section, we provide a worked example of an apperception task, along with different unified interpretations. We wish to highlight that there are always many alternative ways of interpreting a sensory sequence, each with different latent information (although some may have higher cost than others).

We continue to use our running example, the sensory sequence from Example 1. Here there are two sensors a and b, and each sensor can be *on* or *off*.

$$S_1 = \{\} \quad S_2 = \{off(a), on(b)\} \quad S_3 = \{on(a), off(b)\} \quad S_4 = \{on(a), on(b)\} \quad S_5 = \{on(b)\}$$
$$S_6 = \{on(a), off(b)\} \quad S_7 = \{on(a), on(b)\} \quad S_8 = \{off(a), on(b)\} \quad S_9 = \{on(a)\} \quad S_{10} = \{\}$$

Let $\phi = (T, O, P, V)$ where $T = \{sensor\}$, $O = \{a, b\}$, $P = \{on(sensor), off(sensor)\}$, $V = \{X{:}sensor\}$. Let $C = \{\forall X{:}sensor, on(X) \oplus off(X)\}$.

Examples 6, 7, and 8 below show three different unified interpretations of Example 1.

Example 6. One possible way of interpreting Example 1 is as follows. The sensors a and b are simple state machines that cycle between states p_1, p_2, and p_3. Each sensor switches between *on* and *off* depending on which state it is in. When it is in states p_1 or p_2, the sensor is on; when it is in state p_3, the sensor is off. In this interpretation, the two state machines a and b do not interact with each other in any way. Both sensors are following the same state transitions. The reason the sensors are out of sync is because they start in different states.

The type signature for this first unified interpretation is $\phi' = (T, O, P, V)$, where $T = \{sensor\}$, $O = \{a{:}sensor, b{:}sensor\}$, $P = \{on(sensor), off(sensor), r(sensor, sensor), p_1(sensor), p_2(sensor), p_3(sensor)\}$, and $V = \{X{:}sensor, Y{:}sensor\}$. The three unary predicates p_1, p_2, and p_3 are used to represent the three states of the state machine.

Our first unified interpretation is the tuple (ϕ', I, R, C'), where:

$$I = \left\{ \begin{array}{l} p_2(a) \\ p_1(b) \\ r(a,b) \\ r(b,a) \end{array} \right\} \quad R = \left\{ \begin{array}{l} p_1(X) \ni p_2(X) \\ p_2(X) \ni p_3(X) \\ p_3(X) \ni p_1(X) \\ p_1(X) \rightarrow on(X) \\ p_2(X) \rightarrow on(X) \\ p_3(X) \rightarrow off(X) \end{array} \right\} \quad C' = \left\{ \begin{array}{l} \forall X{:}sensor,\ on(X) \oplus off(X) \\ \forall X{:}sensor,\ p_1(X) \oplus p_2(X) \oplus p_3(X) \\ \forall X{:}sensor,\ \exists! Y{:}sensor\, r(X,Y) \end{array} \right\}$$

The update rules R contain three causal rules (using $\ni$) describing how each sensor cycles from state p_1 to p_2 to p_3, and then back again to p_1. For example, the causal rule $p_1(X) \ni p_2(X)$ states that if sensor X satisfies p_1 at time t, then X satisfies p_2 at time $t+1$. We know that X is a sensor from the variable typing information in ϕ'. R also contains three static rules (using $\rightarrow$) describing how the *on* or *off* attribute of a sensor depends on its state. For example, the static rule $p_1(Y) \rightarrow on(X)$ states that if X satisfies p_1 at time t, then X also satisfies *on* at time t.

The constraints C' state that (i) every sensor is (exclusively) either *on* or *off*, that every sensor is (exclusively) either p_1, p_2, or p_3, and that every sensor has exactly one sensor that is related by r to it. The binary r predicate, or something like it, is needed to satisfy the constraint of spatial unity.

In this first interpretation, three new predicates are invented (p_1, p_2, and p_3) to represent the three states of the state machine. In the next interpretation, we will introduce new invented objects instead of invented predicates.

Given the initial conditions I and the update rules R, we can use our interpretation to compute which atoms hold at which time step. In this case, $\tau(\theta) = (A_1, A_2, ...)$ where $S_i \sqsubseteq A_i$. Note that this trace repeats: $A_i = A_{i+3}$. We can use the trace to predict the future values of our two sensors at time step 10, since $A_{10} = \{on(a), on(b), r(a,b), r(b,a), p_2(a), p_1(b)\}$.

As well as being able to predict future values, we can retrodict past values (filling in A_1), or interpolate intermediate unknown values (filling in A_5 or A_9).[20] But although an interpretation provides the resources to "fill in" missing data, it has no particular bias to predicting future time-steps. The conditions which it is trying to satisfy (the unity conditions of Section 3.3) do not explicitly insist that an interpretation must be able to predict future time-steps. Rather, the ability to predict the future (as well as the ability to retrodict the past, or interpolate intermediate values) is a *derived* capacity that emerges from the more fundamental capacity to "make sense" of the sensory sequence. ◁

[20] This ability to "impute" intermediate unknown values is straightforward given an interpretation. Recent results show that current neural methods for sequence learning are more comfortable predicting future values than imputing intermediate values.

Example 7. There are always infinitely many different ways of interpreting a sensory sequence. Next, we show a rather different interpretation of $S_{1:10}$ from that of Example 6. In our second unified interpretation, we no longer see sensors a and b as self-contained state-machines. Now, we see the states of the sensors as depending on their left and right neighbours. In this new interpretation, we no longer need the three invented unary predicates (p_1, p_2, and p_3), but instead introduce a new *object*.

Object invention is much less explored than predicate invention in inductive logic programming. But Dietterich et al. [71] anticipated the need for it, and Inoue [18] uses meta-level abduction to posit unperceived objects.

Our new type signature $\phi' = (T, O, P, V)$, where $T = \{sensor\}$, $O = \{a{:}sensor, b{:}sensor, c{:}sensor\}$, $P = \{on(sensor), off(sensor), r(sensor, sensor)\}$, and $V = \{X{:}sensor, Y{:}sensor\}$.

In this new interpretation, imagine there is a one-dimensional cellular automaton with three cells, a, b, and (unobserved) c. The three cells wrap around: the right neighbour of a is b, the right neighbour of b is c, and the right neighbour of c is a. In this interpretation, the spatial relations are fixed. (We shall see another interpretation later where this is not the case.) The cells alternate between on and off according to the following simple rule: if X's left neighbour is on (respectively off) at t, then X is on (respectively off) at $t+1$.

Note that objects a and b are the two sensors we are given, but c is a new unobserved latent object that we posit in order to make sense of the data. Many interpretations follow this pattern: new latent unobserved objects are posited to make sense of the changes to the sensors we are given.

Note further that part of finding an interpretation is constructing the spatial relation between objects; this is not something we are given, but something we must *construct*. In this case, we posit that the imagined cell c is inserted to the right of b and to the left of a.

We represent this interpretation by the tuple (ϕ', I, R, C'), where:

$$I = \left\{ \begin{array}{l} on(a) \\ on(b) \\ off(c) \\ r(a,b) \\ r(b,c) \\ r(c,a) \end{array} \right\} \quad R = \left\{ \begin{array}{l} r(X,Y) \wedge off(X) \supset off(Y) \\ r(X,Y) \wedge on(X) \supset on(Y) \end{array} \right\} \quad C' = \left\{ \begin{array}{l} \forall X{:}sensor,\ on(X) \oplus off(X) \\ \forall X{:}sensor,\ \exists!Y{:}sensor,\ r(X,Y) \end{array} \right\}$$

Here, ϕ' extends ϕ, C' extends C, and the interpretation satisfies the unity conditions. ◁

Example 8. We shall give one more way of interpreting the same sensory sequence, to show the variety of possible interpretations.

In our third interpretation, we will posit three latent cells, c_1, c_2, and c_3 that are distinct from the sensors a and b. Cells have static attributes: each cell can be either black or white, and this is a permanent unchanging feature of the cell. Whether a sensor is on or off depends on whether the cell it is currently contained in is black or white. The reason why the sensors change from on to off is because they *move* from one cell to another.

Our new type signature (T, O, P, V) distinguishes between cells and sensors as separate types: $T = \{cell, sensor\}$, $O = \{a : sensor, b : sensor, c_1 : cell, c_2 : cell, c_3 : cell\}$, $P = \{on(sensor), off(sensor), part(sensor, cell), r(cell, cell), black(cell), white(cell)\}$, and $V = \{X : sensor, Y : cell, Y_2 : cell\}$. Our interpretation is the tuple (ϕ, I, R, C), where:

$$I = \left\{ \begin{array}{l} part(a,c_1) \\ part(b,c_2) \\ r(c_1,c_2) \\ r(c_2,c_3) \\ r(c_3,c_1) \\ black(c_1) \\ black(c_2) \\ white(c_3) \end{array} \right\} \quad R = \left\{ \begin{array}{l} part(X,Y) \wedge black(Y) \rightarrow on(X) \\ part(X,Y) \wedge white(Y) \rightarrow off(X) \\ r(Y,Y_2) \wedge part(X,Y_2) \supset part(X,Y) \end{array} \right\} \quad C = \left\{ \begin{array}{l} \forall X{:}sensor,\ on(X) \oplus off(X) \\ \forall Y{:}cell,\ black(Y) \oplus white(Y) \\ \forall X{:}sensor,\ \exists!Y : cell,\ part(X,Y) \\ \forall Y{:}cell,\ \exists!Y_2 : cell,\ r(Y,Y_2) \end{array} \right\}$$

The update rules R state that the *on* or *off* attribute of a sensor depends on whether its current cell is black or white. They also state that the sensors move from right-to-left through the cells.

In this interpretation, there is no state information in the sensors. All the variability is explained by the sensors moving from one static object to another.

Here, the sensors move about, so spatial unity is satisfied by different sets of atoms at different time-steps. For example, at time-step 1, sensors a and b are indirectly connected via the ground atoms $part(a, c_1), r(c_1, c_2), part(b, c_2)$. But at time-step 2, a and b are indirectly connected via a different set of ground atoms $part(a, c_3), r(c_3, c_1), part(b, c_1)$. Spatial unity requires all pairs of objects to always be connected via some chain of ground atoms at each time-step, but it does not insist that it is the *same* set of ground atoms at each time-step. ◁

Examples 6, 7, and 8 provide different ways of interpreting the same sensory input. In Example 6, the sensors are interpreted as self-contained state machines. Here, there are no causal interactions between the sensors: each is an isolated

machine. In Examples 7 and 8, by contrast, there are causal interactions between the sensors. In Example 7, the *on* and *off* attributes move from left to right along the sensors. In Example 8, it is the sensors that move, not the attributes, moving from right to left. The difference between these two interpretations is in terms of what is moving and what is static.

Note that the interpretations of Examples 6, 7, and 8 have costs 16, 12, and 17 respectively. So the theory of Example 7, which invents an unseen object, is preferred to the other theories that posit more complex dynamics.

3.6. Properties of interpretations

In this section, we provide some results about the correctness properties of unified interpretations. The next theorem shows that all objects appearing in sensor readings satisfy object permanence.

Theorem 3. *For each sensory sequence $S = (S_1, ..., S_t)$ and each unified interpretation θ of S, for each object x that features in S (i.e. x appears in some ground atom $p(x)$ or $q(x, y)$ in some state S_i in S), for each state A_i in $\tau(\theta) = (A_1, A_2, ...)$, x features in A_i. In other words, if x features in any state in S, then x features in every state in $\tau(\theta)$.*

Proof. Let $\theta = (\phi, I, R, C)$ and $\phi = (T, O, P, V)$. Since object x features in sequence S, there exists some atom α involving x in some state S_j in $(S_1, ..., S_t)$. Since θ is an interpretation, $S \sqsubseteq \tau(\theta)$, and hence $\alpha \in (\tau(\theta))_j$. Consider the two possible forms of α:

1. $\alpha = p(x)$. Since θ satisfies *conceptual unity*, there must be a constraint involving p of the form $\forall X : t, p(X) \oplus q_1(X) ... \oplus q_n(X)$ in C. Since ϕ is suitable for S, $x \in O$ and $\kappa_O(x) = t$. Let $\tau(\theta) = (A_1, A_2, ...)$ and consider any A_i in $\tau(\theta)$. Since θ satisfies *static unity*, A_i satisfies each constraint in C and in particular $A_i \models \forall X : t, p(X) \oplus q_1(X) ... \oplus q_n(X)$. Since $\kappa_O(x) = t$, $A_i \models p(x) \oplus q_1(x) ... \oplus q_n(x)$. Hence $\{p(x), q_1(x), ..., q_n(x)\} \cap A_i \neq \emptyset$ i.e. x features in A_i.
2. $\alpha = q(x, y)$ for some y. Since θ satisfies *conceptual unity*, there must be a constraint involving q. This constraint can either be (i) a binary constraint of the form $\forall X : t_1, \forall Y : t_2, q(X, Y) \oplus p_1(X, Y) \oplus ... \oplus p_n(X, Y)$ or (ii) a uniqueness constraint of the form $\forall X : t_1, \exists! Y : t_2, q(X, Y)$.
   Considering first case (i), since ϕ is suitable for S, $x, y \in O$, $\kappa_O(x) = t_1$, and $\kappa_O(y) = t_2$. Again, let $\tau(\theta) = (A_1, A_2, ...)$ and consider any A_i in $\tau(\theta)$. Since θ satisfies *static unity*, A_i satisfies each constraint in C and in particular $A_i \models \forall X : t_1, \forall Y : t_2, q(X, Y) \oplus p_1(X, Y) \oplus ... \oplus p_n(X, Y)$. Since $\kappa_O(x) = t_1, \kappa_O(y) = t_2$, $A_i \models q(x, y) \oplus p_1(x, y) \oplus ... \oplus p_n(x, y)$. Hence $\{q(x, y), p_1(x, y), ..., p_n(x, y)\} \cap A_i \neq \emptyset$ i.e. x features in A_i.
   For case (ii), again let $\tau(\theta) = (A_1, A_2, ...)$ and consider any A_i in $\tau(\theta)$. Since θ satisfies *static unity*, A_i satisfies each constraint in C and in particular $A_i \models \forall X : t_1, \exists! Y : t_2, q(X, Y)$. Since $\kappa_O(x) = t_1$, $A_i \models \exists! Y : t_2, q(x, Y)$. Therefore there must be some y such that $\kappa_O(y) = t_2$ and $q(x, y) \in A_i$. □

Note that this proof can be extended to apply to latent objects as well. Any object appearing in any atom in some state A_i must also appear in every other state A_j.

Theorem 3 provides some guarantee that admissible interpretations that satisfy the unity conditions will always be acceptable in the minimal sense that they always provide some value for each sensor. This theorem is important because it justifies the claim that a unified interpretation will always be able to support prediction (of future values), retrodiction (of previous values), and imputation (of missing values).

Note that this theorem does not imply that the predicate of the atom in which x appears is one of the predicates appearing in the sensory sequence S. It is entirely possible that it is some distinct predicate that appears in ϕ but has never been observed in S. The following example illustrates this possibility.

Example 9. Suppose the sensory sequence is just $S = (\{p(a)\})$. Suppose the type signature (T, O, P, V) introduces another unary predicate q, i.e. $T = \{t\}$, $O = \{a\}$, $P = \{p(t), q(t)\}$, and $V = \{X : t\}$. Suppose our interpretation is (ϕ, I, R, C) where $I = \{p(a)\}$, $R = \{p(X) \ni q(X)\}$, and $C = \{\forall X : t, p(X) \oplus q(X)\}$. Here, $\tau(\theta) = (\{p(a)\}, \{q(a)\}, \{q(a)\}, \{q(a)\}, ...)$. Note that q is a new predicate that does not appear in the sensory input; q is a "peer" of p (in that they are connected by an xor constraint), but q was never observed. ◁

The next theorem is a form of "completeness", showing that every sensory sequence has some admissible interpretation that satisfies the unity conditions.

Theorem 4. *For every apperception task (S, ϕ, C) there exists some interpretation $\theta = (\phi', I, R, C')$ that makes sense of S, where ϕ' extends ϕ and $C' \supseteq C$.*

Proof. We construct a degenerate theory that simply memorizes the sequence it has seen. First, we define ϕ' given $\phi = (T, O, P, V)$. For each sensor x_i that features in S, $i = 1..n$, and each state S_j in S, $j = 1..m$, create a new unary predicate p^i_j. The intention is that $p^i_j(X)$ is true if X is the i'th object x_i at the j'th time-step. If $\kappa_O(x_i) = t$ then let $\kappa_P(p^i_j) = (t)$. For

each type $t \in T$, create a new variable X_t where $\kappa_V(X_t) = t$. Let $\phi' = (T, O, P', V')$ where $P' = P \cup \{p^i_j \mid i = 1..n, j = 1..m\}$, and $V' = V \cup \{X_t \mid t \in T\}$.

Second, we define $\theta = (\phi', I, R, C')$. Let the initial conditions I be:

$$\left\{ p^i_1(x_i) \mid i = 1..n \right\}$$

Let the rules R contain the following causal rules for $i = 1..n$ and $j = 1..m-1$ (where x_i is of type t):

$$p^i_j(X_t) \Rightarrow p^i_{j+1}(X_t)$$

together with the following static rules for each unary atom $q(x_i) \in S_j$:

$$p^i_j(X_t) \rightarrow q(X_t)$$

and the following static rules for each binary atom $r(x_i, x_k) \in S_j$ (where x_i is of type t and x_k is of type t'):

$$p^i_j(X_t) \wedge p^k_j(Y_{t'}) \rightarrow r(X_t, Y_{t'})$$

We augment C to C' by adding the following additional constraints. Let P_t be the unary predicates for all objects of type t:

$$P_t = \left\{ p^i_j \mid \kappa_O(x_i) = t, j = 1..m \right\}$$

Let $P_t = \{p'_1, ..., p'_k\}$. Then for each type t add a unary constraint:

$$\forall X_t : t, p'_1(X_t) \oplus ... \oplus p'_k(X_t)$$

It is straightforward to check that θ as defined satisfies the constraint of conceptual unity, that the constraints C' are satisfied by each state in $\tau(\theta)$, and that the sensory sequence is covered by $\tau(\theta)$. To satisfy spatial unity, add a new "world" object w of a new type t_w and for each type t add a relation $part_t(t, t_w)$ and a constraint $\forall X : t, \exists! Y : t_w, part_t(X, Y)$. For each object x of type t, add an initial condition atom $part_t(x, w)$ to I. Thus, all the conditions of unity are satisfied, and θ is a unified interpretation of S. □

Example 10. Consider the following apperception problem (S, ϕ, C). Suppose there is one sensor a with values *on* and *off*. Suppose the sensory sequence is $S_{1:7}$ where:

$$S_1 = \{on(a)\} \quad S_2 = \{off(a)\} \quad S_3 = \{on(a)\} \quad S_4 = \{off(a)\}$$
$$S_5 = \{on(a)\} \quad S_6 = \{off(a)\} \quad S_7 = \{on(a)\}$$

Let $\phi = (T, O, P, V)$ where $T = \{t\}$, $O = \{a : t\}$, $P = \{on(t), off(t)\}$, and $V = \{\}$. Clearly, ϕ is suitable for S. The constraints C are just $\{\forall X : t, on(X) \oplus off(X)\}$.

Applying Theorem 4, we generate 7 unary predicates $p_1, ..., p_7$. The type signature ϕ' for this interpretation is (T', O', P', V') where $T' = \{t, t_w\}$, $O' = \{a, w\}$, $P' = P \cup \{p_1(t), p_2(t), ..., p_7(t), part(t, t_w)\}$, and $V' = \{X : t, Y : t_w\}$. Our interpretation is (ϕ', I, R, C') where:

$$I = \left\{ \begin{array}{l} p_1(a) \\ part(a, w) \end{array} \right\} \quad R = \left\{ \begin{array}{l} p_1(X) \Rightarrow p_2(X) \\ p_2(X) \Rightarrow p_3(X) \\ ... \\ p_6(Y) \Rightarrow p_7(Y) \\ p_1(X) \rightarrow on(X) \\ p_2(X) \rightarrow off(X) \\ p_3(X) \rightarrow on(X) \\ p_4(X) \rightarrow off(X) \\ p_5(X) \rightarrow on(X) \\ p_6(X) \rightarrow off(X) \\ p_7(X) \rightarrow on(X) \end{array} \right\} \quad C' = \left\{ \begin{array}{l} \forall X : t,\ on(X) \oplus off(X) \\ \forall X : t,\ p_1(X) \oplus p_2(X) \oplus ... \oplus p_7(X) \\ \forall X : t,\ \exists! Y : t_w\ part(X, Y) \end{array} \right\} \quad \triangleleft$$

Note that the interpretation provided by Theorem 4 is degenerate and unilluminating: it treats each object entirely separately (failing to capture any regularities between objects' behaviour) and treats every time-step entirely separately (failing to capture any laws that hold over multiple time-steps). This unilluminating interpretation provides an *upper bound* on the complexity of the theory needed to make sense of the sensory sequence.

4. Computer implementation

The Apperception Engine is our system for solving apperception tasks.[21] Given as input an apperception task (S, ϕ, C), the engine searches for a type signature ϕ' and a theory $\theta = (\phi', I, R, C')$ where ϕ' extends ϕ, $C' \supseteq C$ and θ makes sense of S. In this section, we describe how it is implemented.

Definition 20. A **template** is a structure for circumscribing a large but finite set of theories. It is a type signature together with constants that bound the complexity of the rules in the theory. Formally, a template χ is a tuple $(\phi, N_{\rightarrow}, N_{\Rightarrow}, N_B)$ where ϕ is a type signature, $N_{\rightarrow}$ is the max number of static rules allowed in R, $N_{\Rightarrow}$ is the max number of causal rules allowed in R, and N_B is the max number of atoms allowed in the body of a rule in R.

Each template χ specifies a large (but finite) set of theories that conform to χ. Let $\Theta_{\chi,C} \subset \Theta$ be the subset of theories (ϕ, I, R, C') in Θ that conform to χ and where $C' \supseteq C$.

Algorithm 1: The Apperception Engine algorithm in outline.

input : (S, ϕ, C), an apperception task
output: θ^*, a unified interpretation of S

$(s^*, \theta^*) \leftarrow (max(float), nil)$

foreach *template* χ *extending* ϕ *of increasing complexity* **do**
  $\theta \leftarrow \text{argmin}_\theta\{cost(\theta) \mid \theta \in \Theta_{\chi,C}, S \sqsubseteq \tau(\theta), unity(\theta)\}$
  if $\theta \neq nil$ **then**
    $s \leftarrow cost(\theta)$
    if $s < s^*$ **then**
      $(s^*, \theta^*) \leftarrow (s, \theta)$
    end
  end
  if *exceeded processing time* **then**
    return θ^*
  end
end

Our method, presented in Algorithm 1, is an anytime algorithm that enumerates templates of increasing complexity. For each template χ, it finds the $\theta \in \Theta_{\chi,C}$ with lowest cost (see Definition 17) that satisfies the conditions of unity. If it finds such a θ, it stores it. When it has run out of processing time, it returns the lowest cost θ it has found from all the templates it has considered.

Note that the relationship between the complexity of a template and the cost of a theory satisfying the template is not always simple. Sometimes a theory of lower cost may be found from a template of higher complexity. This is why we cannot terminate as soon as we have found the first theory θ. We must keep going, in case we later find a lower cost theory from a more complex template.

The two non-trivial parts of this algorithm are the way we enumerate templates, and the way we find the lowest-cost theory θ for a given template χ. We consider each in turn.

4.1. Iterating through templates

We need to enumerate templates in such a way that every template is (eventually) visited by the enumeration. Since the objects, predicates, and variables are *typed* (see Definition 3), the acceptable ranges of O, P, and V depend on T. Because of this, our enumeration procedure is two-tiered: first, enumerate sets T of types; second, given a particular T, enumerate $(O, P, V, N_{\rightarrow}, N_{\Rightarrow}, N_B)$ tuples for that particular T. We cannot, of course, enumerate *all* $(O, P, V, N_{\rightarrow}, N_{\Rightarrow}, N_B)$ tuples because there are infinitely many. Instead, we specify a constant bound (n) on the number of tuples, and gradually increase that bound:

foreach (T, n) **do**
  emit n tuples of the form $(O, P, V, N_{\rightarrow}, N_{\Rightarrow}, N_B)$
end

In order to enumerate (T, n) pairs, we use a standard diagonalization procedure. See Table 1.

[21] The source code is available at https://github.com/RichardEvans/apperception.

Table 1
Enumerating (t, n) pairs. Row t means that there are t types in T, while column n means there are n tuples of the form $(O, P, V, N_{\rightarrow}, N_{\ni}, N_B)$ to enumerate. We increment n by 100. The entries in the table represent the order in which the (t, n) pairs are visited.

	100	200	300	400	...
1	1	2	4	7	...
2	3	5	8	...	
3	6	9	...		
4	10	...			
...	...				

Once we have a (T, n) pair, we need to emit n $(O, P, V, N_{\rightarrow}, N_{\ni}, N_B)$ tuples using the types in T. One way of enumerating k-tuples, where $k > 2$, is to use the diagonalization technique recursively: first enumerate pairs, then apply the diagonalization technique to enumerate pairs consisting of individual elements paired with pairs, and so on. But this recursive application will result in heavy biases towards certain k-tuples. Instead, we use the Haskell function `Universe.Helpers.choices` to enumerate n-tuples while minimizing bias. The `choices :: [[a]] -> [[a]]` function takes a finite number n of (possibly infinite) lists, and produces a (possibly infinite) list of n-tuples, generating a n-way Cartesian product that is guaranteed to eventually produce every such n-tuple.

We use `choices` to generate 6-tuples $(O, P, V, N_{\rightarrow}, N_{\ni}, N_B)$ tuples by creating six infinite streams: (i) $\mathcal{S}_O$: an infinite list of finite lists of typed objects, (ii) $\mathcal{S}_P$: an infinite list of finite lists of typed predicates, (iii) $\mathcal{S}_V$: an infinite list of finite lists of typed variables, (iv) $\mathcal{S}_{\rightarrow} = \{0, 1, ...\}$: the number of static rules, (v) $\mathcal{S}_{\ni} = \{0, 1, ...\}$: the number of causal rules, and (vi) $\mathcal{S}_B = \{0, 1, ...\}$: the max number of body atoms. Now when we pass this list of streams to the `choices` function, it produces an enumeration of the 6-way Cartesian product $\mathcal{S}_O \times \mathcal{S}_P \times \mathcal{S}_V \times \mathcal{S}_{\rightarrow} \times \mathcal{S}_{\ni} \times \mathcal{S}_B$.

Example 11. Recall the apperception problem from Example 1. There are two sensors a and b, and each sensor can be *on* or *off*.

$$S_1 = \{\} \quad S_2 = \{off(a), on(b)\} \quad S_3 = \{on(a), off(b)\} \quad S_4 = \{on(a), on(b)\} \quad S_5 = \{on(b)\}$$
$$S_6 = \{on(a), off(b)\} \quad S_7 = \{on(a), on(b)\} \quad S_8 = \{off(a), on(b)\} \quad S_9 = \{on(a)\} \quad S_{10} = \{\}$$

We shall start with an initial template $\chi_0 = (\phi = (T, O, P, V), N_{\rightarrow}, N_{\ni}, N_B)$, where $T = \{sensor, grid\}$, $O = \{a{:}sensor, b{:}sensor, g{:}grid\}$, $P = \{on(sensor), off(sensor), part(sensor, grid)\}$, $V = \{X{:}sensor, Y{:}sensor\}$, $N_{\rightarrow} = 1$, $N_{\ni} = 3$, and $N_B = 2$. We use the template enumeration procedure described above to generate increasingly complex templates $\chi_1, \chi_2, ...$, using χ_0 as a base. This produces the following augmented templates :

$$\Delta\chi_1 = (\emptyset, \emptyset, \emptyset, \{V_1{:}sensor\}, 0, 0, 0) \quad \Delta\chi_2 = (\emptyset, \emptyset, \emptyset, \emptyset, 0, 0, 1) \quad \Delta\chi_3 = (\emptyset, \emptyset, \{p_1(sensor)\}, \emptyset, 0, 0, 0)$$
$$\Delta\chi_4 = (\emptyset, \emptyset, \emptyset, \{V_1{:}sensor\}, 0, 0, 1) \quad \Delta\chi_5 = (\emptyset, \emptyset, \emptyset, \emptyset, 0, 0, 2) \quad \Delta\chi_6 = (\emptyset, \{o_1{:}sensor\}, \emptyset, \emptyset, 0, 0, 0)$$
$$\Delta\chi_7 = (\emptyset, \emptyset, \{p_1(sensor)\}, \emptyset, 0, 0, 1) \quad \Delta\chi_8 = (\emptyset, \emptyset, \emptyset, \{V_1{:}sensor\}, 0, 0, 2) \quad \Delta\chi_9 = (\emptyset, \emptyset, \emptyset, \emptyset, 0, 0, 2)$$
$$\Delta\chi_{10} = (\emptyset, \emptyset, \{p_1(sensor)\}, \{V_1{:}sensor\}, 0, 0, 0) \quad \ldots \quad \ldots$$

In the list above, we display the change from the base template χ_0, so $\Delta\chi_i$ means the changes in template χ_i from the base template χ_0. Each template $\chi = (\phi = (T, O, P, V), N_{\rightarrow}, N_{\ni}, N_B)$ is flattened as a 7-tuple $(T, O, P, V, N_{\rightarrow}, N_{\ni}, N_B)$.

Many of these templates do not have the expressive resources to find a unified interpretation. But some do. The first solution the Apperception Engine finds has the following type signature (the new elements are in bold):

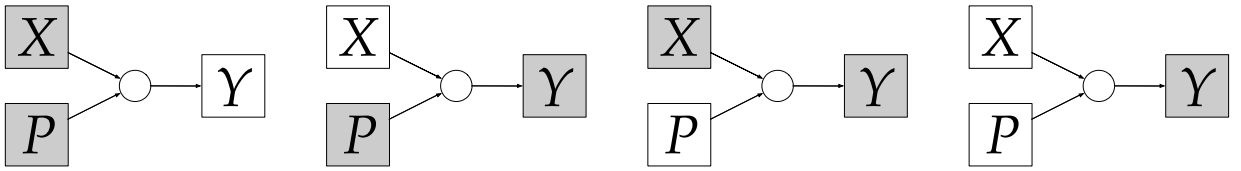
$$T = \left\{ \begin{array}{l} grid \\ sensor \end{array} \right\} \quad O = \left\{ \begin{array}{l} a : sensor \\ b : sensor \\ g : grid \end{array} \right\} \quad P = \left\{ \begin{array}{l} \mathbf{p_1}(sensor) \\ \mathbf{p_2}(sensor) \\ off(sensor) \\ on(sensor) \\ part(sensor, grid) \end{array} \right\} \quad V = \left\{ \begin{array}{l} S : sensor \\ S2 : sensor \end{array} \right\}$$

together with the following theory $\theta = (\phi, I, R, C)$, where:

$$I = \left\{ \begin{array}{l} p_1(a) \\ p_2(b) \\ on(a) \\ part(a, g) \\ part(b, g) \end{array} \right\} \quad R = \left\{ \begin{array}{l} p_2(S) \rightarrow on(S) \\ p_2(S) \ni p_1(S) \\ p_1(S) \wedge on(S) \ni off(S) \\ off(S) \wedge p_1(S) \ni p_2(S) \end{array} \right\} \quad C = \left\{ \begin{array}{l} \forall X : sensor,\ p_1(X) \oplus p_2(X) \\ \forall X : sensor,\ \exists! Y : grid,\ part(X, Y) \end{array} \right\}$$

This solution uses the invented predicates p_1 and p_2 to represent two states of a state-machine. This is recognisable as a compressed version of Example 6 above.

Later, the Apperception Engine finds another solution using the type signature $\phi = (T, O, P, V)$ (again, the augmented parts of the type signature are in bold):



(a) Deduction (b) Abduction (c) Induction (d) Abduction and induction

Fig. 1. The varieties of inference. Here, shaded elements are given, and unshaded elements must be generated. X and Y are sets of facts, while P is a set of rules for transforming X into Y.

$$T = \left\{ \begin{array}{l} grid \\ sensor \end{array} \right\} \quad O = \left\{ \begin{array}{l} a : sensor \\ b : sensor \\ g : grid \\ \mathbf{o_1} : sensor \end{array} \right\} \quad P = \left\{ \begin{array}{l} \mathbf{r_1}(sensor, sensor) \\ off(sensor) \\ on(sensor) \\ part(sensor, grid) \end{array} \right\} \quad V = \left\{ \begin{array}{l} S : sensor \\ S2 : sensor \end{array} \right\}$$

together with the following theory $\theta = (\phi, I, R, C)$, where:

$$I = \left\{ \begin{array}{ll} off(o_1) & on(a) \\ r_1(a, o_1) & r_1(b, a) \\ r_1(o_1, b) & part(a, grid) \\ part(b, grid) & part(o_1, grid) \end{array} \right\} \qquad R = \left\{ \begin{array}{l} off(S) \wedge r_1(S, S2) \rightarrow on(S2) \\ off(S2) \wedge r_1(S, S2) \wedge on(S) \ni off(S) \end{array} \right\}$$

$$C = \left\{ \begin{array}{l} \forall X{:}sensor,\ \exists! Y{:}grid,\ part(X, Y) \\ \forall X{:}sensor,\ \exists! Y{:}sensor,\ r_1(X, Y) \end{array} \right\}$$

Here, it has constructed an invented object o_1:*sensor* and posited a one-dimensional spatial relationship r_1 between the three sensors. This solution is recognisable as a variant of Example 7 above. ◁

4.2. Finding the best theory from a template

The most complex part of Algorithm 1 is:

$$\theta \leftarrow \underset{\theta}{\text{argmin}}\{cost(\theta) \mid \theta \in \Theta_{\chi,C}, S \sqsubseteq \tau(\theta), unity(\theta)\}$$

Here, we search for a theory θ with the lowest cost (see Definition 17) such that θ conforms to the template χ and includes the constraints in C, such that $\tau(\theta)$ covers S, and θ satisfies the conditions of unity. In this sub-section, we explain in outline how this works.

Our approach combines abduction and induction to generate a unified interpretation θ.[22] See Fig. 1. Here, $X \subseteq \mathcal{G}$ is a set of facts (ground atoms), $P : \mathcal{G} \rightarrow \mathcal{G}$ is a procedure for generating the consequences of a set of facts, and $Y \subseteq \mathcal{G}$ is the result of applying P to X. If X and P are given, and we wish to generate Y, then we are performing *deduction*. If P and Y are given, and we wish to generate X, then we are performing *abduction*. If X and Y are given, and we wish to generate P, then we are performing *induction*. Finally, if only Y is given, and we wish to generate both X and P, then we are jointly performing abduction and induction. This is what the Apperception Engine does.[23]

Our method is described in Algorithm 2. In order to jointly abduce a set I (of initial conditions) and induce sets R and C (of rules and constraints), we implement a Datalog$^{\ni}$ interpreter in ASP. See Section 4.3 for the details. This interpreter takes a set I of atoms (represented as a set of ground ASP terms) and sets R and C of rules and constraints (represented again as a set of ground ASP terms), and computes the trace of the theory $\tau(\theta) = (S_1, S_2, ...)$ up to a finite time limit.

Concretely, we implement the interpreter as an ASP program π_τ that computes $\tau(\theta)$ for theory θ. We implement the conditions of unity as ASP constraints in a program π_u. We implement the cost minimization as an ASP program π_m that counts the number of atoms in each rule plus the number of initialisation atoms in I, and uses an ASP weak constraint [60] to minimize this total. Then we generate ASP programs representing the sequence S, the initial conditions, the rules and constraints. We combine the ASP programs together and ask the ASP solver (`clingo` [73]) to find a lowest cost solution. (There may be multiple solutions that have equally lowest cost; the ASP solver chooses one of the optimal answer sets.) We extract a readable interpretation θ from the ground atoms of the answer set. In Section 4.3, we explain how Algorithm 2 is implemented in ASP. In Section 4.4, we evaluate the computational complexity. In Section 4.5, we describe the various optimisations used to prune the search. In Section 5.2.4, we compare with ILASP, a state-of-the-art ILP system.

[22] For related work that also uses abduction, see [11,72,19].

[23] At a high level, our system is similar to XHAIL [11]. But there are a number of differences. First, our program P contains causal rules and constraints as well as standard Horn clauses. Second, our conclusion Y is an infinite sequence $(S_1, S_2, ...)$ of sets, rather than a single set. Third, we add additional filters on acceptable theories in the form of the unity conditions (see Definition 11).

Algorithm 2: Finding the lowest cost θ for sequence S and template χ. Here, π_τ computes the trace, π_u checks that the unity conditions are satisfied, and π_m minimizes the cost of θ.

input : S, a sensory sequence
input : $\chi = (\phi, N_\rightarrow, N_\ni, N_B)$, a template
input : C, a set of constraints on the predicates of the sensory sequence
output: θ, the simplest unified interpretation of S that conforms to χ

$\pi_S \leftarrow$ gen_input(S)
$\pi_I \leftarrow$ gen_inits(ϕ)
$\pi_R \leftarrow$ gen_rules($\phi, N_\rightarrow, N_\ni, N_B$)
$\pi_C \leftarrow$ gen_constraints(ϕ, C)
$\Pi \leftarrow \pi_\tau \cup \pi_u \cup \pi_m \cup \pi_S \cup \pi_I \cup \pi_R \cup \pi_C$
$A \leftarrow$ clingo(Π)
if satisfiable(A) **then**
  $\theta \leftarrow$ extract(A)
  return θ
end
return *nil*

4.3. The ASP encoding

Our Datalog$^\ni$ interpreter is written in ASP. All elements of Datalog$^\ni$, including variables, are represented by ASP constants. A variable X is represented by a constant `var_x`, and a predicate p is represented by a constant `c_p`. Elements of the target language are *reified* in ASP, so an unground atom $p(X)$ of Datalog$^\ni$ is represented by a term `s(c_p, var_x)`, and a rule is represented by a set of ground atoms for the body, and a single ground atom for the head. For example, the static rule $p(X) \wedge q(X, Y) \rightarrow r(Y)$ is represented as:

```
rule_body(r1, s(c_p, var_x)).
rule_body(r1, s2(c_q, var_x, var_y)).
rule_head_static(r1, s(c_r, var_y)).
```

The causal rule $on(X) \ni off(X)$ is represented as:

```
rule_body(r2, s(c_on, var_x)).
rule_head_causes(r2, s(c_off, var_x)).
```

Given a type signature ϕ, we construct ASP terms that represent every well-typed unground atom in U_ϕ, and wrap these terms in the `is_var_atom` predicate. For example, to represent that $r(C, C_2)$ is a well-typed unground atom, we write `is_var_atom(atom(c_r, var_c, var_c2))`. Similarly, we construct ASP terms that represent every well-typed ground atom in G_ϕ using the `is_ground_atom` predicate.

The initial conditions $I \subseteq G_\phi$ are represented by the `init` predicate. For example, to represent that $I = \{p(a), r(b, c)\}$, we write:

```
init(s(c_p, c_a)).
init(s2(c_r, c_b, c_c)).
```

Our meta-interpreter π_τ implements $\tau : \Theta \rightarrow (2^\mathcal{G})^*$ from Definition 9. We use `holds(a, t)` to represent that atom `a` is true at time `t` (i.e. $a \in S_t$ where $\tau(\theta) = (S_1, S_2, ..., S_t, ...)$).

```
% initial conditions
holds(A, T) :-
    init(A),
    init_time(T).

% update for static rules
holds(GA, T) :-
    rule_head_static(R, VA),
    eval_body(R, Subs, T),
    ground_atom(VA, GA, Subs).

% update for causal rules
holds(GC, T+1) :-
    rule_head_causes(R, VC),
    eval_body(R, Subs, T),
    ground_atom(VC, GC, Subs),
    is_time(T+1).
```

```
% frame axiom
holds(S, T+1) :-
    holds(S, T),
    is_time(T+1),
    not -holds(S, T+1).
```

Since $\tau(\theta)$ is an infinite sequence $(A_1, A_2, ...)$, we cannot compute the whole of it. Instead, we only compute the sequence up to the max time of the original sensory sequence S.

Note that the frame axiom uses negation as failure and strong negation [64] to check that some other incompatible atom has not already been added. Thus, the frame axiom is not restrictive and can be overridden as needed by the causal rules, to handle predicates that are not inertial.

The conditions of unity described in Section 3.3 are represented directly as ASP constraints in π_u. For example, spatial unity is encoded as:

```
:-  spatial_unity_counterexample(X, Y, T).
```

Here, there is a counterexample to spatial unity at time `T` if objects `X` and `Y` are not `related`:

```
spatial_unity_counterexample(X, Y, T) :-
    is_object(X),
    is_object(Y),
    is_time(T),
    not related(X, Y, T).
```

Here, `related` is the reflexive symmetric transitive closure of the relation holding between `X` and `Y` if there is some relation `R` connecting them. Note that `related` can quantify over relations because the Datalog$^{\ni}$ atoms and predicates have been reified into terms:

```
related(X, Y, T) :-
    holds(s2(R, X, Y), T).

related(X, X, T) :-
    is_object(X),
    is_time(T).

related(X, Y, T) :- related(Y, X, T).

related(X, Y, T) :-
    related(X, Z, T),
    related(Z, Y, T).
```

When constructing a theory $\theta = (\phi, I, R, C)$, the solver needs to choose which ground atoms to use as initial conditions in I, which static and causal rules to include in R, and which xor or uniqueness conditions to use as conditions in C.

To allow the solver to choose what to include in I, we add the ASP choice rule to π_I:

```
{ init(A) } :- is_ground_atom(A).
```

To allow the solver to choose which rules to include in R, we add the following clauses to π_R:

```
0 { rule_body(R, VA) : is_var_atom(VA) } k_max_body :- is_rule(R).
1 { rule_head_static(R, VA) : is_var_atom(VA) } 1 :- is_static_rule(R).
1 { rule_head_causes(R, VA) : is_var_atom(VA) } 1 :- is_causes_rule(R).
```

Here, `k_max_body` is the N_B parameter of the template that specifies the max number of body atoms in any rule. The number of rules satisfying `is_static_rule` and `is_causes_rule` is determined by the parameters $N_{\ni}$ and $N_{\rightarrow}$ in the template (see Definition 20).

The ASP program π_m minimizes the cost of the theory θ (see Definition 17) by using weak constraints [60]:

```
:~ rule_body(R,A). [1@1,R,A]

:~ rule_head_static(R,A). [1@1,R,A]

:~ rule_head_causes(R,A). [1@1,R,A]

:~ init(A). [1@1,A]
```

Table 2
The number of ground clauses in the ASP encoding of Algorithm 2.

Predicate	# ground clauses
`holds`	$\lvert\Sigma_\phi\rvert \cdot \lvert U_\phi\rvert \cdot (N_\rightarrow + N_\ni) \cdot t + \lvert G_\phi\rvert \cdot (t+1)$
`eval_atom(VA, Subs, T)`	$\lvert\Sigma_\phi\rvert \cdot \lvert U_\phi\rvert \cdot t$
`eval_body(R, Subs, T)`	$\lvert\Sigma_\phi\rvert \cdot (N_\rightarrow + N_\ni) \cdot t$

Table 3
The number of ground clauses in the ASP encoding of Algorithm 2.

Clause	# ground clauses	# atoms

`holds(GA, T) :-`
`  rule_head_static(R, VA),`
`  eval_body(R, Subs, T),`
`  ground_atom(VA, GA, Subs).`
	$\lvert\Sigma_\phi\rvert \cdot \lvert U_\phi\rvert \cdot N_\rightarrow \cdot t$	4

`holds(GA, T+1) :-`
`  rule_head_causes(R, VA),`
`  eval_body(R, Subs, T),`
`  ground_atom(VA, GA, Subs).`
	$\lvert\Sigma_\phi\rvert \cdot \lvert U_\phi\rvert \cdot N_\ni \cdot t$	4

`eval_body(R, Subs, T) :-`
`  is_rule(R),`
`  is_subs(Subs),`
`  is_time(T),`
`  eval_atom(V, Subs, T) :`
`  rule_body(R, V).`
	$\lvert\Sigma_\phi\rvert \cdot (N_\rightarrow + N_\ni) \cdot t$	$\lvert U_\phi\rvert + 4$

4.4. Complexity

This section describes the complexity of Algorithm 2.

We assume basic concepts and standard terminology from complexity theory. Let P be the class of problems that can be solved in polynomial time by a deterministic Turing machine, NP be the class of problems solved in polynomial time by a non-deterministic Turing machine, and $EXPTIME$ be the class of problems solved in time 2^{n^d} by a deterministic Turing machine. Let $\Sigma_{i+1}^P = NP^{\Sigma_i^P}$ be the class of problems that can be solved in polynomial time by a non-deterministic Turing machine with a Σ_i^P oracle. Finding a solution to an ASP program is in NP [74,75], while finding an optimal solution to an ASP program with weak constraints is in Σ_2^P [76,77].

Since deciding whether a non-disjunctive ASP program has a solution is in NP [74,75], our ASP encoding of Algorithm 2 shows that finding a unified interpretation θ for a sequence given a template is in NP. Since verifying whether a solution to an ASP program with preferences is indeed optimal is in Σ_2^P [76,77], our ASP encoding shows that finding the lowest cost theory is in Σ_2^P.

However, the standard complexity results assume the ASP program has *already been grounded into a set of propositional clauses*. To really understand the space and time complexity of Algorithm 2, we need to examine how the set of ground atoms in the ASP encoding grows as a function of the parameters in the template $\chi = (\phi = (T, O, P, V), N_\rightarrow, N_\ni, N_B)$.

Observe that, since we restrict ourselves to unary and binary predicates, the number of ground and unground atoms is a small polynomial function of the type signature parameters[24]:

$$|G_\phi| \leq |P| \cdot |O|^2$$
$$|U_\phi| \leq |P| \cdot |V|^2$$

But note that the number of substitutions Σ_ϕ that is compatible with the signature ϕ is an exponential function of the number of variables V:

$$|\Sigma_\phi| \leq |O|^{|V|}$$

Thus, in our ASP encoding, finding the lowest cost theory is in Σ_2^P, but the number of ground propositional clauses grows exponentially with $|O|$, the number of variables allowed in a rule. Table 2 shows the number of ground clauses for the three most expensive clauses.

[24] The actual numbers will be less than these bounds because type-checking rules out certain combinations.

4.5. Optimization

Because of the combinatorial complexity of the apperception task, we had to introduce a number of optimizations to get reasonable performance on even the simplest of domains.

4.5.1. Reducing grounding with type checking

We use the type signature ϕ to dramatically restrict the set of ground atoms (G_ϕ), the unground atoms (U_ϕ), the substitutions (Σ_ϕ), and rules R_ϕ. Type-checking has been shown to drastically reduce the search space in program synthesis tasks [78].

4.5.2. Symmetry breaking

We use symmetry breaking to remove candidates that are equivalent. We remove programs that are equivalent up to a variable renaming by using a strict ordering on variables. We also remove programs that are equivalent up to a reordering of the rules by using a strict ordering on unground atoms.

4.5.3. Adding redundant constraints

ASP programs can be significantly optimized by adding redundant constraints (constraints that are provably entailed by the other clauses in the program) [79]. We speeded up solving time (by about 30%) by adding the following redundant constraints:

```
:-  init(A),
    init(B),
    incompossible(A, B).

:-  rule_body(R, A),
    rule_body(R, B),
    incompossible_var_atoms(A, B).

:-  rule_body(R, A),
    rule_head_static(R, A).

:-  rule_body(R, A),
    rule_head_causes(R, A).

:-  rule_body(R, A),
    rule_head_static(R, B),
    incompossible_var_atoms(A, B).

:-  rule_body(R, A),
    rule_head_causes(R, B),
    incompossible_var_atoms(A, B).
```

5. Experiments

5.1. Five experimental domains

To evaluate the generality of our system, we tested it in a variety of domains: elementary (one-dimensional) cellular automata, drum rhythms and nursery tunes, sequence induction tasks, multi-modal binding tasks, and occlusion tasks. These particular domains were chosen because they represent a diverse range of tasks that are simple for humans but are hard for state-of-the-art machine learning systems. The tasks were chosen to highlight the difference between mere perception (the classification tasks that machine learning systems already excel at) and apperception (assimilating information into a coherent integrated theory, something traditional machine learning systems are not designed to do).

5.1.1. Results

We implemented the Apperception Engine in Haskell and ASP. We used `clingo` [73] to solve the ASP programs generated by our system. We ran all experiments with a time-limit of 4 hours on a standard Unix desktop.

Our experiments (on the prediction task) are summarised in Table 4. Note that our accuracy metric for a single task is rather exacting: the model is accurate (Boolean) on a task iff *every* hidden sensor value is predicted correctly.[25] It does

[25] The reason for using this strict notion of accuracy is that, as the domains are deterministic and noise-free, there is a simplest possible theory that explains the sensory sequence. In such cases where there is a correct answer, we wanted to assess whether the system found that correct answer exactly – not whether it was fortunate enough to come close while misunderstanding the underlying dynamics.

Table 4
Results for prediction tasks on five domains. We show the mean information size of the sensory input, to stress the scantiness of our sensory sequences. We also show the mean information size of the held-out data. Our metric of accuracy for prediction tasks is whether the system predicted *every* sensor's value correctly.

Domain	Tasks (#)	Memory (megs)	Input size (bits)	Held out size (bits)	Accuracy (%)
ECA	256	473.2	154.0	10.7	97.3%
Rhythm & music	30	2172.5	214.4	15.3	73.3%
Seek Whence	30	3767.7	28.4	2.5	76.7%
Multi-modal binding	20	1003.2	266.0	19.1	85.0%
Occlusion	20	604.3	109.2	10.1	90.0 %

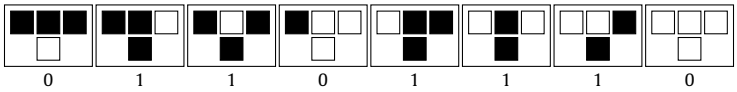


Fig. 2. Updates for ECA rule 110. The top row shows the context: the target cell together with its left and right neighbour. The bottom row shows the new value of the target cell given the context. A cell is black if it is on and white if it is off.

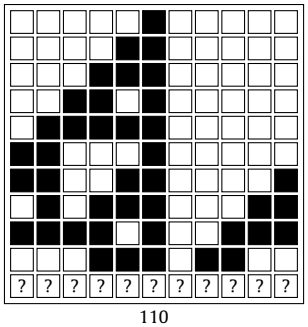


Fig. 3. One trajectory for Rule 110. Each row represents the state of the ECA at one time-step. In this prediction task, the bottom row (representing the final time-step) is held out.

not score any points for predicting most of the hidden values correctly. As can be seen from Table 4, our system is able to achieve good accuracy across all five domains.

5.1.2. Elementary cellular automata

An Elementary Cellular Automaton (ECA) [80,81] is a one-dimensional Cellular Automaton. The world is a circular array of cells. Each cell can be either *on* or *off*. The state of a cell depends only on its previous state and the previous state of its left and right neighbours.

Fig. 2 shows one set of ECA update rules.[26] Each update specifies the new value of a cell based on its previous left neighbour, its previous value, and its previous right neighbour. The top row shows the values of the left neighbour, previous value, and right neighbour. The bottom row shows the new value of the cell. There are 8 updates, one for each of the 2^3 configurations. In the diagram, the leftmost update states that if the left neighbour is *on*, and the cell is *on*, and its right neighbour is *on*, then at the next time-step, the cell will be turned *off*. Given that each of the 2^3 configurations can produce *on* or *off* at the next time-step, there are $2^{2^3} = 256$ total sets of update rules.

Given update rules for each of the 8 configurations, and an initial starting state, the trajectory of the ECA is determined. Fig. 3 shows the state sequence for Rule 110 above from one initial starting state of length 11.

In our experiments, we attach sensors to each of the 11 cells, produce a sensory sequence, and then ask our system to find an interpretation that makes sense of the sequence. For example, for the state sequence of Fig. 3, the sensory sequence is $(S_1, ..., S_{10})$ where:

$$S_1 = \{off(c_1), off(c_2), off(c_3), off(c_4), off(c_5), on(c_6), off(c_7), off(c_8), off(c_(), off(c_{10}), off(c_{11})\}$$

$$S_2 = \{off(c_1), off(c_2), off(c_3), off(c_4), on(c_5), on(c_6), off(c_7), off(c_8), off(c_(), off(c_{10}), off(c_{11})\}$$

[26] This particular set of update rules is known as Rule 110. Here, 110 is the decimal representation of the binary 01101110 update rule, as shown in Fig. 2. This rule has been shown to be Turing-complete [81].

$$S_3 = \{off(c_1), off(c_2), off(c_3), on(c_4), on(c_5), on(c_6), off(c_7), off(c_8), off(c_(), off(c_{10}), off(c_{11})\}$$

$$S_4 = \{off(c_1), off(c_2), on(c_3), on(c_4), off(c_5), on(c_6), off(c_7), off(c_8), off(c_(), off(c_{10}), off(c_{11})\}$$

$$S_5 = \{off(c_1), on(c_2), on(c_3), on(c_4), on(c_5), on(c_6), off(c_7), off(c_8), off(c_(), off(c_{10}), off(c_{11})\}$$

Note that we do *not* provide the spatial relation between cells. The system does not know that e.g. cell c_1 is directly to the left of cell c_2.

Results. Given the 256 ECA rules, all with the same initial configuration, we treated the trajectories as a prediction task and applied our system to it. Our system was able to predict 249/256 correctly. In each of the 7/256 failure cases, the Apperception Engine found a unified interpretation, but this interpretation produced a prediction which was not the same as the oracle. For example, the dynamics found for Fig. 3 are:

$$R = \left\{ \begin{array}{l} r(X,Y) \wedge on(X) \wedge off(Y) \supset on(Y) \\ r(X,Y) \wedge r(Y,Z) \wedge on(X) \wedge on(Z) \wedge on(Y) \supset off(Y) \end{array} \right\}$$

These rules exactly capture the dynamics of the cells that *change*. The other cells retain their value from the previous time-step, according to the frame axiom of Definition 9.

The initial conditions found by the Apperception Engine describe the initial values of the cells, and also specify the latent r relation between cells:

$$I = \left\{ \begin{array}{lllllllllll} off(c_1) & off(c_2) & off(c_3) & off(c_4) & off(c_5) & on(c_6) & off(c_7) & off(c_8) & off(c_9) & off(c_{10}) & off(c_{11}) \\ r(c_1,c_{11}) & r(c_2,c_1) & r(c_3,c_2) & r(c_4,c_3) & r(c_5,c_4) & r(c_6,c_5) & r(c_7,c_6) & r(c_8,c_7) & r(c_9,c_8) & r(c_{10},c_9) & r(c_{11},c_{10}) \end{array} \right\}$$

Here, the system uses $r(X,Y)$ to mean that cell Y is immediately to the right of cell X. Note that *the system has constructed the spatial relation itself*. It was not given the spatial relation r between cells. All it was given was the sensor readings of the 11 cells. It constructed the spatial relationship r between the cells in order to make sense of the data.

5.1.3. Drum rhythms and nursery tunes

We also tested our system on simple melodies and rhythms. Here, each sensor is an auditory receptor that is tuned to listen for a particular note or drum beat. In the tune tasks, there is one sensor for *C*, one for *D*, one for *E*, all the way to *HighC*. (There are no flats or sharps.) In the rhythm tasks, there is one sensor listening out for bass drum, one for snare drum, and one for hi-hat. Each sensor can distinguish four loudness levels, between 0 and 3. When a note is pressed, it starts at max loudness (3), and then decays down to 0. Multiple notes can be pressed simultaneously.

For example, the *Twinkle Twinkle Little Star* tune generates the following sensor readings (assuming 8 time-steps for a bar):

$$S_1 = \{v(s_c,3), v(s_d,0), v(s_e,0), v(s_f,0), v(s_g,0), v(s_a,0), v(s_b,0), v(s_{c*},0)\}$$

$$S_2 = \{v(s_c,2), v(s_d,0), v(s_e,0), v(s_f,0), v(s_g,0), v(s_a,0), v(s_b,0), v(s_{c*},0)\}$$

$$S_3 = \{v(s_c,3), v(s_d,0), v(s_e,0), v(s_f,0), v(s_g,0), v(s_a,0), v(s_b,0), v(s_{c*},0)\}$$

$$S_4 = \{v(s_c,2), v(s_d,0), v(s_e,0), v(s_f,0), v(s_g,0), v(s_a,0), v(s_b,0), v(s_{c*},0)\}$$

$$S_5 = \{v(s_c,1), v(s_d,0), v(s_e,0), v(s_f,0), v(s_g,3), v(s_a,0), v(s_b,0), v(s_{c*},0)\}$$

$$S_6 = \{v(s_c,0), v(s_d,0), v(s_e,0), v(s_f,0), v(s_g,2), v(s_a,0), v(s_b,0), v(s_{c*},0)\}$$

...

Results. Recall that our accuracy metric is stringent and only counts a prediction as accurate if *every* sensor's value is predicted correctly. In the rhythm and music domain, this means the Apperception Engine must correctly predict the loudness value (between 0 and 3) for each of the sound sensors. There are 8 sensors for tunes and 3 sensors for rhythms. When we tested the Apperception Engine on the 20 drum rhythms and 10 nursery tunes, our system was able to predict 22/30 correctly. Note that the interpretations found are large and complex programs by the standards of state-of-the-art ILP systems. In *Three Blind Mice*, for example, the interpretation contained 10 update rules and 34 initialisation atoms making a total of 44 clauses. The reason we are able to find such large programs is that our ASP encoding represents the space of possible programs much more efficiently than other encodings (see Section 5.2.4).

5.1.4. Seek Whence *and C-test sequence induction intelligence tests*

Hofstadter introduced the **Seek Whence**[27] domain in [24]. The task is, given a sequence $s_1, ..., s_t$ of symbols, to predict the next symbol s_{t+1}. Typical *Seek Whence* tasks include[28]:

[27] The name is a pun on "sequence". See also the related Copycat domain [82].

[28] Hofstadter used natural numbers, but we transpose the sequences to letters, to bring them in line with the Thurstone letter completion problems [83] and the C-test [84].

a,a,b,a,b,c,a,b,c,d,a, ...	**a,b,c,d,e, ...**
b,a,b,b,b,b,b,c,b,b,d,b,b,e, ...	**a,b,b,c,c,c,d,d,d,d,e, ...**
a,f,e,f,a,f,e,f,a,f,e,f,a, ...	**b,a,b,b,b,c,b,d,b,e, ...**
a,b,b,c,c,d,d,e,e, ...	**a,b,c,c,d,d,e,e,e,f,f,f, ...**
f,a,f,b,f,c,f,d,f, ...	**a,f,e,e,f,a,a,f,e,e,f,a,a, ...**
b,b,b,c,c,b,b,b,c,c,b,b,b,c,c, ...	**b,a,a,b,b,b,a,a,a,a,a,b,b,b,b,b, ...**
b,c,a,c,a,c,b,d,b,d,b,c,a,c,a, ...	**a,b,b,c,c,d,d,e,e,f,f, ...**
a,a,b,a,b,c,a,b,c,d,a,b,c,d,e, ...	**b,a,c,a,b,d,a,b,c,e,a,b,c,d,f, ...**
a,b,a,c,b,a,d,c,b,a,e,d,c,b, ...	**c,b,a,b,c,b,a,b,c,b,a,b,c,b, ...**
a,a,a,b,b,c,e,f,f, ...	**a,a,b,a,a,b,c,b,a,a,b,c,d,c,b, ...**
a,a,b,c,a,b,b,c,a,b,c,c,a,a,a, ...	**a,b,a,b,a,b,a,b,a, ...**
a,c,b,d,c,e,d, ...	**a,c,f,b,e,a,d, ...**
a,a,f,f,e,e,d,d, ...	**a,a,a,b,b,b,c,c, ...**
a,a,b,b,f,a,b,b,e,a,b,b,d, ...	**f,a,d,a,b,a,f,a,d,a,b,a, ...**
a,b,a,f,a,a,e,f,a, ...	**b,a,f,b,a,e,b,a,d, ...**

Fig. 4. Sequences from *Seek Whence* and the C-test.

- **b, b, b, c, c, b, b, b, c, c, b, b, b, c, c, ...**
- **a, f, b, f, f, c, f, f, f, d, f, f, ...**
- **b, a, b, b, b, b, b, c, b, b, d, b, b, e, b, ...**

Hofstadter called the third sequence the "theme song" of the *Seek Whence* project because of its difficulty. There is a "perceptual mirage" in the sequence because of the sub-sequence of five *b*'s in a row that makes it hard to see the intended pattern: $(b, x, b)^*$ for ascending x.

Results. Given the 30 *Seek Whence* sequences, we treated the trajectories as a prediction task and applied our system to it. Our system was able to predict 23/30 correctly. For the 7 failure cases, 4 of them were due to the system not being able to find any unified interpretation within the memory and time limits, while in 3 of them, the system found a unified interpretation that produced the "incorrect" prediction (Fig. 4).

The first key point we want to emphasise here is that our system was able to achieve human-level performance[29] on these tasks without hand-coded domain-specific knowledge.[30] This is a *general* system for making sense of sensory data that, when applied to the *Seek Whence* domain, is able to solve these particular problems. The second point we want to stress is that our system did not learn to solve these sequence induction tasks after seeing many previous examples.[31] On the contrary: our system had never seen any such sequences before; it confronts each sequence *de novo*, without prior experience. This system is, to the best of our knowledge, the first such general system that is able to achieve such a result.

5.1.5. Binding tasks

The binding problem [87] is the task of recognising that information from different sensory modalities should be collected together as different aspects of a single external object. For example, you hear a buzzing in your auditory field and you see an insect in your visual field. How do you associate the buzzing and the insect-appearance as aspects of one single object?

To investigate how our system handles such binding problems, we tested it on the following multi-modal variant of the ECA described above. Here, there are two types of sensor. The light sensors have just two states: black and white, while the touch sensors have four states: fully un-pressed (0), fully pressed (3), and two intermediate states (1, 2). After a touch sensor is fully pressed (3), it slowly depresses, going from states 2 to 1 to 0 over 3 time-steps. In this example, we chose Rule 110 (the Turing-complete ECA rule) with the same initial configuration as in Fig. 3, as described earlier. In this multi-modal variant, there are 11 light sensors, one for each cell in the ECA, and two touch sensors on cells 3 and 11. See Fig. 5.

Results. We ran 20 multi-modal binding experiments, with different ECA rules, different initial conditions, and the touch sensors attached to different cells. The engine achieved 85% accuracy.

5.1.6. Occlusion tasks

Neural nets that predict future sensory data conditioned on past sensory data struggle to solve occlusion tasks because it is hard to inject into them the prior knowledge that objects persist over time. Our system, by contrast, was designed to posit latent objects that persist over time.

To test our system's ability to solve occlusion problems, we generated a set of tasks of the following form: there is a 2D grid of cells in which objects move horizontally. Some move from left to right, while others move from right to left, with wrap around when they get to the edge of a row. The objects move at different speeds. Each object is placed in its own row, so there is no possibility of collision. There is an "eye" placed at the bottom of each column, looking up. Each eye can only

[29] See Meredith [85] for empirical results 25 students on the "Blackburn dozen" *Seek Whence* problems.

[30] The one piece of domain-specific knowledge we inject is the successor relation between the letters *a*, *b*, *c*, ...

[31] Machine learning approaches to these tasks need thousands of examples before they can learn to predict. See for example [86].

l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}	l_{11}	t_1	t_2
W	W	W	W	W	B	W	W	W	W	W	0	0
W	W	W	W	B	B	W	W	W	W	W	0	0
W	W	W	B	B	B	W	W	W	W	W	0	0
W	W	B	B	W	B	W	W	W	W	W	3	0
W	B	B	B	B	B	W	W	W	W	W	3	0
B	B	W	W	W	B	W	W	W	W	W	2	0
B	B	W	W	B	B	W	W	W	W	B	1	3
W	B	W	B	B	B	W	W	W	B	B	0	3
B	B	B	B	W	B	W	W	B	B	B	3	3
W	W	W	B	B	B	W	B	B	W	W	2	2
?	?	?	?	?	?	?	?	?	?	?	?	?

110

Fig. 5. A multi-modal trace of ECA rule 110 with eleven light sensors (left) $l_1, ..., l_{11}$ and two touch sensors (right) t_1, t_2 attached to cells 3 and 11. Each row represents the states of the sensors for one time-step. For this prediction task, the final time-step is held out.

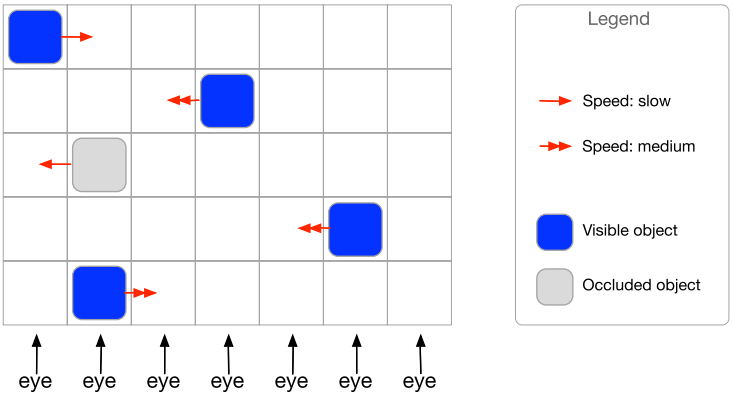


Fig. 6. An occlusion task.

see the objects in the column it is placed in. An object is occluded if there is another object below it in the same column. See Fig. 6.

The system receives a sensory sequence consisting of the positions of the moving objects whenever they are visible. The position of the objects when they are occluded is used as held-out test data to verify the predictions of the model. This is an imputation task.

Results. We generated 20 occlusion tasks by varying the size of the grid, the number of moving objects, their direction and speed. The Apperception Engine achieved 90% accuracy.

5.2. Empirical evaluations

In this section, we test our system to evaluate the truth of the following hypotheses:

1. The five domains of section 5.1 are challenging tasks for existing baselines.
2. Our system handles retrodiction and imputation just as easily as prediction.
3. The features of our system (unity conditions, cost minimisation) are essential to its performance.
4. Our system outperforms state-of-the-art inductive logic programming approaches in the five domains.

We consider each in turn.

5.2.1. The five domains of section 5.1 are challenging tasks for existing baselines

To evaluate whether our domains are indeed sufficiently challenging, we compared our system against four baselines. The first **constant** baseline always predicts the same constant value for every sensor for each time-step. The second **inertia** baseline always predicts that the final hidden time-step equals the penultimate time-step. The third **MLP** baseline is a fully-connected multilayer perceptron (MLP) [88] that looks at a window of up to 10 earlier time-steps to predict the next

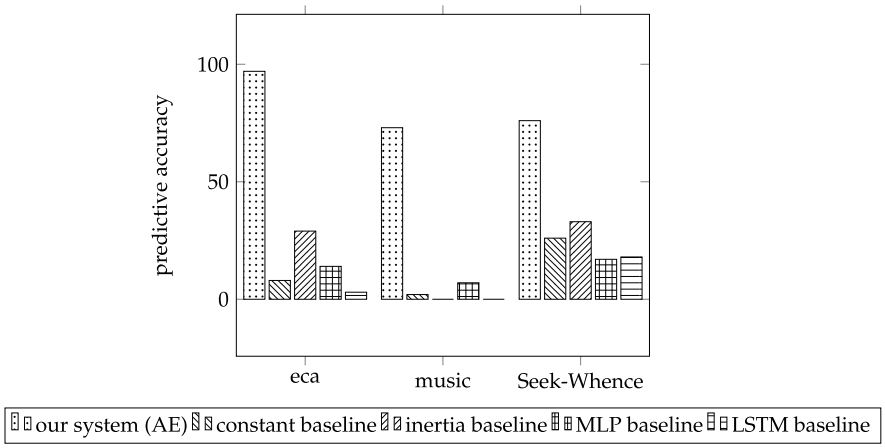


Fig. 7. Comparison with baselines. We display predictive accuracy on the held-out final time-step.

time-step. The fourth **LSTM** baseline is a recurrent neural net based on the long short-term memory (LSTM) architecture [89].

The neural baselines are designed to exploit potential statistical patterns that are indicative of hidden sensor states. In the MLP baseline, we formulate the problem as a multi-class classification problem, where the input consists in a feature representation **x** of relevant context sensors, and a feed-forward network is trained to predict the correct state **y** of a given sensor in question. In the prediction task, the feature representation comprises one-hot[32] representations for the state of every sensor in the previous two time steps before the hidden sensor. The training data consists of the collection of all observed states in an episode (as potential hidden sensors), together with the respective history before. Samples with incomplete history window (at the beginning of the episode) are discarded.

The MLP classifier is a 2-layer feed-forward neural network, which is trained on all training examples derived from the current episode (thus no cross-episode transfer is possible). We restrict the number of hidden neurons to (20, 20) for the two layers, respectively, in order to prevent overfitting given the limited number of training points within an episode. We use a learning rate of 10^{-3} and train the model using the *Adam* optimiser [90] for up to 200 epochs, holding aside 10% of data for early stopping.

Given that the input is a temporal sequence, a recurrent neural network (that was designed to model temporal dynamics) is a natural choice of baseline. But we found that the LSTM performs only slightly better than the MLP on Seek Whence tasks, and worse on the other tasks. The reason for this is that the paucity of data (a single temporal sequence consisting of a small number of time-steps) does not provide enough information for the high capacity LSTM to learn desirable gating behaviour. The simpler and more constrained MLP with fewer weights is able to do slightly better on some of the tasks, yet both neural baselines achieve low accuracy in absolute terms.

Fig. 7 shows the results. Clearly, the tasks are very challenging for all four baseline systems.

5.2.2. Our system handles retrodiction and imputation just as easily as prediction

To evaluate whether our system is just as capable of retrodicting earlier values and imputing missing intermediate values as it is at predicting future values, we ran tests where the unseen hidden sensor values were at the first time step (in the case of retrodiction) or randomly scattered through the time-series (in the case of imputation).[33] We made sure that the number of hidden sensor values was the same for prediction, retrodiction, and imputation.

Fig. 8 shows the results. The results are significantly lower for retrodiction in the ECA tasks, but otherwise comparable. The reason for retrodiction's lower performance on ECA is that for a particular initial configuration there are a significant number (more than 50%) of the ECA rules that wipe out all the information in the current state after the first state transition, and all subsequent states then remain the same. The results for imputation are comparable with the results for prediction. Although the results for rhythm and music are lower, the results on Seek Whence are slightly higher.

[32] A one-hot representation of feature i of n possible features is a vector of length n in which all the elements are 0 except the i'th element.

[33] A deterministic transition dynamic is a function from a set of ground atoms to a set of ground atoms. If this function is not injective, then information is lost as we go through time: there will be a unique next step from the current time-step, but there can be multiple previous steps that transition into the current time-step. Our search procedure uses maximum a posteriori (MAP) estimation: we find a single model with the highest posterior (based on the likelihood – how well it explains the sequence, and on the prior – the program length), and uses that model to predict, retrodict, and impute. A more ambitious Bayesian approach would construct a probability distribution over rival theories, and use a mixture model for retrodiction. But, given the computation complexity of finding a single solution (see Section 4.4), this ambitious approach is – in the short term at least – prohibitively expensive.

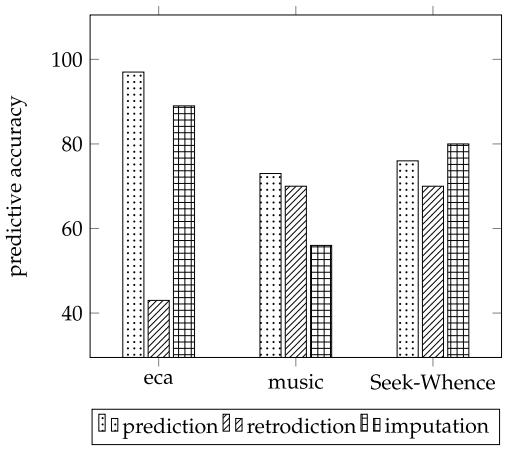


Fig. 8. Comparing prediction with retrodiction and imputation. In retrodiction, we display accuracy on the held-out initial time-step. In imputation, a random subset of atoms are held-out; the held-out atoms are scattered throughout the time-series. In other words, there may be different held-out atoms at different times. The number of held-out atoms in imputation matches the number of held-out atoms in prediction and retrodiction.

Table 5
Ablation experiments. We display predictive accuracy on the final held-out time-step.

	ECA	Rhythm & music	Seek Whence
Full system (AE)	97.3%	73.3%	76.7%
No check that $S \sqsubseteq \tau(\theta)$	5.1%	3.0%	4.6%
No conceptual unity	5.3%	0.0%	6.7%
No spatial unity	95.7%	73.3%	73.3%
No cost minimization	96.7%	56.6%	73.3%

5.2.3. The features of our system are essential to its performance

To verify that the unity conditions are doing useful work, we performed a number of experiments in which the various conditions were removed, and compared the results. We ran four ablation experiments. In the first, we removed the check that the theory's trace covers the input sequence: $S \sqsubseteq \tau(\theta)$ (see Definition 16). In the second, we removed the check on conceptual unity. Removing this condition means that the unary predicates are no longer connected together via exclusion relations $\oplus$, and the binary predicates are no longer constrained by $\exists!$ conditions. (See Definition 13.) In the third ablation test, we removed the check on spatial unity. Removing this condition means allowing objects which are not connected via binary relations. In the fourth ablation test, we removed the cost minimization part of the system. Removing this minimization means that the system will return the first interpretation it finds, irrespective of size.

The results of the ablation experiments are displayed in Table 5. The first ablation test, where we remove the check that the generated sequence of sets of ground atoms respects the original sensory sequence ($S \sqsubseteq \tau(\theta)$), performs very poorly. Of course, if the generated sequence does not cover the given part of the sensory sequence, it is highly unlikely to accurately predict the held-out part of the sensory sequence. This test is just a sanity check that our evaluation scripts are working as intended.

The second ablation test, where we remove the check on conceptual unity, also performs poorly. The reason is that without constraints, there are no incompossible atoms. Recall from Definition 9 that two atoms are incompossible if there is some $\oplus$ constraint or some $\exists!$ constraint that means the two atoms cannot be simultaneously true. But in Definition 9, the frame axiom forces an atom that was true at the previous time-step to also be true at the next time-step unless the old atom is incompossible with some new atom: we add α to H_t if α is in H_{t-1} and there is no atom in H_t that is incompossible with α. But if there are no incompossible atoms, then all previous atoms are always added. Therefore, if there are no $\oplus$ and $\exists!$ constraints, then the set of true atoms monotonically increases over time. This in turn means that state information becomes meaningless, as once something becomes true, it remains always true, and cannot be used to convey information.

When we remove the spatial unity constraint, the results for the rhythm tasks are identical, but the results for the ECA and Seek Whence tasks are lower. The reason why the results are identical for the rhythm tasks is because the background knowledge provided (the r relation on notes, see Section 5.1.3) means that the spatial unity constraint is guaranteed to be satisfied. The reason why the results are lower for ECA tasks is because interpretations that fail to satisfy spatial unity contain disconnected clusters of cells (e.g. cells $\{c_1, ..., c_5\}$ are connected by r in one cluster, while cells $\{c_6, ..., c_{11}\}$ are connected in another cluster, but $\{c_1, ..., c_5\}$ and $\{c_6, ..., c_{11}\}$ are disconnected). Interpretations with disconnected clusters tend to generalize poorly and hence predict with less accuracy. The reason why the results are only slightly lower for the

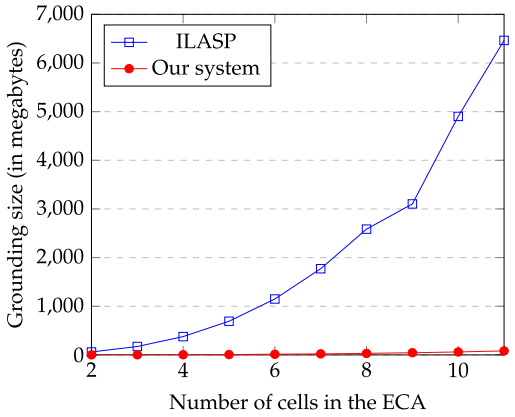


Fig. 9. Comparing our system and ILASP w.r.t. grounding size.

Seek Whence tasks is because the lowest cost unified interpretation for most of these tasks also happens to satisfy spatial unity.

The results for the fourth ablation test, where we remove the cost minimization, are broadly comparable with the full system in ECA and Seek Whence, but are markedly worse in the rhythm / music tasks. But even if the results were comparable in all tasks, there are independent reasons to want to minimize the size of the interpretation. Shorter interpretations are more human-readable, and transfer better to new situations (since they tend to be more general, as they have fewer atoms in the bodies of the rules).

5.2.4. Our system outperforms state-of-the-art inductive logic programming approaches in the five domains

In order to assess the efficiency of our system, we compared it to ILASP[34] [96–99], a state-of-the-art Inductive Logic Programming algorithm.[35] Unlike traditional ILP systems that learn definite logic programs, ILASP learns *answer set programs*.[36] ILASP is a powerful and general framework for learning answer set programs; it is able to learn choice rules, constraints, and even preferences over answer sets [97].

ILASP is able to solve some simple apperception tasks. For example, ILASP is able to solve the task in Example 6. But for the ECA tasks, the music and rhythm tasks, and the Seek Whence tasks, the ASP programs generated by ILASP were not solvable because they required too much memory.

In order to understand the memory requirements of ILASP on these tasks, and to compare our system with ILASP in a fair like-for-like manner, we looked at the size of the grounded ASP programs. Recall that both our system and ILASP generate ASP programs that are then grounded into propositional clauses that are then passed to a SAT solver. The grounding size determines the memory usage and is strongly correlated with solution time.

We took a representative ECA rule, Rule 245, and looked at the grounding size as the number of cells increased from 2 to 11. We used the same template for both ILASP and the Apperception Engine. The results are in Fig. 9.

As we increase the number of cells, the grounding size of the ILASP program grows much faster than the corresponding Apperception Engine program. The reason for this marked difference is the different ways the two approaches represent rules. In our system, rules are interpreted by an interpreter that operates on reified representations of rules. In ILASP, by contrast, rules are *compiled* into ASP rules. This means, if there are $|U_\phi|$ unground atoms and there are at most N_B atoms in the body of a rule, then ILASP will generate $|U_\phi|^{N_B+1}$ different clauses. When it comes to grounding, if there are $|\Sigma_\phi|$ substitutions and t time-steps, then ILASP will generate at most $|U_\phi|^{N_B+1} \cdot |\Sigma_\phi| \cdot t$ ground instances of the generated clauses. Each ground instance will contain $N_B + 1$ atoms, so there are $(N_B + 1) \cdot |U_\phi|^{N_B+1} \cdot |\Sigma_\phi| \cdot t$ ground atoms in total.

Compare this with our system. Here, we do not represent every possible rule explicitly as a separate clause. Rather, we represent the possible atoms in the body of a rule by an ASP choice rule:

```
0 { rule_body(R, VA) : is_unground_atom(VA) } k_max_body :- is_rule(R).
```

[34] We compared against ILASP rather than Metagol (another state-of-the-art inductive logic programming system [91,92]) because (i) ILASP is comparable in performance (it achieved slightly better results than Metagol in the Inductive General Game Playing task suite [93], getting 40% correct as opposed to Metagol's 36%), and (ii) since ILASP also uses ASP we can compare the grounding size of our program with ILASP and get a fair apples-for-apples comparison. We used ILASP rather than HEXMIL (the ASP implementation of Metagol [94]) because of scaling problems with HEXMIL [95]. Our decision to use ILASP rather than Metagol for these tests was based on a number of discussions with Andrew Cropper (the developer of Metagol) and Mark Law (the developer of ILASP). We are very grateful to both for their advice on this.

[35] Strictly speaking, ILASP is a family of algorithms, rather than a single algorithm. We used ILASP2 [97] in this evaluation.

[36] Answer set programming under the stable model semantics is distinguished from traditional logic programming in that it is purely declarative and each program has multiple solutions (known as answer sets). Because of its non-monotonicity, ASP is well suited for knowledge representation and commonsense reasoning [100,101].

If there are $N_{\rightarrow}$ static rules and $N_{\Rightarrow}$ causal rules, then this choice rule only generates $N_{\rightarrow} + N_{\Rightarrow}$ ground clauses, each containing $|U_\phi|$ atoms.

The most expensive clauses in our encoding are analysed in Table 3. Recall from Section 4.4 that the total number of atoms in the ground clauses is approximately $5 \cdot |\Sigma_\phi| \cdot (N_{\rightarrow} + N_{\Rightarrow}) \cdot |U_\phi| \cdot t$.

To compare this with ILASP, let us set $N_B = 4$ (which is representative). Then ILASP generates ground clauses with $5 \cdot |U_\phi|^5 \cdot |\Sigma_\phi| \cdot t$ ground atoms while our system generates clauses with $5 \cdot |\Sigma_\phi| \cdot (N_{\rightarrow} + N_{\Rightarrow}) \cdot |U_\phi| \cdot t$ ground atoms. The reason, then, why our system has such lower grounding sizes than ILASP is because $(N_{\rightarrow} + N_{\Rightarrow}) << |U_\phi|^4$. Intuitively, the key difference is that ILASP considers *every possible* subset of the hypothesis space, while our system (by restricting to at most $N_{\rightarrow} + N_{\Rightarrow}$ rules) only considers *subsets of length at most* $N_{\rightarrow} + N_{\Rightarrow}$.

6. Noisy apperception

So far, we have assumed that our sensor readings are entirely noise-free: some of the sensory readings may be missing, but none of the readings are inaccurate.

If we give the Apperception Engine a sensory sequence with mislabelled data, it will struggle to provide a theoretical explanation of the mislabelled input. Consider, for example, $S_{1:20}$:

$$\begin{array}{lllll} S_1 = \{p(a)\} & S_2 = \{p(a)\} & S_3 = \{\mathbf{q}(\mathbf{a})\} & S_4 = \{p(a)\} & S_5 = \{p(a)\} \\ S_6 = \{p(a)\} & S_7 = \{p(a)\} & S_8 = \{p(a)\} & S_9 = \{p(a)\} & S_{10} = \{p(a)\} \\ S_{11} = \{p(a)\} & S_{12} = \{p(a)\} & S_{13} = \{p(a)\} & S_{14} = \{p(a)\} & S_{15} = \{p(a)\} \\ S_{16} = \{p(a)\} & S_{17} = \{p(a)\} & S_{18} = \{p(a)\} & S_{19} = \{p(a)\} & S_{20} = \{p(a)\} \end{array}$$

Here, $S_3 = \{\mathbf{q}(\mathbf{a})\}$ is an outlier in the otherwise tediously predictable sequence.

If we give sequences such as this to the Apperception Engine, it attempts to make sense of *all* the input, including the anomalies. In this case, it finds the following baroque explanation:

$$I = \left\{ \begin{array}{l} p(a) \\ c_1(a) \end{array} \right\} \quad R = \left\{ \begin{array}{l} q(X) \rightarrow c_3(X) \\ c_3(X) \Rightarrow p(X) \\ c_1(X) \Rightarrow c_2(X) \\ c_2(X) \Rightarrow q(X) \end{array} \right\} \quad C' = \left\{ \begin{array}{l} \forall X{:}s,\ p(X) \oplus q(X) \\ \forall X{:}s,\ c_1(X) \oplus c_2(X) \oplus c_3(X) \end{array} \right\}$$

Here, the Apperception Engine has introduced three new invented predicates c_1, c_2, c_3 in order to count how many p's it has seen, so that it knows when to switch to q. If we move the anomalous entry $q(a)$ later in the sequence, or add further anomalies, the engine is forced to construct increasingly complex theories. This is clearly unsatisfactory.

In order to handle noisy mislabelled data, we shall relax our insistence that the sequence $S_{1:T}$ is *entirely* covered by the trace of the theory θ. Instead of insisting that $S \sqsubseteq \tau(\theta)$, we shall minimise the number of discrepancies between each S_i and $\tau(\theta)_i$, for $i = 1..T$.

We want to find the most probable theory θ given our noisy input sequence $S_{1:T}$:

$$\arg\max_\theta \ p\left(\theta \mid S_{1:T}\right) \tag{1}$$

By Bayes' rule, this is equivalent to

$$\arg\max_\theta \ \frac{p(\theta) \cdot p\left(S_{1:T} \mid \theta\right)}{p(S_{1:T})} \tag{2}$$

Since the denominator does not depend on θ, this is equivalent to:

$$\arg\max_\theta \ p(\theta) \cdot p\left(S_{1:T} \mid \theta\right) \tag{3}$$

Since the probability of the state S_i is conditionally independent of the previous state S_{i-1} given θ (this is the assumption behind the Hidden Markov Model), the above is equivalent to:

$$\arg\max_\theta \ p(\theta) \cdot \prod_{i=1}^{T} p\left(S_i \mid \theta\right) \tag{4}$$

Now each S_i depends only on $\tau(\theta)_i$, the trace of θ at time step i. Thus we can rewrite to:

$$\arg\max_\theta \ p(\theta) \cdot \prod_{i=1}^{T} p\left(S_i \mid \tau(\theta)_i\right) \tag{5}$$

Since we assume a description length prior over theories, $p(\theta) \propto 2^{-len(\theta)}$. Let the probability of S_i given $\tau(\theta)_i$ be $p\left(S_i \mid \tau(\theta)_i\right) = 2^{-|S_i - \tau(\theta)_i|}$. Then we can rewrite to:

$$\arg\max_{\theta} 2^{-len(\theta)} \cdot \prod_{i=1}^{T} 2^{-|S_i - \tau(\theta)_i|} \tag{6}$$

Thus, we define the $cost_{noise}$ of the theory to be:

$$cost_{noise} = len(\theta) + \sum_{i=1}^{T} |S_i - \tau(\theta)_i| \tag{7}$$

and search for the theory with lowest cost.

Example 12. Consider, for example, the following sequence $S_{1:10}$:

$$\begin{array}{lllll} S_1 = \{\} & S_2 = \{off(a), on(b)\} & S_3 = \{on(a), off(b)\} & S_4 = \{on(a), on(b)\} & S_5 = \{on(b)\} \\ S_6 = \{on(a), off(b)\} & S_7 = \{on(a), on(b)\} & S_8 = \{off(a), on(b)\} & S_9 = \{on(a)\} & S_{10} = \{\} \end{array}$$

Because the sequence is so short, the lowest $cost_{noise}$ theory is:

$$I = \{\ \} \quad R = \{\ \} \quad C' = \{\ \forall X{:}s,\ on(X) \oplus off(X)\ \}$$

This degenerate empty theory has cost 14 (the number of atoms in S) which is shorter than any "proper" explanation that captures the regularities. But as the sequence gets longer, the advantage of a "proper" explanation over a degenerate solution becomes more and more apparent. Consider, for example, the following extension $S'_{1:30}$:

$$\begin{array}{lllll} S'_1 = \{\} & S'_2 = \{off(a), on(b)\} & S'_3 = \{on(a), off(b)\} & S'_4 = \{on(a), on(b)\} & S'_5 = \{on(b)\} \\ S'_6 = \{on(a), off(b)\} & S'_7 = \{on(a), on(b)\} & S'_8 = \{off(a), on(b)\} & S'_9 = \{on(a)\} & S'_{10} = \{\} \\ S'_{11} = \{off(a), on(b)\} & S'_{12} = \{on(a), off(b)\} & S'_{13} = \{on(a), on(b)\} & S'_{14} = \{off(a), on(b)\} & S'_{15} = \{on(a), off(b)\} \\ S'_{16} = \{on(a), on(b)\} & S'_{17} = \{off(a), on(b)\} & S'_{18} = \{on(a), off(b)\} & S'_{19} = \{on(a), on(b)\} & S'_{20} = \{off(a), on(b)\} \\ S'_{21} = \{on(a), off(b)\} & S'_{22} = \{on(a), on(b)\} & S'_{23} = \{off(a), on(b)\} & S'_{24} = \{on(a), off(b)\} & S'_{25} = \{on(a), on(b)\} \\ S'_{26} = \{off(a), on(b)\} & S'_{27} = \{on(a), off(b)\} & S'_{28} = \{on(a), on(b)\} & S'_{29} = \{off(a), on(b)\} & S'_{30} = \{\} \end{array}$$

Now the lowest $cost_{noise}$ theory is one that finds the underlying regularity:

$$I = \left\{ \begin{array}{l} on(a) \\ p_1(a) \\ p_2(b) \end{array} \right\} \quad R = \left\{ \begin{array}{l} off(X) \rightarrow p_3(X) \\ p_2(X) \rightarrow on(X) \\ p_1(X) \supset off(X) \\ p_3(X) \supset p_2(X) \\ p_2(X) \supset p_1(X) \end{array} \right\} \quad C' = \left\{ \begin{array}{l} \forall X{:}s,\ on(X) \oplus off(X) \\ \forall X{:}s,\ p_1(X) \oplus p_2(X) \oplus p_3(X) \end{array} \right\}$$

We can see, then, that the noise-robust version of the Apperception Engine is somewhat less data-efficient than the noise-intolerant version described earlier. ◁

6.1. Experiments

We used the following sequences to compare the noise-intolerant Apperception Engine with the noise-robust version:

a,b,a,b,a,b,a,b,a,b,a,b, ...	**a,a,b,a,a,b,a,a,b,a,a,b, ...**
a,a,b,b,a,a,b,b,a,a,b,b, ...	**a,a,a,b,a,a,a,b,a,a,a,b, ...**
a,b,b,a,a,b,b,a,a,b,b,a, ...	**a,b,c,a,b,c,a,b,c,a,b,c, ...**
a,b,c,b,a,a,b,c,b,a,a,b,c,b,a, ...	**a,b,a,c,a,b,a,c,a,b,a,c, ...**
a,b,c,c,a,b,c,c,a,b,c,c, ...	**a,a,b,b,c,c,a,a,b,b,c,c, ...**

We chose these particular sequences because they are simple, noise-free, and the Apperception Engine is able to solve them in a reasonably short time.

We performed two groups of experiments. In the first, we evaluated how much longer the sequence needs to be for the noise-robust version to capture the underlying regularity, in comparison with the noise-intolerant version which is more data-efficient. Fig. 10 shows the results. We plot mean percentage accuracy (over the ten sequences) against the length of the sequence that is provided to the Apperception Engine. Note that the noise-intolerant version only needs sequences of length 10 to achieve 100% accuracy, while the noise-tolerant version needs sequences of length 30.[37]

In the second experiment, we evaluate how much better the noise-robust version of the Apperception Engine is at handling mislabelled data. We take the same ten sequences above, extended to length 100, and consider various perturbations

[37] Note also that the noise-tolerant version finds a solution significantly (17x) faster than the noise-intolerant version.

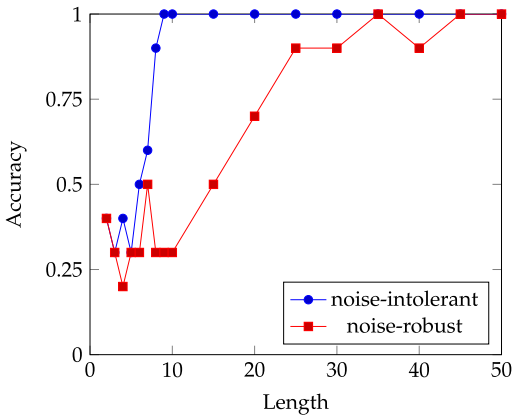


Fig. 10. Comparing the data-efficiency of the noise-robust version of the Apperception Engine with the noise-intolerant version. We plot mean percentage accuracy against length of the sequence. The noise-intolerant version achieves 100% accuracy when the sequence is length 10 or over, while the noise-robust version only achieves this level of accuracy when the length is over 30.

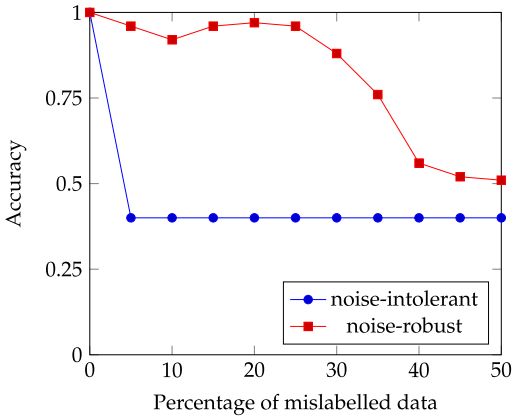


Fig. 11. Comparing the accuracy of the noise-robust version of the Apperception Engine with the noise-intolerant version. We plot mean percentage accuracy against the number of mislabellings. The noise-intolerant version deteriorates to random as soon as any noise is introduced, while the noise-robust version is able to maintain reasonable (88%) accuracy with up to 30% of the sequence mislabelled.

of the sequence where we randomly mislabel a certain number of entries. Fig. 11 shows the results. We plot mean percentage accuracy (over the ten sequences) against the percentage of mislabellings. Note that the noise-intolerant version deteriorates to random as soon as any noise is introduced, while the noise-robust version is able to maintain reasonable accuracy with up to 30% of the sequence mislabelled.

7. Related work

In this section, we describe particular systems that are related to our approach. For a general overview of the space of different approaches, see Section 1.1.

7.1. "Theory learning as stochastic search in a language of thought"

Ullman et al. [67,68] describe a system for learning first-order rules from symbolic data. Recasting their approach into our notation, their system is given as input a set S of ground atoms,[38] and it searches for a set of static rules R and a set I of atoms such that $R, I \models S$.

Of course, the task as just formulated admits of entirely trivial solutions: for example, let $I = S$ and $R = \{\}$. Ullman et al. rule out such trivial solutions by adding two restrictions. First, they distinguish between two disjoint sets of predicates: the *surface* predicates are the predicates that appear in the input S, while the *core* predicates are the latent predicates. Only core predicates are allowed to appear in the initial conditions I. This distinction rules out the trivial solution above, but

[38] Compare with our system, which is given a sequence $(S_1, ..., S_T)$ of sets of ground atoms.

there are other degenerate solutions: for each surface predicate p, add a new core predicate p_c. If $p(k_1, ..., k_n)$ is in S, add $p_c(k_1, ..., k_n)$ to I. Also, add the rule $p(X_1, ..., X_n) \leftarrow p_c(X_1, .., X_n)$ to R. Clearly, $R, I \models S$ but this solution is unilluminating, to say the least. To prevent such degenerate solutions, the second restriction that Ullman et al. add is to prefer *shorter* rule-sets R and smaller sets I of initial atoms. The idea is that if S contains structural regularities, their system will find an R and I that are much simpler than the degenerate solution above.

Consider, for example, the various surface relations in a family tree: John is the father of William; William is the husband of Anne; Anne is the mother of Judith; John is the grandfather of Judith. All the various surface relations (father, mother, husband, grandfather...) can be explained by a small number of core relations: $parent(X, Y)$, $spouse(X, Y)$, $male(X)$, and $female(X)$. Now the surface facts $S = \{father(john, william), ...\}$ can be explained by a small number of facts involving core predicates $I = \{parent(john, william), male(john), ...\}$ together with rules such as:

$$father(X, Y) \leftarrow parent(X, Y), male(X)$$

At the computational level, then, the task that Ullman et al. set out to solve is: given a set S of ground atoms featuring surface predicates, find the smallest set I of ground atoms featuring only core predicates, and the smallest set R of static rules, such that $R, I \models S$. Recasting this task in the language of probability, they wish to find:

$$\arg\max_{R,I} p(R, I \mid S)$$

Using Bayes' rule this can be recast as:

$$\begin{aligned}\arg\max_{R,I} p(R, I \mid S) &= \arg\max_{R,I} \frac{p(S \mid R, I)p(R, I)}{p(S)} \\ &= \arg\max_{R,I} p(S \mid R, I)p(R, I) \\ &= \arg\max_{R,I} p(S \mid R, I)p(R)p(I \mid R)\end{aligned}$$

Here, the likelihood $p(S \mid R, I)$ is the proportion of S that is entailed by R and I, the prior $p(R)$ is the size of the rules, and $p(I \mid R)$ is the size of I.

At the algorithmic level, Ullman et al. apply Markov Chain Monte Carlo (MCMC). MCMC is a stochastic search procedure. When it is currently considering search element x, it generates a candidate next element x' by randomly perturbing x. Then it compares the scores of x and x'. If x' is better, it switches attention to focus on x'. Otherwise, if x' is worse than x, there is still a non-zero probability of switching (to avoid local minima), but the probability is lower when x' is significantly worse than the current search element x.

In their algorithm, MCMC is applied at two levels. At the first level, a set R of rules is perturbed into R' by adding or removing atoms from clauses, or by switching one predicate for another predicate with the same arity. At the second level, I is perturbed into I' by changing the extension of the core predicates.

Given that the search space of sets of rules is so enormous, and that MCMC is a stochastic search procedure that only operates locally, the algorithm needs additional guidance to find solutions. In their case, they provide a *template*, a set of meta-rules that constrain the types of rules that are generated in the outermost MCMC loop. A meta-rule is a higher-order clause in which the predicates are themselves variables. For example, in the following meta-rule for transitivity, P is a variable ranging over two-place predicates:

$$P(X, Y) \leftarrow P(X, Z), P(Z, Y)$$

Meta-rules are a key component in many logic program synthesis systems [91,102,103,96,99].

Ullman et al. tested their system in a number of domains including taxonomy hierarchies, simplified magnetic theories, kinship relations, and psychological explanations of action. In each domain, their system is able to learn human-interpretable theories from small amounts of data.

At a high level, Ullman et al.'s system has much in common with the Apperception Engine. They are both systems for generating interpretable explanations from small quantities of symbolic data. While the Apperception Engine generates a (ϕ, I, R, C) tuple from a sequence $(S_1, ..., S_T)$, their system generates an (I, R) pair from a single set S of atoms. But there are a number of significant differences. First, and most importantly, our system learns causal dynamics from time series, while their system only learns static rules. Second, our system posits *latent objects* as well as latent predicates, while their system only posits latent predicates. The ability to imagine unobserved objects, with unobserved attributes that explain the observed attributes of observed objects, is a key feature of the Apperception Engine. Third, their system requires *hand-engineered templates* in order to find a theory that explains the input. This reliance on hand-engineered templates restricts the domain of application of their technique: in a domain in which they do not know, in advance, the structure of the rules they want to learn, their system will not be applicable. Fourth, a unified interpretation $\theta = (\phi, I, R, C)$ in our system includes a set C of *constraints*. These constraints play a critical role in our system: they are both regulative (ruling out certain incompossible combinations of atoms) and constitutive (the constraints determine the incompossible relation that

in turn grounds the frame axiom). There is no equivalent of our constraints C in their system. A fifth key difference is that our system has to produce a theory that, as well as explaining the sensory sequence, also has to satisfy the *unity conditions*: spatial unity, conceptual unity, static unity, and temporal unity. There is no analog of our unity conditions in Ullman et al.'s system.

At the algorithmic level, the systems are very different. While we use a form of meta-interpretive learning (see Section 4.2), they use MCMC. Our system compiles an apperception problem into the task of finding an answer set to an ASP program that minimises the program cost. The ASP problem is given to an ASP solver, that is guaranteed to find the *global minimum*. MCMC, by contrast, is a stochastic procedure that operates *locally* (moving from one single point in program space to another), and is not guaranteed to (in fact, in practice, it rarely does) find a global minimum.

7.2. "Learning from interpretation transition"

Inoue, Ribeiro, and Sakama [12] describe a system (**LFIT**) for learning logic programs from sequences of sets of ground atoms. Since their task definition is broadly similar to ours, we focus on specific differences. In our formulation of the apperception task, we must construct a (ϕ, I, R, C) tuple from a sequence $(S_1, ..., S_T)$ of sets of ground atoms. In their task formulation, they learn a set of causal rules from a set $\{(A_i, B_i)\}_{i=1}^{N}$ of pairs of sets of ground atoms.

In some respects, their task formulation is more general than ours. First, their input $\{(A_i, B_i)\}_{i=1}^{N}$ can represent transitions from *multiple* trajectories, rather than just a single trajectory, and corresponds to a generalized apperception task (see Definition 19). Second, they learn *normal* logic programs, allowing negation as failure in the body of a rule, while our system only learns definite clauses.

But there are a number of other ways in which our task formulation is significantly more general than LFIT. First, our system posits *latent information* to explain the observed sequence, while LFIT does not construct any latent information. Their system searches for a program P that generates *exactly* the output state. In our approach, by contrast, we search for a program whose trace *covers* the output sequence, but does not need to be identical to it. The trace of a unified interpretation typically contains much extra information that is not part of the original input sequence, but that is used to explain the input information.

Second, our system abduces a set of *initial conditions* as well as a set of rules, while LFIT does not construct initial conditions. Because of this, our system is able to predict the future, retrodict the past, and impute missing intermediate values. LFIT, by contrast, can only be used to predict future values.

Third, our system generates a set of *constraints* as well as rules. The constraints perform double duty: on the one hand, they restrict the sets of compossible atoms that can appear in traces; on the other hand, they generate the incompossibility relation that grounds the frame axiom. Note that there is no frame axiom in LFIT.

In [12], Inoue et al. use a bottom-up synthesis method to learn rules. Given a state transition (A, B) in E, they construct a normal ground rule for each $\beta \in B$:

$$\bigwedge_{\alpha \in A} \alpha \wedge \bigwedge_{\alpha \in \mathcal{G}-A} not\ \alpha \ni \beta$$

Then, they use resolution to generalize the individual ground rules. It is important to note that this strategy is quite conservative in the generalizations it performs, since it only produces a more general rule if it turns out to be a resolvent of a pair of previous rules. While the Apperception Engine searches for the shortest (and hence most general) rules, LFIT searches for the most specific generalization. In more recent work [104], LFIT has been changed to perform top-down specialization, rather than bottom-up generalization. With this change, LFIT is guaranteed to find the shortest set of rules that explain the transitions.

LFIT was tested on Boolean networks and on Elementary Cellular Automata. It is instructive to compare our system with LFIT on the ECA tasks. When LFIT is applied to the ECA task, it is provided with the one-dimensional spatial relation between the cells as background knowledge. In our approach, by contrast, we do *not* hand-code the spatial relation, but rather let the Apperception Engine generate the spatial relation itself as part of the initial conditions. (See Section 5.1.2.) It is precisely because our system is able to posit latent information to explain the surface features that it is able to generate the spatial relation itself, rather than having to be given it.

In some situations, positing latent information allows us to constructer a simpler theory. In other situations, however, positing latent information is absolutely essential to making sense of the sequence. There are many apperception tasks for which every interpretation that makes sense of the sequence *must* include latent information: consider, for example, learning the dynamics of the ECA without spatial information (Section 5.1.2), the *Seek Whence* sequences (Section 5.1.4), or the binding tasks (Section 5.1.6). Even the following simple example shows the unavoidable need to posit latent information:

Example 13. Consider the task (S, ϕ, C) where $S = (S_1, S_2, S_3, ..., S_n)$, $S_1 = S_2 = \{p(a)\}$, and $S_3 = S_4 = ... = S_n = \{q(a)\}$. Here, ϕ contains one type t, one object a of type t, two unary predicates p and q, and one constraint $\forall X : t, p(X) \oplus q(X)$. Since p and q are incompatible, the transition from p to q from state S_2 to S_3 must be explained by a causal rule $\phi \ni q(X)$, where ϕ is a set of atoms. Now ϕ cannot be empty, or the rule would be unsafe, ϕ cannot be $\{p(X)\}$, or else $q(a)$ would be derivable at the second time-step, contradicting $S_2 = \{p(a)\}$. Similarly, ϕ cannot be $\{q(X)\}$, or else $q(a)$ would

not be derivable at time-step 3. Hence ϕ must contain an atom featuring a predicate distinct from p or q. Hence, every interpretation that makes sense of S must invoke latent information.

One of the theories found by the Apperception Engine for this is $\theta = (\phi, I, R, C)$, where:

$$I = \left\{ \begin{array}{l} p(a) \\ r(a) \end{array} \right\} \quad R = \left\{ \begin{array}{l} r(X) \supset s(X) \\ p(X) \wedge s(X) \supset q(X) \end{array} \right\} \quad C = \left\{ \begin{array}{l} \forall X{:}t,\ p(X) \oplus q(X) \\ \forall X{:}t,\ r(X) \oplus s(X) \end{array} \right\}$$

Here, there are two latent predicates, r and s, that are used as counters, so the system can distinguish between the two occurrences of $p(a)$ in S_1 and S_2. Thus for some sequences, the positing of latent predicates, and the abduction of initial conditions for the latent atoms, is *unavoidable*.[39] ◁

LFIT has been extended in a number of ways, to increase the range of real-world problems that it can tackle. In [106], LFIT was extended to learn probabilistic models. In [105], the system was extended from the Markov(1) assumption (where the new state depends only on the current state) to the more general Markov(k) setting (where the new state depends on the last k states). In [107], LFIT was generalised so that as well as working with deterministic models (where all state transitions happen simultaneously), it also can work with other semantics (where only a subset of the transitions may happen at each time-step). In [108], LFIT was extended to work directly with continuous sensor data, rather than assuming the continuous sensor data has first been discretised by some other process. In [109,110], LFIT was reimplemented in a feed-forward neural network, so as to robustly handle noisy and continuous data.

7.3. "Unsupervised learning by program synthesis"

Ellis et al. [22] use program synthesis to solve an unsupervised learning problem. Given an unlabelled dataset $\{x_i\}_{i=1}^N$, they find a program f and a set of inputs $\{I_i\}_{i=1}^N$ such that $f(I_i)$ is close to x_i for each $i = 1..N$. More precisely, they use Bayesian inference to find the f and $\{I_i\}_{i=1}^N$ that minimizes the total description length (negative log probability) of the program, the initial conditions, and the data-reconstruction error:

$$-\log P_f(f) + \sum_{i=1}^{N} \left(-\log P_{x|z}(x_i \mid f(I_i)) - \log P_I(I_i) \right)$$

where $P_f(f)$ is a description length prior over programs, $P_I(I_i)$ is a description length prior over initial conditions, and $P_{x|z}(\cdot \mid z_i)$ is a noise model. This system was designed from the outset to be robust to noise, using Bayesian inference to calculate the desired tradeoff between the program length, the initial conditions length, and the data-reconstruction error cost. They tested this system in two domains: reproducing two dimensional pictures, and learning morphological rules for English verbs.

This system is similar to ours in that it produces interpretable programs from a small number of data samples. Like ours, their program length prior acts as an inductive bias that prefers general solutions over special-case memorized solutions. Like ours, as well as constructing a program, they also learn initial conditions that combine with the program to produce the desired results.[40] At a high level, their algorithm is also similar: they generate a Sketch program [111] from the dataset $\{x_i\}_{i=1}^N$ of examples, and use a SMT solver to fill in the holes. They then extract a readable program from the SMT solution, which they then apply to new instances, exhibiting strong generalization.

As well as the high level architectural similarities, there are a number of important differences. First, their goal was to generate an object $f(I_i)$ that matches as closely as possible to the input object x_i. Our goal is more general: we seek to generate a sequence $\tau(\theta)$ that *covers* the input sequence. The covering relation is much more general, as S_i only has to be a *subset* of $(\tau(\theta))_i$, not identical to it. This allows the addition of latent information to the trace of the theory. A second key difference is that we focus on generating sequences, not individual objects. Our system is designed for making sense (unsupervisedly) of time series, sequences of states, not of reconstructing individual objects. A third key difference is that we use a single domain-independent language, Datalog$^\exists$, for all domains, while Ellis et al. use a different domain-specific imperative language for each domain they consider. A fourth key difference is that we use a declarative language, rather than an imperative language. An individual rule or constraint has a truth-conditional interpretation, and can be interpreted as a *belief* of the synthesising agent. An individual line of an imperative procedure, by contrast, cannot be interpreted as a belief. A fifth major difference is that we synthesise *constraints* as well as rules. Constraints are the "special sauce" of our system: exclusive disjunctions combine predicates into groups, enforce that each state is fully determinate, and ground the incompossibility relation that underlies the frame axiom.

[39] It is unavoidable unless we relax the Markov assumption. In [105], examples like this can be handled by using a rule that tests whether p holds at the last two time-steps: $p(T-2, X) \wedge p(T-1, X) \rightarrow q(T, X)$.

[40] In fact, they learn a different set of initial conditions I_i for each data point x_i. This corresponds to the generalized apperception task of Definition 19.

7.4. "Learning symbolic models of stochastic domains"

Pasula et al. [10] describe a system for learning a state transition model from data. The model learns a probability distribution $p(s' \mid s, a)$ where s is the previous state, a is the action that was performed and s' is the next state.

Each state is represented as a set of ground atoms, just like in our system. They assume *complete observability*: they assume they are given the value of every sensor and the task is just to predict the next values of the sensors.

They represent a state transition model by a set of "dynamic rules": these are first-order clauses determining the future state given a current state and an action. These dynamic rules are very close to the causal rules in Datalog$^{\ni}$. Unlike in our system, their rules have a probability outcome for each possible head. Note their system does not include static rules or constraints.

In their semantics, they assume that *exactly one dynamic rule fires every time-step*. This is a very strong assumption. But it makes it easier to learn rules with probabilistic outcomes.

They learn state transitions for the noisy gripper domain (where a robot hand is stacking bricks, and sometimes fails to pick up what it attempts to pick up) and a logistics problem (involving trucks transporting objects from one location to another). Impressively, they are able to learn probabilistic rules in noisy settings. They also verify the usefulness of their learned models by passing them to a planner (a sparse sampling MDP planner), and show, reassuringly, that the agent achieves more reward with a more accurate model.

At a strategic level, their system is similar in approach to ours. First, they learn first-order rules, not merely propositional ones. In fact, they show in ablation studies that learning propositional rules generalise significantly less well, as you would expect. Second, they use an inductive bias against constants (p. 14), just as we do: "learning action models which are restricted to be free of constants provides a useful bias that can improve generalisation when training with small data sets". Third, their system is able to construct new invented predicates.

But there are also a number of differences. In our system, many rules can fire simultaneously. But in theirs, only one rule can fire in any state. Because of this assumption, they cannot model e.g. a cellular automaton, where each cell has its own individual update rule firing simultaneously. Another limiting assumption is that they assume they have complete observability of all sensory predicates. This means they would not be able to solve e.g. occlusion tasks.

7.5. "Nonmonotonic abductive inductive learning"

Ray [11] described a system, **XHAIL**, for jointly learning to abduce ground atoms and induce first-order rules. XHAIL learns normal logic programs that can include negation as failure in the body of a rule.

XHAIL is similar to the Apperception Engine in that as well as inducing general first-order rules, it also constructs a set of initial ground atoms. This enables it to model latent (unobserved) information, which is a very powerful and useful feature. At the implementation level, it uses a similar strategy in that solutions are found by iterative deepening over a series of increasingly complex ASP programs. The simplified event calculus [54] is represented explicitly as background knowledge.

But there are also a number of key differences. First, it does not model constraints. This means it is not able to represent the incompossibility relation between ground atoms. Also, XHAIL does not try to satisfy our other unity conditions, such as spatial and conceptual unity. Second, the induced rules are *compiled* in XHAIL, rather than being interpreted (as in our system). Representing each candidate induced rule explicitly as a separate ASP rule means that the number of ASP rules considered grows exponentially with the size of the rule body.[41] Third, XHAIL needs to be provided with a set of mode declarations to limit the search space of possible induced rules. These mode declarations constitute a significant piece of background knowledge. Now of course there is nothing wrong with allowing an ILP system to take advantage of background knowledge to aid the search. But when an ILP system *relies* on this hand-engineered knowledge, then it restricts the range of applicability to domains in which human engineers can anticipate in advance the form of the rules they want the system to learn.[42]

7.6. The Game Description Language and Datalog$^{\ni}$

Our language Datalog$^{\ni}$ is an extension of Datalog that incorporates, as well as the standard static rules of Datalog, both causal rules (Definition 7) and constraints (Definition 8). The semantics of Datalog$^{\ni}$ are defined according to Definition 9. Unlike standard Datalog, the atoms and rules of Datalog$^{\ni}$ are strongly typed (see Definitions 4, 5, and 7).

At a high level, Datalog$^{\ni}$ is related to the Game Description Language (GDL) [113]. The GDL is an extension of Datalog that was designed to express deterministic multi-agent discrete Markov decision processes. The GDL includes (stratified) negation by failure, as well as some (restricted) use of function symbols, but these extensions were carefully designed to preserve the key Datalog property that a program has a unique subset-minimal Herbrand model. The GDL includes special keywords, including `init` for specifying initial conditions (equivalent to the initial conditions I in a (ϕ, I, R, C) theory), and

[41] It shares the same implementation strategy as ASPAL [112] and ILASP [96]. See Section 5.2.4 for discussion of the grounding problem associated with this family of approaches. The discussion is specifically focused on ILASP, but we believe the same issue affects ASPAL and XHAIL *mutatis mutandem*.

[42] See Appendix C of [62] for a discussion of the use of mode declarations as a language bias in ILP systems.

`next` for specifying state transitions (equivalent to our causal rules). The *inductive general game playing* (IGGP) task [114,93] involves learning the rules of a game from observing traces of play.

An IGGP task is broadly similar to an apperception task in that both involve inducing initial conditions and rules from traces. But there are many key differences. One major feature of Datalog$^{\ni}$ is the use of *constraints* to generate incompossible sets of ground atoms. These exclusion constraints are needed to generate the incompossibility relation which in turn is needed to restrict the scope of the frame axiom (see Definition 9).

The main difference between Datalog$^{\ni}$ and the GDL is that the former includes exclusion constraints. The exclusion constraints play two essential roles. First, they enable the theory as a whole to satisfy the condition of conceptual unity. Second, they provide constraints, via the condition of static unity, on the generated trace: since the constraints must always be satisfied, this restricts the rules that can be constructed. Satisfying these constraints means *filling in missing information*. This is why a unified interpretation is able to make sense of incomplete traces where some of the sensory data is missing.

7.7. Related application areas

We briefly outline four related research areas where the Apperception Engine can be applied.

One related area of research is learning action theories [115–119]. Here, the aim is to learn the transition dynamics in the presence of exogenous actions performed by an agent. The aim is not to predict what actions the agent performs, but rather to predict the effects of the action on the state.

Another application area is relational reinforcement learning [120–122]. Here, the agent works out how to optimize its reward in an environment by constructing (using ILP) a first-order model of the dynamics of that environment, which it then uses to plan. Here, the Apperception Engine can be used to construct the dynamics model.

A related application area is learning game rules from player traces [123,96]. Here, the learning system is presented with traces (typically, sequences of sets of ground atoms), representing the state of the game at various points in time, and has to learn the transition dynamics (or reward function, or action legality function) of the underlying system.

A fourth related area is the predictive processing (PP) paradigm [124,6–8], an increasingly popular model in computational and cognitive neuroscience. Inspired by Helmholtz, the model learns to make sense of its sensory stream by attempting to predict future percepts. When the predicted percepts diverge from the actual percepts, the model updates its parameters to minimize prediction error. The PP model is probabilistic, Bayesian, and hierarchical: probabilistic in that the predicted sensory readings are represented as probability density functions, Bayesian in that the likelihood is combined with prior expectations [6], and hierarchical in that each layer provides predictions of sensory input for the layer below; there are typically many layers. While PP focuses on prediction, the Apperception Engine generates an interpretation that is equally adept at predicting future signals, retrodicting past signals, and imputing missing intermediary signals. In our approach, the ability to predict future signals is a derived capacity, a capacity that emerges from the more general capacity to construct a unified interpretation – but prediction is not singled out in particular. The Apperception Engine is able to predict, retrodict, and impute – in fact, it is able to do all three *simultaneously* using a single incomplete sensory sequence with elements missing at the beginning, at the end, and in the middle.

7.8. Summary

In summary, there are various other systems that construct dynamic rules for explaining sequences. But these systems are unable to posit latent hidden information to make sense of the sequence. These systems are able to predict future elements of the sequence, but are not able to retrodict earlier elements, or impute missing intermediary elements. For problems that require positing latent hidden information,[43] or problems that require retrodiction and imputation as well as prediction, the Apperception Engine is particularly well suited.

8. Conclusion

This paper is an attempt to answer a key question of unsupervised learning: what does it mean to "make sense" of a sensory sequence? Our answer is that making sense means constructing a symbolic theory containing a set of objects that persist over time, with attributes that change over time, according to general laws. This theory must both explain the sensory input, and satisfy the unity conditions of Section 3.3. As well as providing a precise formalization of this task, we also provide a concrete implementation of a system that is able to make sense of the sensory stream. We have tested the Apperception Engine in a variety of domains; in each domain, we tested its ability to predict future values, retrodict previous values, and impute missing intermediate values. Our system achieves good results across the board, outperforming neural network baselines and also state-of-the-art ILP systems.

Of particular note is that the Apperception Engine is able to achieve human performance on challenging sequence induction intelligence tests. We stress, once more, that the system was not hard-coded to solve these tasks. Rather, it is

[43] For example, making sense of the ECA sequences *without* spatial information (Section 5.1.2), or interpreting the *Seek Whence* sequences (Section 5.1.4), or the binding tasks (Section 5.1.6).

a general *domain-independent* sense-making system that is able to apply its general architecture to the particular problem of Seek Whence induction tasks, and is able to solve these problems "out of the box" without human hand-engineered help. We also stress, again, that the system did not learn to solve these sequence induction tasks by being presented with hundreds of training examples.[44] Indeed, the system had never seen a *single* such task before. Instead, it applied its general sense-making urge to each individual task, *de novo*.

Our architecture, an unsupervised program synthesis system, is a purely symbolic system, and as such, it inherits two key advantages of ILP systems [62]. First, the interpretations produced are *interpretable*. Because the output is symbolic, it can be read and verified by a human.[45] Second, it is very *data-efficient*. Because of the language bias of the Datalog$^{\ni}$ language, and the strong inductive bias provided by the unity conditions, the system is able to make sense of extremely short sequences of sensory data, without having seen any others.

However, the system in its current form has some limitations that we wish to make explicit. First, the sensory input must be discretized before it can be passed to the system. We assume some prior system has already discretized the continuous sensory values by grouping them into buckets. One possible approach to deal with continuous sensory values is to combine the Apperception Engine with a neural network that maps the raw continuous inputs into categories. We have recently developed such an extension, in which we discretize the input by simulating a binary neural network. The binary neural network is implemented in ASP, so the weights of the network and the rules of the theory can be found *simultaneously* by solving one large SAT problem.

Second, our implementation as described above assumes all causal rules are fully deterministic. It is quite straightforward to add non-determinism to the Datalog$^{\ni}$ framework: we can define an **extended theory** as a theory with initial conditions for *each time-step* (rather than only allowing initial conditions for the first time-step, as in Definition 2). An extended theory $\theta = (\phi, \{I_1, ..., I_T\}, R, C)$ generates a trace $\tau(\theta) = (A_1, A_2, ...)$ in exactly the same way as in Definition 9, with one small exception: $I_t \subseteq A_t$ replaces $I \subseteq A_1$. In other words, new atoms can be abduced at each time-step. This would allow us to handle non-determinism by abducing atoms that change their truth-value according to $\{I_1, ..., I_T\}$ instead of according to the rules in R.

Third, the size of the search space means that our system is currently restricted to small-to-medium-size problems.[46] Going forward, we believe that the right way to build complex theories is incrementally, using curriculum learning: the system should consolidate what it learns in one episode, storing it as background knowledge, and reusing it in subsequent episodes.[47]

We hope in future work to address these limitations. But we believe that, even in its current form, the Apperception Engine shows considerable promise as a prototype of what a general-purpose domain-independent sense-making machine must look like.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We are very grateful to Andrew Cropper, Jessica Hamrick, Mark Law, Matko Bošnjak, Murray Shanahan, Nando de Freitas, and the anonymous reviewers for extensive feedback. JHO has been partially supported by the Future of Life Institute, FLI, under grant RFP2-152, the EU (FEDER) and Spanish MINECO grant RTI2018-094403-B-C32 and Generalitat Valenciana under grant PROMETEO/2019/098.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.artint.2020.103438.

References

[1] M. Mathieu, C. Couprie, Y. LeCun, Deep multi-scale video prediction beyond mean square error, e-prints, arXiv:1511.05440, 2015.
[2] C. Finn, S. Levine, Deep visual foresight for planning robot motion, in: IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2017, pp. 2786–2793.
[3] S. Racanière, T. Weber, D. Reichert, L. Buesing, A. Guez, D.J. Rezende, A.P. Badia, O. Vinyals, N. Heess, Y. Li, et al., Imagination-augmented agents for deep reinforcement learning, in: Advances in Neural Information Processing Systems, NEURIPS, 2017, pp. 5690–5701.

[44] Barrett et al. [86] train a neural network to learn to solve Raven's progressive matrices from thousands of training examples.

[45] Large machine-generated programs are not always easy to understand. But machine-generated symbolic programs are certainly easier to understand than the weights of a neural network. See Muggleton et al. [23] for an extensive discussion.

[46] This is not because our system is carelessly implemented: the Apperception Engine is able to synthesize significantly larger programs than state-of-the-art ILP systems (see the comparison with ILASP in Section 5.2.4).

[47] A number of recent papers have shown the potential of reusing learned subroutines to synthesize progressively complex programs [125–130].

[4] Z. Liu, R.A. Yeh, X. Tang, Y. Liu, A. Agarwala, Video frame synthesis using deep voxel flow, in: ICCV, 2017, pp. 4473–4481.
[5] L. Buesing, T. Weber, S. Racaniere, S.M.A. Eslami, D. Rezende, D.P. Reichert, F. Viola, F. Besse, K. Gregor, D. Hassabis, D. Wierstra, Learning and querying fast generative models for reinforcement learning, e-prints, arXiv:1802.03006, 2018.
[6] K. Friston, The history of the future of the Bayesian brain, NeuroImage 62 (2) (2012) 1230–1233.
[7] A. Clark, Whatever next? Predictive brains, situated agents, and the future of cognitive science, Behav. Brain Sci. 36 (3) (2013) 181–204.
[8] L.R. Swanson, The predictive processing paradigm has roots in Kant, Front. Syst. Neurosci. 10 (2016) 79–102.
[9] J.B. Tenenbaum, T.L. Griffiths, C. Kemp, Theory-based bayesian models of inductive learning and reasoning, Trends Cogn. Sci. 10 (7) (2006) 309–318.
[10] H.M. Pasula, L.S. Zettlemoyer, L.P. Kaelbling, Learning symbolic models of stochastic domains, J. Artif. Intell. Res. 29 (2007) 309–352.
[11] O. Ray, Nonmonotonic abductive inductive learning, J. Appl. Log. 7 (3) (2009) 329–340.
[12] K. Inoue, T. Ribeiro, C. Sakama, Learning from interpretation transition, Mach. Learn. 94 (1) (2014) 51–79.
[13] B.M. Lake, T.D. Ullman, J.B. Tenenbaum, S.J. Gershman, Building machines that learn and think like people, Behav. Brain Sci. 40 (2017).
[14] G. Marcus, E. Davis, Rebooting AI, Pantheon, 2019.
[15] E.S. Spelke, K.D. Kinzler, Core knowledge, Dev. Sci. 10 (1) (2007) 89–96.
[16] C. Diuk, A. Cohen, M.L. Littman, An object-oriented representation for efficient reinforcement learning, in: Proceedings of the 25th International Conference on Machine Learning, 2008, pp. 240–247.
[17] J. McCarthy, Challenges to machine learning: relations between reality and appearance, in: International Conference on Inductive Logic Programming, Springer, 2006, pp. 2–9.
[18] K. Inoue, Meta-level abduction, FLAP 3 (1) (2016) 7–36.
[19] T. Teijeiro, P. Félix, On the adoption of abductive reasoning for time series interpretation, Artif. Intell. 262 (2018) 163–188.
[20] G.W. Leibniz, G.W.F. von Leibniz, Leibniz: New Essays on Human Understanding, Cambridge University Press, 1996.
[21] J. Dewey, Leibniz's New Essays Concerning the Human Understanding: A Critical Exposition, SC Griggs, 1888.
[22] K. Ellis, A. Solar-Lezama, J. Tenenbaum, Unsupervised learning by program synthesis, in: Advances in Neural Information Processing Systems, NEURIPS, 2015, pp. 973–981.
[23] S.H. Muggleton, U. Schmid, C. Zeller, A. Tamaddoni-Nezhad, T. Besold, Ultra-strong machine learning: comprehensibility of programs learned with ILP, Mach. Learn. 107 (7) (2018) 1119–1140.
[24] D.R. Hofstadter, Fluid Concepts and Creative Analogies, Basic Books, 1995.
[25] J. Hernández-Orallo, F. Martínez-Plumed, U. Schmid, M. Siebers, D.L. Dowe, Computer models solving intelligence test problems: progress and implications, Artif. Intell. 230 (2016) 74–107.
[26] K.J.W. Craik, The Nature of Explanation, vol. 445, CUP Archive, 1967.
[27] P.L. Harris, The Work of the Imagination, Blackwell Publishers, Oxford, 2000.
[28] T. Gerstenberg, J.B. Tenenbaum, Intuitive theories, in: Oxford Handbook of Causal Reasoning, 2017, pp. 515–548.
[29] L. Kaiser, M. Babaeizadeh, P. Milos, Model based reinforcement learning for Atari, preprint, arXiv:1903.00374v2, 2019.
[30] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al., A general reinforcement learning algorithm that masters chess, shogi, and go through self-play, Science 362 (6419) (2018) 1140–1144.
[31] J.B. Hamrick, Analogues of mental simulation and imagination in deep learning, Cur. Opin. Behav. Sci. 29 (2019) 8–16.
[32] L.E. Baum, T. Petrie, Statistical inference for probabilistic functions of finite state Markov chains, Ann. Math. Stat. 37 (6) (1966) 1554–1563.
[33] Z. Ghahramani, An introduction to hidden Markov models and Bayesian networks, in: Hidden Markov models: Applications in Computer Vision, World Scientific, 2001, pp. 9–41.
[34] V. Feinberg, A. Wan, I. Stoica, M. Jordan, J. Gonzalez, S. Levine, Model-based value expansion for efficient model-free reinforcement learning, in: Proceedings of the 35th International Conference on Machine Learning, ICML, 2018.
[35] A. Nagabandi, G. Kahn, R.S. Fearing, S. Levine, Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning, in: 2018 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2018, pp. 7559–7566.
[36] P. Battaglia, R. Pascanu, M. Lai, D.J. Rezende, et al., Interaction networks for learning about objects, relations and physics, in: Advances in Neural Information Processing Systems, NEURIPS, 2016, pp. 4502–4510.
[37] M.B. Chang, T. Ullman, A. Torralba, J.B. Tenenbaum, A compositional object-based approach to learning physical dynamics, e-prints, arXiv:1612.00341, 2016.
[38] D. Mrowca, C. Zhuang, E. Wang, N. Haber, L.F. Fei-Fei, J. Tenenbaum, D.L. Yamins, Flexible neural representation for physics prediction, in: Advances in Neural Information Processing Systems, NEURIPS, 2018, pp. 8813–8824.
[39] A. Sanchez-Gonzalez, N. Heess, J.T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, P. Battaglia, Graph networks as learnable physics engines for inference and control, preprint, arXiv:1806.01242, 2018.
[40] A. Lerer, S. Gross, R. Fergus, Learning physical intuition of block towers by example, e-prints, arXiv:1603.01312, 2016.
[41] A. Bhattacharyya, M. Malinowski, B. Schiele, M. Fritz, Long-term image boundary prediction, in: AAAI Conference on Artificial Intelligence, 2018, pp. 34–50.
[42] J. Oh, X. Guo, H. Lee, R.L. Lewis, S. Singh, Action-conditional video prediction using deep networks in Atari games, in: Advances in Neural Information Processing Systems, NEURIPS, 2015, pp. 2863–2871.
[43] S. Chiappa, S. Racaniere, D. Wierstra, S. Mohamed, Recurrent environment simulators, e-prints, arXiv:1704.02254, 2017.
[44] D. Ha, J. Schmidhuber, Recurrent world models facilitate policy evolution, in: Advances in Neural Information Processing Systems, NEURIPS, 2018, pp. 2455–2467.
[45] M. Janner, S. Levine, W.T. Freeman, J.B. Tenenbaum, C. Finn, J. Wu, Reasoning about physical interactions with object-oriented prediction and planning, e-prints, arXiv:1812.10972, 2018.
[46] A. Zhang, A. Lerer, S. Sukhbaatar, R. Fergus, A. Szlam, Composable planning with attributes, in: Proceedings of the International Conference on Machine Learning, vol. 11, ICML, 2018, pp. 11–22.
[47] Z. Xu, Z. Liu, C. Sun, K. Murphy, W. Freeman, J. Tennenbaum, J. Wu, Unsupervised discovery of parts, structure, and dynamics, in: Proceedings of the International Conference on Learning Representations, ICLR, 2019, pp. 1418–1424.
[48] M. Asai, A. Fukunaga, Classical planning in deep latent space: bridging the subsymbolic-symbolic boundary, in: AAAI Conference on Artificial Intelligence, 2018, pp. 45–62.
[49] M. Asai, Unsupervised grounding of plannable first-order logic representation from images, e-prints, arXiv:1902.08093, 2019.
[50] N. Katzouris, A. Artikis, G. Paliouras, Incremental learning of event definitions with inductive logic programming, Mach. Learn. 100 (2–3) (2015) 555–585.
[51] E. Michelioudakis, A. Skarlatidis, G. Paliouras, A. Artikis, Online structure learning using background knowledge axiomatization, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2016, pp. 232–247.
[52] N. Katzouris, A. Artikis, G. Paliouras, Online learning of event definitions, Theory Pract. Log. Program. 16 (5–6) (2016) 817–833.
[53] E. Michelioudakis, A. Artikis, G. Paliouras, Semi-supervised online structure learning for composite event recognition, preprint, arXiv:1803.00546, 2018.

[54] R. Kowalski, M. Sergot, A logic-based calculus of events, New Gener. Comput. 4 (1) (1986) 67–96.
[55] R. Kowalski, Predicate logic as programming language, in: IFIP Congress, Vol. 74, 1974, pp. 544–569.
[56] K.R. Apt, Logic programming, in: Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, 1990, pp. 493–574.
[57] J.W. Lloyd, Foundations of Logic Programming, Springer Science & Business Media, 1987.
[58] M.H. Van Emden, R.A. Kowalski, The semantics of predicate logic as a programming language, J. ACM 23 (4) (1976) 733–742.
[59] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: Proc. Fifth International Conference on Logic Programming, in: Logic Programming, vol. 88, MIT Press, 1988, pp. 1070–1080.
[60] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, T. Schaub, Asp-core-2 input language format, Theory Pract. Log. Program. 20 (2) (2020) 294–309.
[61] R. Kowalski, Logic for Problem-Solving, Elsevier North-Holland, 1979.
[62] R. Evans, E. Grefenstette, Learning explanatory rules from noisy data, J. Artif. Intell. Res. 61 (2018) 1–64.
[63] I. Kant, Critique of Pure Reason, Cambridge University Press, 1781.
[64] C. Baral, M. Gelfond, Logic programming and knowledge representation, J. Log. Program. 19 (1994) 73–148.
[65] K.L. Clark, Negation as failure, in: Logic and Databases, Springer, 1978, pp. 293–322.
[66] A. Van Gelder, K.A. Ross, J.S. Schlipf, The well-founded semantics for general logic programs, J. ACM 38 (3) (1991) 619–649.
[67] N.D. Goodman, T.D. Ullman, J.B. Tenenbaum, Learning a theory of causality, Psychol. Rev. 118 (1) (2011) 110.
[68] T.D. Ullman, N.D. Goodman, J.B. Tenenbaum, Theory learning as stochastic search in the language of thought, Cogn. Dev. 27 (4) (2012) 455–480.
[69] A.N. Kolmogorov, On tables of random numbers, Sankhyā, Ind. J. Stat., Ser. A (1963) 369–376.
[70] L.A. Levin, Universal sequential search problems, Probl. Pereda. Inf. 9 (3) (1973) 115–116.
[71] T.G. Dietterich, P. Domingos, L. Getoor, S. Muggleton, P. Tadepalli, Structured machine learning: the next ten years, Mach. Learn. 73 (1) (2008) 3.
[72] C. Henson, A. Sheth, K. Thirunarayan, Semantic perception: converting sensory observations to abstractions, IEEE Internet Comput. 16 (2) (2012) 26–34.
[73] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Clingo = ASP + control: preliminary report, e-prints, arXiv:1405.3694, 2014.
[74] R. Ben-Eliyahu, R. Dechter, Propositional semantics for disjunctive logic programs, Ann. Math. Artif. Intell. 12 (1–2) (1994) 53–87.
[75] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov, Complexity and expressive power of logic programming, ACM Comput. Surv. 33 (3) (2001) 374–425.
[76] G. Brewka, I. Niemelä, M. Truszczynski, Answer set optimization, in: Proceedings of the International Joint Conference on Artificial Intelligence, vol. 3, IJCAI, 2003, pp. 867–872.
[77] M. Gebser, R. Kaminski, T. Schaub, Complex optimization in answer set programming, Theory Pract. Log. Program. 11 (4–5) (2011) 821–839.
[78] R. Morel, A. Cropper, L. Ong, Typed meta-interpretive learning of logic programs, in: European Conference on Logics in Artificial Intelligence, JELIA, 2019, pp. 973–981.
[79] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Answer set solving in practice, Synth. Lect. Artif. Intell. Mach. Learn. 6 (3) (2012) 1–238.
[80] S. Wolfram, Statistical mechanics of cellular automata, Rev. Mod. Phys. 55 (3) (1983) 601.
[81] M. Cook, Universality in elementary cellular automata, Complex Syst. 15 (1) (2004) 1–40.
[82] M. Mitchell, Analogy-Making as Perception, MIT Press, 1993.
[83] L.L. Thurstone, T.G. Thurstone, Factorial Studies of Intelligence, Psychometric Monographs, 1941, pp. 8–23.
[84] J. Hernandez-Orallo, N. Minaya-Collado, A formal definition of intelligence, in: Proceedings of International Symposium of Engineering of Intelligent Systems, EIS 98, 1998, pp. 146–163.
[85] M.J.E. Meredith, Seek-Whence: a model of pattern perception, Tech. rep., Indiana University (USA), 1986.
[86] D.G.T. Barrett, F. Hill, A. Santoro, A.S. Morcos, T. Lillicrap, Measuring abstract reasoning in neural networks, e-prints, arXiv:1807.04225, 2018.
[87] A. Holcombe, The binding problem, in: The Sage Encyclopedia of Perception, 2009, pp. 47–71.
[88] K.P. Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012.
[89] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[90] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, e-prints, arXiv:1412.6980, 2014.
[91] S.H. Muggleton, D. Lin, A. Tamaddoni-Nezhad, Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited, Mach. Learn. 100 (1) (2015) 49–73, https://doi.org/10.1007/s10994-014-5471-y.
[92] A. Cropper, S.H. Muggleton, Learning efficient logic programs, Mach. Learn. (2018) 1–21, https://doi.org/10.1007/s10994-018-5712-6.
[93] A. Cropper, R. Evans, M. Law, Inductive general game playing, Mach. Learn. (2019) 1–42.
[94] T. Kaminski, T. Eiter, K. Inoue, Exploiting answer set programming with external sources for meta-interpretive learning, Theory Pract. Log. Program. 18 (3–4) (2018) 571–588.
[95] A. Cropper, R. Morel, S. Muggleton, Learning higher-order logic programs, Mach. Learn. 109 (2020) 1289–1322.
[96] M. Law, A. Russo, K. Broda, Inductive learning of answer set programs, in: European Conference on Logics in Artificial Intelligence - (JELIA), 2014, pp. 311–325.
[97] M. Law, A. Russo, K. Broda, Learning weak constraints in answer set programming, Theory Pract. Log. Program. 15 (4–5) (2015) 511–525.
[98] M. Law, A. Russo, K. Broda, Iterative learning of answer set programs from context dependent examples, Theory Pract. Log. Program. 16 (5–6) (2016) 834–848.
[99] M. Law, A. Russo, K. Broda, The complexity and generality of learning answer set programs, Artif. Intell. 259 (2018) 110–146.
[100] E.T. Mueller, Commonsense Reasoning, Morgan Kaufmann, 2014.
[101] M. Gelfond, Y. Kahl, Knowledge Representation, Reasoning, and the Design of Intelligent Agents, Cambridge University Press, 2014.
[102] A. Cropper, S.H. Muggleton, Learning higher-order logic programs through abstraction and invention, in: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI, IJCAI/AAAI Press, 2016, pp. 1418–1424.
[103] A. Cropper, Efficiently learning efficient programs, Ph.D. thesis, Imperial College, London, UK, 2017, http://hdl.handle.net/10044/1/58488.
[104] T. Ribeiro, K. Inoue, Learning prime implicant conditions from interpretation transition, in: Inductive Logic Programming, Springer, 2015, pp. 108–125.
[105] T. Ribeiro, M. Magnin, K. Inoue, C. Sakama, Learning multi-valued biological models with delayed influence from time-series observations, in: 2015 IEEE 14th International Conference on Machine Learning and Applications, ICMLA, 2015, pp. 25–31.
[106] D. Martínez, G. Alenya, T. Ribeiro, C. Torras, Relational reinforcement learning for planning with exogenous effects, J. Mach. Learn. Res. 18 (1) (2017) 2689–2732.
[107] T. Ribeiro, M. Folschette, M. Magnin, O. Roux, K. Inoue, Learning dynamics with synchronous, asynchronous and general semantics, in: International Conference on Inductive Logic Programming, Springer, 2018, pp. 118–140.
[108] T. Ribeiro, S. Tourret, M. Folschette, M. Magnin, D. Borzacchiello, F. Chinesta, O. Roux, K. Inoue, Inductive learning from state transitions over continuous domains, in: International Conference on Inductive Logic Programming, Springer, 2017, pp. 124–139.
[109] S. Tourret, E. Gentet, K. Inoue, Learning human-understandable description of dynamical systems from feed-forward neural networks, in: International Symposium on Neural Networks, Springer, 2017, pp. 483–492.
[110] Y. Phua, T. Ribeiro, S. Tourret, K. Inoue, Learning logic program representation for delayed systems with limited training data, in: Inductive Logic Programming, Springer, 2017, pp. 103–119.

[111] A. Solar-Lezama, L. Tancau, R. Bodik, S. Seshia, V. Saraswat, Combinatorial sketching for finite programs, ACM SIGPLAN Not. 41 (11) (2006) 404–415.
[112] D. Corapi, A. Russo, E. Lupu, Inductive logic programming in answer set programming, in: Inductive Logic Programming, Springer, 2012, pp. 91–97.
[113] M. Genesereth, N. Love, General Game Playing: Game Description Language Specification, Computer Science Department, Stanford University, Stanford, CA, USA, Tech. Rep., 2005, pp. 1–48.
[114] M. Genesereth, Y. Björnsson, The international general game playing competition, AI Mag. 34 (2) (2013) 107.
[115] S. Moyle, Using theory completion to learn a robot navigation control program, in: International Conference on Inductive Logic Programming, Springer, 2002, pp. 182–197.
[116] R.P. Otero, Induction of the indirect effects of actions by monotonic methods, in: International Conference on Inductive Logic Programming, Springer, 2005, pp. 279–294.
[117] K. Inoue, H. Bando, H. Nabeshima, Inducing causal laws by regular inference, in: International Conference on Inductive Logic Programming, Springer, 2005, pp. 154–171.
[118] D. Corapi, D. Sykes, K. Inoue, A. Russo, Probabilistic rule learning in nonmonotonic domains, in: International Workshop on Computational Logic in Multi-Agent Systems, Springer, 2011, pp. 243–258.
[119] C. Rodrigues, P. Gérard, C. Rouveirol, H. Soldano, Active learning of relational action models, in: International Conference on Inductive Logic Programming, Springer, 2011, pp. 302–316.
[120] S. Džeroski, L. De Raedt, K. Driessens, Relational reinforcement learning, Mach. Learn. 43 (1–2) (2001) 7–52.
[121] L. De Raedt, Logical and Relational Learning, Springer Science & Business Media, 2008.
[122] L. Bu, R. Babu, B. De Schutter, et al., A comprehensive survey of multiagent reinforcement learning, IEEE Trans. Syst. Man Cybern., Part C, Appl. Rev. 38 (2) (2008) 156–172.
[123] J. Goodacre, Inductive learning of chess rules using progol, Ph.D. thesis, University of Oxford, 1996.
[124] K. Friston, A theory of cortical responses, Philos. Trans. R. Soc. Lond. B, Biol. Sci. 360 (1456) (2005) 815–836.
[125] J. Schmidhuber, Optimal ordered problem solver, Mach. Learn. 54 (3) (2004) 211–254.
[126] E. Dechter, J. Malmaud, R.P. Adams, J.B. Tenenbaum, Bootstrap learning via modular concept discovery, in: Proceedings of the International Joint Conference on Artificial Intelligence, AAAI Press, 2013.
[127] R.J. Henderson, Cumulative learning in the lambda calculus, Ph.D. thesis, Imperial College London, 2013.
[128] D. Lin, E. Dechter, K. Ellis, J.B. Tenenbaum, S.H. Muggleton, Bias Reformulation for One-Shot Function Induction, 2014.
[129] K. Ellis, L. Morales, M. Sablé-Meyer, A. Solar-Lezama, J. Tenenbaum, Learning libraries of subroutines for neurally–guided bayesian program induction, in: Advances in Neural Information Processing Systems, 2018, pp. 7805–7815.
[130] K. Ellis, C. Wong, M. Nye, M. Sable-Meyer, L. Cary, L. Morales, L. Hewitt, A. Solar-Lezama, J.B. Tenenbaum, Dreamcoder: growing generalizable, interpretable knowledge with wake-sleep bayesian program learning, preprint, arXiv:2006.08381, 2020.