Imperial College London
Department of Computing

Kant's Cognitive Architecture

Richard Evans

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of Imperial College London and
the Diploma of Imperial College London, March 2020

Declaration of Originality

I, Richard Evans, declare that the work in this thesis is my own. The work of others has been appropriately referenced. A full list of references is given in the bibliography.

Abstract

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

In this thesis, we make three contributions. First, we provide a precise formalization of what it means to "make sense" of a sensory sequence. According to our definition, making sense means constructing a symbolic causal theory that explains the sensory sequence and satisfies a set of unity conditions that were inspired by Kant's discussion in the first half of the *Critique of Pure Reason*. According to our interpretation, 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 our requirements for making sense of a sensory sequence. 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 Kantian unity constraints. A causal theory produced by our system is able to predict future sensor readings, as well as retrodict earlier readings, and impute missing sensory readings. In fact, it is able to do all three tasks simultaneously. The engine is implemented in Answer Set Programming (ASP) and induces theories expressed in Datalog$^{\ni}$, an extension of Datalog that includes causal rules and constraints.

We test 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 IQ 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. These results are significant because neural nets typically struggle to solve the binding problem (where information from different modalities must somehow be combined together 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 IQ tasks, our system achieves human-level performance. This is notable because the Apperception Engine was not designed to solve these IQ tasks; it is not a bespoke hand-engineered solution to this particular domain. – Rather, it is a *general purpose* system that attempts to make sense of *any* sensory sequence, that just happens to be able to solve these IQ tasks "out of the box".

Our third contribution is a major extension of the engine to handle noisy and ambiguous data. While the initial implementation assumes the sensory input has already been preprocessed into ground atoms of first-order logic, our extension makes sense of *raw unprocessed input* – a sequence of pixel images from a video camera, for example. The resulting system is a neuro-symbolic framework for distilling interpretable theories out of streams of raw, unprocessed sensory experience.

Acknowledgements

I would like to thank my PhD supervisor, Professor Marek Sergot for his support and encouragement, his insight and acuity, his patience and generosity, throughout my PhD. I would also like to thank Andrew Stephenson, Jose Hernández-Orallo, Andrew Cropper, Mark Law, Ed Grefenstette, Matko Bošnjak, Kevin Ellis, Josh Tenenbaum, Daniel Selsam, Johannes Welbl, David Pfau, Pushmeet Kohli, Jessica Hamrick, Lars Buesing, Yujia Li, Rob Craven, Stephen Muggleton, Murray Shanahan, Krysia Broda, Robert Long, Nick Shea, Christopher Peacocke, Tom Smith, Demis Hassabis, Ian Holmes, Martin Berger, Ian Wright, Lewis Evans, and Barnaby Evans for insightful feedback. I am particularly grateful to Alessandra Russo and Michiel van Lambalgen for their thoughtful and penetrating comments. Thanks also to DeepMind for being such a stimulating and supportive place in which to do research. Last but not least, I thank my lovely wife Tiffy and our children - Barnaby, Molly, and Josie - for everything.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

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

In machine learning, this is called the *unsupervised learning problem*. It is both fundamentally important and frustratingly ill-defined.

This problem contrasts with the supervised learning problem where the sensory data comes attached with labels. In a supervised learning problem, there is a clear learning objective, and there are a number of powerful techniques that perform very successfully. However, *the real world does not come with labels attached to sensory data*. We just receive the data. As Geoffrey Hinton said[1]:

> When we're learning to see, nobody's telling us what the right answers are – we just look. Every so often, your mother says "that's a dog", but that's very little information. You'd be lucky if you got a few bits of information – even one bit per second – that way. The brain's visual system has 10^{14} neural connections. And you only live for 10^9 seconds. So it's no use learning one bit per second. You need more like 10^5 bits per second. And there's only one place you can get that much information: from the input itself.

In unsupervised learning, we are given a sequence of sensor readings, and want to make sense of that sequence. The trouble is we don't have a clear formalisable understanding of what it means to "make sense". Our problem, here, is *inarticulacy*. It isn't that we have a well-defined quantifiable objective and do not know the best way to optimize for that objective. Rather, we do not know what it is we really want.

[1] Quoted in Kevin Murphy's *Machine Learning: a Probabilistic Perspective* [Mur12].

One approach, the *self-supervised* approach, is to treat the sensory sequence as the input to a prediction problem: given a sequence of sensory data from time steps 1 to t, maximize the probability of the next datum at time $t + 1$. But we believe there is more to "making sense" than merely predicting future sensory readings. 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.

What, then, does it mean to make sense of a sensory sequence? In this thesis, I argue that the solution to this problem has been hiding in plain sight for over two hundred years. In the *Critique of Pure Reason*, Kant defines exactly what it means to make sense of a sequence: to reinterpret that sequence as a *representation of an external world composed of objects, persisting over time, with attributes that change over time, according to general laws*.

In this thesis, I reinterpret part of Kant's first *Critique* as a specification of a cognitive architecture, as a precise computationally-implementable description of what is involved, exactly, in making sense of the sensory stream. This is an interdisciplinary project and as such is in ever-present danger of falling between two stools, neither philosophically faithful to Kant's intentions nor contributing meaningfully to AI research. Kant himself provides[2]:

> the warning not to carry on at the same time two jobs which are very distinct in the way they are to be handled, for each of which a special talent is perhaps required, and the combination of which in one person produces only **bunglers** [AK 4:388]

The danger with an interdisciplinary project, part AI and part philosophy, is that both potential audiences are unsatisfied. The computer science might reasonably ask: why should a two hundred year old book have anything to teach us now? Surely if Kant had anything important to teach us, it would already have been absorbed? The Kant scholar might reasonably complain: is it really necessary to re-express Kant's theory using a computational formalism? We do not need these technicalities to talk about Kant. At best, it is an unnecessary re-articulation. At worst, misunderstandings are piled on misunderstandings, as Kant's ideas are inevitably distorted when shoe-horned into a simple computational formalism.

Nevertheless, I will argue, first, that contemporary AI has something to learn from Kant, and second, that Kant scholarship has something to gain when rearticulated in the language of computer science.

1.1.1 AI has something to learn from Kant

It is increasingly acknowledged that the strengths and weaknesses of neural networks and logic-based learning are complementary. While neural networks are robust to noisy or ambiguous data,

[2] Translations are from the Cambridge Edition of the Works of Immanuel Kant (details at the end), with occasional modifications. With the exception of those to the *Critique of Pure Reason*, which take the standard A/B format, references to Kant are by volume and page number in the Academy Edition [*Immanuel Kants gesammelte Schriften*, 29 volumes, Berlin: de Gruyter, 1902-].

and are able to absorb and compress the information from vast datasets, they are also data hungry, uninterpretable, and do not generalize well outside the training distribution [FP88, Mar18a, LUTG17, EG18]. Logic based learning, by contrast, is very data efficient, produces interpretable models, and can generalise well outside the training distribution, but struggles with noisy or ambiguous data[3], and finds it hard to scale to large datasets [RR16, EG18].

What we would really like, if only we can get it, is a system that combines the advantages of both. But this is, of course, much easier said than done. What, exactly, is involved in combining low-level perception with high-level conceptual thinking?

In the first *Critique* Kant describes, in remarkable detail, exactly what this hybrid architecture should look like. The reason why he was interested in hybrid cognitive architectures is because he was attempting to synthesise the two conflicting philosophical schools of the day, empiricism and rationalism. The neural network is the intellectual ancestor of empiricism, just as logic-based learning is the intellectual ancestor of rationalism. Kant's unification of empiricism and rationalism is a cognitive architecture that attempts to combine the best of both worlds, and points the way to a hybrid architecture that combines the best of neural networks and logic-based approaches.[4]

1.1.2 Kant interpretation has something to learn from AI

Some of the most exciting and ambitious work in recent philosophy [Bra94, Bra08, Bra09, Sel67, Sel68, Sel78] attempts to re-articulate Kantian (and post-Kantian) philosophy in the language of analytic philosophy. Now this re-articulation is not merely window-dressing; it is not just dressing up old ideas in the latest fashionable terminology. Rather, analytic philosophy, when done well, achieves a new level of perspicuity.

My aim in this thesis is to re-articulate Kant's theory at a *further* level of precision, by reinterpreting it as a specification of a *computational architecture*.

Why descend to this particular level of description? What could possibly be gained? The computational level of description is the ultimate level of precise description. There is no more precise you can be: even a mere *computer* can understand a computer program. Computers force us to clarify our thoughts. They admit no waffling or vagueness. Hand-waving is greeted with a compilation error, and a promissory note is returned, unread.

The advantage of re-articulating Kant's vision in computational terms is that it gives us a new level of specificity. The danger is that, in an effort to shoe-horn Kant's theory into a particular implementable system, we distort his original ideas to the point where they are no longer recognisable. Whether this is indeed the unfortunate consequence, the gentle reader must decide.

[3] Some recent systems are able to handle noisy (mislabelled) data effectively [LRB18b]. But, to the best of our knowledge, there are no such systems that handle ambiguous input data, such as the raw data from a video camera.

[4] So far, so programmatic. The hybrid neuro-symbolic architecture is described in Chapter 5, and the ascription of this architecture to Kant in particular is justified in Chapter 6.

1.2 Contributions

In this thesis, we make three contributions. First, we provide a precise formalization of what it means to "make sense" of a sensory sequence. According to our definition, making sense of a sensory sequence involves constructing a symbolic causal theory that explains the sensory sequence and satisfies a set of unity conditions that were inspired by Kant's discussion of the *synthetic unity of apperception* in the *Critique of Pure Reason*. According to our interpretation, 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 our requirements for making sense of a sensory sequence. 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 Kantian unity constraints. 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 fact, it is able to do all three tasks simultaneously. The engine is implemented in Answer Set Programming (ASP) and induces theories expressed in Datalog$^{\ni}$, a simple extension of Datalog to include constraints and causal rules. We show, in a range of experiments, that the engine significantly outperforms neural network baselines.

Our third contribution is a major extension of the engine to handle noisy and ambiguous data. While the initial implementation assumes the sensory input has already been preprocessed into ground atoms of first-order logic, our extension makes sense of *raw unprocessed input* – a sequence of pixel images from a video camera, for example. The resulting system is a neuro-symbolic framework for distilling interpretable theories out of streams of raw, unprocessed sensory experience.

1.2.1 Publications

Some of the work in this thesis has appeared in the following papers:

> Richard Evans. "Kant on Constituted Mental Activity", *The American Philosophical Association*, Volume 16, 2017.
>
> Richard Evans. "A Kantian Cognitive Architecture", *Philosophical Studies*, 2018.
>
> Richard Evans and Ed Grefenstette. "Learning Explanatory Rules from Noisy Data", *JAIR*, Volume 61, 2018.[5]
>
> Richard Evans, Andrew Stephenson, and Marek Sergot. "Formalizing Kant's Rules", *Journal of Philosophical Logic*, 2019.[6]

[5] I designed the system and wrote the first drafts of the paper. Ed Grefenstette designed some of the experiments and improved the text.

[6] I designed the logic and wrote the first draft of the paper. Marek Sergot developed the alternative semantics for KL1, developed the semantics for KL2, and improved the semantics for KL3. Andrew Stephenson improved the philosophical discussion and added further discussion of Kant. All three authors edited, revised, and polished the final draft.

Andrew Cropper, Mark Law, and Richard Evans. "Inductive General Game Playing", *Machine Learning*, 2019.[7]

Richard Evans. "Apperception", in *Human-Like Machine Intelligence*, Oxford University Press (forthcoming).

Richard Evans, Hernandez-Orallo, Johannes Welbl, Pushmeet Kohli, and Marek Sergot. "Making sense of sensory input", *Artificial Intelligence* (forthcoming).[8]

The following paper is under review:

Richard Evans, Matko Bosnjak, Lars Buesing, Kevin Elllis, Pushmeet Kohli, and Marek Sergot. "Making sense of raw input", *Artificial Intelligence*.[9]

1.3 Thesis structure

We first provide the necessary background material in Chapter 2 on logic programming and program synthesis.

Chapter 3 formalises what it means to make sense of a sensory sequence, culminating in the definition of the apperception task. We describe our system, the Apperception Engine, that is able to solve apperception tasks. In **Chapter 4**, we describe a range of experiments and compare with neural network baselines.

The main limitation of the approach in Chapter 3 is that it assumes the sensory input has already been preprocessed into ground atoms of first-order logic. **Chapter 5** removes this limitation, providing a neuro-symbolic architecture for extracting human-readable theories from raw input.

Chapter 6 describes the interpretation of Kant that underlies the Apperception Engine. Although it is usual to present the philosophical motivation before the technical material, this chapter can best be understood only *after* the technical material that precedes it. Readers who are not particularly interested in Kant exegesis should feel free to skip this chapter.

Chapter 7 discusses related work, and **Chapter 8** evaluates the system described, highlighting particular strengths of the approach, as well as limitations.

[7] I proposed the IGGP dataset as an ILP problem, generated the dataset, and wrote the first draft. Andrew Cropper ran the Metagol experiments and rewrote the paper, Mark Law ran the ILASP experiments and wrote the section on ILASP. All three authors edited, revised, and polished the final draft.

[8] I designed and implemented the system, designed the experiments, and wrote the first drafts of the paper. Jose Hernandez-Orallo improved the experiments and the experimental methodology. Johannes Welbl implemented the neural net baselines. Pushmeet Kohli is my advisor at DeepMind.

[9] I designed and implemented the system, designed the experiments, and wrote the first draft. Matko Bosnjak implemented the neural net baselines in Sections 5.5. Lars Buesing helped with the related work. Kevin Ellis helped with the derivation of the formulas in Section 5.3. Pushmeet Kohli is my advisor at DeepMind.

Chapter 2

Background

We first introduce logic programming in Answer Set Programming (ASP) and then describe one way to implement program synthesis in ASP.

2.1 Logic programming

In this thesis, we use basic concepts and standard notation from logic programming. We shall use $a, b, c, ...$ for constants, $X, Y, Z, ...$ for variables, $p, q, r, ...$ for predicate symbols, and $f, g, h, ...$ for function symbols.

A **term** is either simple or complex. A simple term is a constant or variable. A complex term is of the form $f(t_1, ..., t_n)$ where $t_1, ..., t_n$ are terms.

An **atom** is of the form $p(t_1, ..., t_n)$ where p is a predicate symbol, and $t_1, ..., t_n$ are terms. A **function-free atom** is an atom where all the terms are simple. If α is an atom, then $vars(\alpha)$ denotes the variables in α, so e.g. $vars(p(X, f(X, Y), Z)) = \{X, Y, Z\}$. An atom α is **ground** if $vars(\alpha) = \{\}$. We say an atom is **unground** if it contains no constants. According to this definition, some atoms are neither ground nor unground e.g. $p(a, X)$.

A **Datalog clause** is a definite clause of the form :

$$\alpha_1 \wedge ... \wedge \alpha_n \rightarrow \alpha_0$$

where each atom α_i is function-free and $n \geq 0$. Here, α_0 is the **head** and $\{\alpha_1, ..., \alpha_n\}$ is the **body** of the clause. It is traditional to write clauses from right to left: $\alpha_0 \leftarrow \alpha_1, ..., \alpha_n$. But in this thesis, we will define a Datalog interpreter implemented in another logic programming language (ASP). 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 clause $\alpha_1 \wedge ... \wedge \alpha_n \rightarrow \alpha_0$ is **safe** if $vars(\alpha_0) \subseteq \bigcup_{i=1}^{n} vars(\alpha_i)$. Throughout, we will restrict our attention to safe clauses. A clause is **ground** if each atom in the clause is ground (contains no variables).

The **Herbrand universe** of a logic program is the set of all ground terms formed from the constants and functions in the program. The Herbrand universe of a Datalog program is just the set of constants appearing in the program, and is always finite for a finite program. For logic programs that include function symbols, the Herbrand universe is not finite. The **Herbrand base** of a logic program is the set of all ground atoms that can be formed by applying the predicates to the terms of the Herbrand base. For Datalog programs, the Herbrand base is finite.

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. $p(X, Y)\sigma = p(a, b)$. Substitutions σ and σ' can be composed into $\sigma \circ \sigma'$ in the obvious way. There is an empty substitution $\epsilon = \{\}$ such that $\sigma \circ \epsilon = \epsilon \circ \sigma = \sigma$.

Given a set O of constants representing objects, the **grounding** of a clause is the set of all ground clauses obtained by applying all possible substitutions, replacing variables with objects in O.

A set of ground atoms M **satisfies** a ground clause $\alpha_1 \wedge ... \wedge \alpha_n \rightarrow \alpha_0$, written $M \models \alpha_1 \wedge ... \wedge \alpha_n \rightarrow \alpha_0$, if $\{\alpha_1, ...\alpha_n\} \subseteq M$ implies $\alpha_0 \in M$. A set M of atoms satisfies an unground clause if it satisfies all the ground instances of that clause. A **Herbrand model** of a logic program Π is a subset of the Herbrand base that satisfies all the clauses in Π.

If Π is a ground program and M is a set of ground atoms, let $T_\Pi(M)$ be the immediate consequence operator that generates the immediate single-step consequences of the rules in Π when given the atoms in M:

$$T_\Pi(M) = \{head(r) \mid r \in \Pi, body(r) \subseteq M\}$$

Let T_Π^∞ be the least fixpoint of T_Π, the repeated application of the immediate consequence operator until there are no more new consequences to derive.

A key result of logic programming is that every Datalog program has a unique subset-minimal Herbrand model, the **least Herbrand model**, that can be directly computed by repeatedly generating the consequences of the ground instances of the clauses [VEK76].

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 exponential time 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.

If Π is a Datalog program, and A and B are sets of ground atoms, then:

- the **data complexity** is the complexity of testing whether $\Pi \cup A \models B$, as a function of A and B, when Π is fixed

- the **program complexity** (also known as "expression complexity") is the complexity of testing whether $\Pi \cup A \models B$, as a function of Π and B, when A is fixed

Datalog has polynomial time data complexity but exponential time program complexity: deciding whether a ground atom is in the least Herbrand model of a Datalog program is in EXPTIME. The reason for this complexity is because the number of ground instances of a clause is an exponential function of the number of variables in the clause.

We turn now from Datalog to normal logic programs under the stable model (answer set) semantics [GL88]. A **literal** is an atom α or a negated atom *not* α. A **normal logic program** is a set of clauses of the form:

$$a_0 :\text{-}\, a_1, ..., a_n$$

where a_0 is an atom, $a_1, ..., a_n$ 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.

The **reduct** Π^M of a normal logic program Π w.r.t. a set M of atoms results from applying the following procedure to the grounding of Π: first remove every clause that contains a negative literal *not* α where $\alpha \in M$; second, remove every negative literal from the remaining clauses.

A **stable model** of a normal logic program Π is any Herbrand model M that is equal to the least Herbrand model of the reduct of Π w.r.t M. In other words, M is a stable model of Π if $M = T^{\infty}_{\Pi^M}$.

Unlike Datalog programs, which have a unique subset-minimal model, a normal logic program under the stable model semantics can have *multiple* stable models. For example, let Π be the normal logic program:

$$p :\text{-}\, not\ q$$
$$q :\text{-}\, not\ p$$

Here, Π has two stable models $\{p\}$ and $\{q\}$.

Answer Set Programming (ASP) is a logic programming language based on normal logic programs under the stable model semantics. Given a normal logic program, an ASP solver finds the set of stable models for that program.

A **choice rule** is a clause of the form:

$$\{a_1, ..., a_m\} :\text{-}\, a_{m+1}, ..., a_n$$

Intuitively, this rule means if conditions $a_{m+1}, ..., a_n$ hold, then feel free to add any subset of $\{a_1, ..., a_m\}$ to the database. For example, let Π be the normal logic program:

$$\{a, b\} :\text{-}\, c$$
$$c$$

Here, Π has four stable models: $\{c\}, \{a, c\}, \{b, c\}, \{a, b, c\}$.

Choice rules are just syntactic sugar for sets of normal logic rules. For example, $\{a_0\}\texttt{:-}\,a_1, ..., a_n$ is short-hand for the pair of normal logic rules:

$$a_0\texttt{:-}\,a_1, ..., a_n,\ \texttt{not}\ \overline{a_0}$$
$$\overline{a_0}\texttt{:-}\,a_1, ..., a_n,\ \texttt{not}\ a_0$$

Here, $\overline{a_0}$ is a fresh atom not appearing elsewhere in the program representing that a_0 is false.

A **constraint** is a clause that rules out a certain combination of literals:

$$\texttt{:-}\,\mathtt{a_1}, ..., \mathtt{a_n}.$$

This rules out stable models in which $\mathtt{a_1}, ..., \mathtt{a_n}$ are all true. It is short-hand for:

$$\mathtt{p}\,\texttt{:-}\,\mathtt{a_1}, ..., \mathtt{a_n},\ \texttt{not p}$$

Here, `p` is a fresh atom not appearing elsewhere in the program.

Modern ASP solvers can also be used to solve optimization problems by the introduction of weak constraints. A **weak constraint** is a rule that defines the cost of a certain tuple of atoms. A weak constraint is of the form:

$$:\sim \mathtt{a_1}, ..., \mathtt{a_n}.[\mathtt{w@p}, \mathtt{t_1}, ..., \mathtt{t_m}]$$

Here, $\mathtt{a_1}, ..., \mathtt{a_n}$ are literals, `w` is the (integer) cost of this set of literals, `p` is the (integer) priority level, and $\mathtt{t_1}, ..., \mathtt{t_m}$ are terms for determining which aspects of the literals should be considered unique [CFG+12]. Given a program with weak constraints, an ASP solver can find a preferred answer set with the lowest cost. It does this by computing the total summed cost for each answer set at each priority level, and then finding the lowest cost answer at the highest priority level; if there are multiple answers with the same cost, it considers the next priority level down, and so on.

ASP solvers work by first grounding the first-order logic program into a set of ground clauses, and then using a modified SAT solver[1] to find stable models of the ground program. Finding a solution to an ASP program is in NP [BED94, DEGV01], while finding an optimal solution to an ASP program with weak constraints is in Σ_2^P [BNT03, GKS11].

[1] A SAT solver is a program that takes a formula of propositional logic and attempts to find a satisfying assignment: a mapping from propositional variables to $\{\mathit{True}, \mathit{False}\}$ that makes the formula true.

2.2 Program synthesis and inductive logic programming

Given a partial specification ϕ and a language L, the program synthesis problem[2] is to find a program Π in L such that $\phi(\Pi)$. In functional program synthesis, the specification ϕ often involves a set of input-output examples. For example:

$$\begin{aligned} p(1) &= 1 \\ p(2) &= 4 \\ p(3) &= 9 \\ p(4) &= 16 \end{aligned}$$

In inductive logic programming (ILP), the program Π defines a relation, and the specification involves a set E^+ of positive examples together with a set E^- of negative examples. For example:

$$E^+ = \begin{cases} p(1,1) \\ p(2,4) \\ p(3,9) \\ p(4,16) \end{cases} \qquad E^- = \begin{cases} p(1,2) \\ p(2,1) \\ p(3,5) \\ p(4,4) \end{cases}$$

The partial specification does not have to be so direct. It can be any property of the program. We could ask for a program that terminates in exactly five time-steps, or for a program that contains at least three for-loops.

In this thesis, the program specification will be rather indirect: given a sequence of sensory inputs, find a program that makes sense of that sequence. What, exactly, it means to "make sense" of a sequence will be explained in due course.

Until a few years ago, ILP techniques were restricted to learning simple logic programs; they were unable to learn recursive programs or learn programs that make use of additional newly defined predicates (predicate invention). This changed with the introduction of TAL [CRL11a, Cor12, CRL10a] and Metagol [MLPTN14]. These approaches used meta-interpretive learning, providing a practical technique for learning recursive programs that used predicate invention. The next section describes the ideas behind meta-interpretive learning.

2.3 Program synthesis via an interpreter

Suppose we want to write a program in one language (a meta-language) that induces a program in another language L (the target language) satisfying a specification ϕ. One general class of approaches

[2] In this thesis, I use " program synthesis" to mean the search for a program meeting a specification. This specification need not be a complete formal specification, but may be partial (e.g. a small set of input/output examples) [GPS+17].

solves the induction problem by implementing an interpreter of L in the meta-language. Now, armed with an interpreter, the problem of finding a program in L satisfying ϕ is transformed into the problem of finding an object in the meta-language that, when interpreted by the interpreter, satisfies ϕ. We have reduced the induction problem (finding a program in L) to an abduction problem (finding an object in the meta-language) [MLPTN14], and can now use standard search techniques.

One prominent example of this class of approaches in ILP is meta-interpretive learning [MLT15, MLT15, CM16]. In this case, Metagol induces programs in Prolog by having a meta-interpreter of Prolog that is itself written in Prolog. But in general the target language and meta-language do not need to coincide.

In this thesis, we use ASP as the meta-language. Applying the general approach to ASP requires us to:

1. Represent each program in L by a set of ground atoms in ASP.
2. Implement the semantics of L by a set of clauses in ASP. The interpreter takes a program in L (represented by a set of atoms) and an input (also represented by a set of atoms), and produces an execution trace for that program (again represented by a set of atoms).
3. Implement the specification ϕ as an ASP constraint that checks that the execution trace does indeed satisfy ϕ.
4. Implement the search over programs in L by means of a set of ASP choice rules that choose various sets of atoms representing the various programs in L.
5. Add a weak constraint that minimises the size of the induced program.

We illustrate the technique with a simple example: synthesising finite state machines from sets of acceptable and unacceptable strings.

Suppose, for example, we have the following acceptable and unacceptable strings:

Acceptable	**Unacceptable**
a	*ab*
bb	*ba*
abb	*aab*
bab	*bbb*
bba	*aba*

We want to synthesise a finite state machine (FSM) that accepts strings when they have an even number of b's.

First, we need to represent each FSM by a set of ground atoms. We shall use *state*(S) to represent that state S is used in the machine. We use natural numbers to represent states, and fix that state

1 is always the initial state. We use *final*(S) to represent that S is a final (accepting) state. We use *trans*(S, A, S_2) to represent that there is a transition from S to S_2 when given symbol A.

So, for example, a FSM that accepts the regular language ab^* is represented as:

$$
\begin{array}{l}
state(1).\\
state(2).\\
final(2).\\
trans(1, a, 2).\\
trans(2, b, 2).
\end{array}
$$

Second, we implement the semantics of the FSM by the following clauses:

$$in(E, 1, 1) :\text{-}\, example(E).$$

$$in(E, T + 1, S2) :\text{-}\, in(E, T, S), trans(S, A, S2), seq(E, T, A).$$

$$succeed(E) :\text{-}\, end(E, T), in(E, T, S), final(S).$$

Here, *in*(E, T, S) means that for example string E, at time step T the machine is in state S; *succeed*(E) means that the FSM accepts example string E; *seq*(E, T, A) means that for example string E, the symbol at position T is A ; and *end*(E, T) means that time step T is the final time step for example string E.

The first clause states that for every example string, the FSM starts off in the initial state at the initial time step. The second clause states that we move from state S to S_2 if there is a transition from state S when receiving symbol A. The third clause states that we accept a string if we are in a final state at the end of the computation.

Third, we implement the specification as an ASP constraint. We want the FSM to accept the acceptable strings and reject the unacceptable ones. We represent example string E of length T using T atoms of the form *seq*(E, T, A), representing that the T'th symbol of example string E is A. For example, the acceptable string a and unacceptable string ab are represented by:

$$
\begin{array}{l}
seq(e_1, 1, a).\\
accept(e_1).\\
\\
seq(e_2, 1, a).\\
seq(e_2, 2, b).\\
reject(e_2).
\end{array}
$$

We add constraints to check that the right strings are accepted:

$$:\text{-}\, accept(E),\ \texttt{not}\ succeed(E).$$

$$:\text{-}\, reject(E), succeed(E).$$

Fourth, we implement the following choice rules to search over FSM machines[3]:

$$possible(1..max_s).$$

$$state(1).$$

$$\{state(S)\} :\text{-}\, possible(S).$$

$$\{final(S)\} :\text{-}\, state(S).$$

$$\{trans(S, A, S2)\} :\text{-}\, state(S), symbol(A), state(S2).$$

Here, we insist that there is at least one state, the initial state 1. We use *possible*(S) to represent that state S might be used in the FSM, and *state*(S) to mean that S is actually used. The first choice rule allows as many possible states (from 1 to max_s) as we need. The second choice rule chooses any subset of states to be final. The third choice rule chooses the transitions between states.

We use the following helper functions:

$$example(E) :\text{-}\, seq(E, \_, \_).$$

$$symbol(A) :\text{-}\, seq(\_, \_, A).$$

$$end(E, N + 1) :\text{-}\, seq2(E, N),\ \texttt{not}\ seq2(E, N + 1).$$

$$seq2(E, N) :\text{-}\, seq(E, N, \_).$$

The above code synthesises FSMs from examples. If we want to find the shortest FSM that solves the examples, we just add the weak constraint:

$$:\sim trans(S, A, S2).[1@1, S, A, S2]$$

[3] The line possible(1..$\mathtt{max}_s$) uses the .. syntactic sugar. This is short-hand for possible(1), ..., possible($\mathtt{max}_s$).

When we run this code on the acceptable and unacceptable strings above, it produces the following FSM describing the language $a^*(bb)^*$:

$$
\begin{array}{l}
state(1).\\
state(2).\\
trans(1, a, 1).\\
trans(1, b, 2).\\
trans(2, a, 2).\\
trans(2, b, 1).\\
final(1).
\end{array}
$$

In this thesis, we shall synthesise programs in an extension of Datalog, and so the interpreter will be somewhat more involved than that for the FSM. But the basic technique remains essentially the same.

2.4 Neural networks

We shall make occasional use of neural networks, both as baselines (Section 6.12.3), and as parts of a larger system (Chapter 5). An introduction to neural networks is beyond the scope of this thesis and we refer to [Mur12]. The key distinction we shall use is between a **feed-forward neural network** (in which connections between nodes form a directed acylic graph) and **recurrent neural networks** in which connections between nodes may form cycles. The simplest feed-forward network is the **fully-connected multi-layer perceptron**, in which the nodes are divided into layers $L_1, L_2, ...,$ and there is a connection from each node in layer L_i to each node in L_{i+1}. The most common form of recurrent network is the LSTM [HS97]. References to specific techniques and methods are given in the text where they are first used.

In Chapter 5, we use a binary neural network [HCS$^+$16, KS16, CNHR18, NKR$^+$18] as a parameterised perceptual classifier. Binary neural networks (BNNs) are increasingly popular because they are more efficient (both in memory and processing) than standard artificial neural networks. But our interest in BNNs is not so much in their resource efficiency as in their *discreteness*.

In the BNNs that we use [CNHR18], the node activations and weights are all binary values in $\{0, 1\}$. If a node has n binary inputs $x_1, ..., x_n$, with associated binary weights $w_1, ..., w_n$, the node is activated if the total sum of inputs *xnor*-ed with their weights is greater than or equal to half the number of inputs. In other words, the node is activated if

$$\sum_{i=1}^{n} \mathbb{1}[x_i = w_i] \geq \left\lceil \frac{n}{2} \right\rceil$$

Chapter 3

Making sense of discrete input

This material is based on "Making sense of sensory input", forthcoming in *Artificial Intelligence*.[1] It is also based on my article "Apperception", in *Human-Like Machine Intelligence*, Oxford University Press, 2020 (forthcoming).

What does it mean to make sense of a sensory sequence? In this chapter, we formalize what this means, and describe our implementation. For now, we assume that the sensory sequence has already been discretised into ground atoms of first-order logic representing sensor readings. In the next chapter, we let go of our simplifying assumption of already-discretised sensory input and consider sequences of *raw unprocessed input*: consider, for example, a sequence of pixel arrays from a video camera.

But for now, assume that the sensor readings have already been discretized, 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. In this thesis, for performance reasons, we restrict our attention to unary and binary predicates.

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* $S \in \left(2^{\mathcal{G}}\right)^*$.

△

[1] The paper is co-authored with Jose Hernandez-Orallo, Johannes Welbl, Pushmeet Kohli, and Marek Sergot. Jose Hernandez-Orallo improved the experiments and the experimental methodology. Johannes Welbl implemented the neural net baselines. Pushmeet Kohli is my advisor at DeepMind.

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*.

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

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 incompossible with it. Two facts are incompossible 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 $\kappa_O : O \rightarrow T$ for the type of an object, $\kappa_P : P \rightarrow T^*$ for the types of the predicate's arguments, and $\kappa_V : V \rightarrow T$ for the type of a variable.* △

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) where:

$$\begin{array}{rcl} T & = & \{s\} \\ O & = & \{a{:}s, b{:}s\} \\ P & = & \{on(s), off(s)\} \\ V & = & \{X{:}s, Y{:}s\} \end{array}$$

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 $U_\phi = \{on(X), \mathit{off}(X), on(Y), \mathit{off}(Y)\}$. 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 rules define the dynamics of the theory:

Definition 7. *There are two types of rule in Datalog$^{\ni\!\!\!\rightarrow}$. 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 $\alpha_1 \wedge ... \wedge \alpha_n \mathrel{\ni\!\!\!\rightarrow} \alpha_0$, 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 $\alpha_1 \wedge ... \wedge \alpha_n \mathrel{\ni\!\!\!\rightarrow} \alpha_0$ 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, $on(X) \mathrel{\ni\!\!\!\rightarrow} \mathit{off}(X)$ 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 from co-occurring in any state in the sequence[2]:

Definition 8. *There are three types of constraint in Datalog$^{\ni\!\!\!\rightarrow}$. 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.[3]

[2] 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.

[3] This restriction also occurs in some ILP systems [IRS14, EG18]. See [Lon98, ESS19] for a Kantian justification.

3.2 Explaining the sensory sequence

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

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 $\beta_1 \wedge ... \wedge \beta_m \mathrel{\ni\!\!\!\!-} \alpha$ 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 incompossible with it.[4] 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.

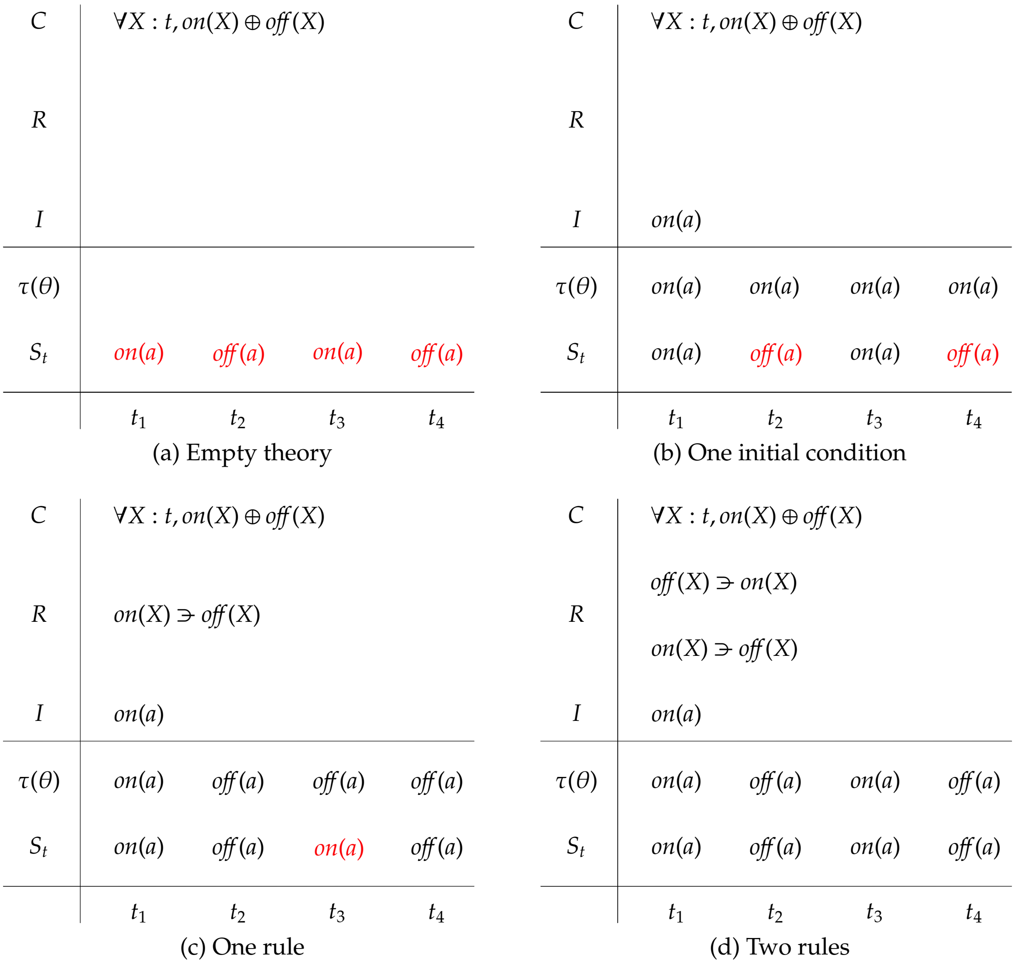
Note that the state transition function is deterministic: A_t is uniquely determined by A_{t-1}.

Theorem 1. *The trace of every theory repeats after some finite number of steps. For any theory θ, there exists a $k > 0$ 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. As the set G_ϕ of ground atoms is finite, there must be a k such that $A_1 = A_k$, since each A_i is a subset of G_ϕ, and there are only a finite number of such subsets. 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:

[4] In the *Metaphysical Foundations of Natural Science* 4, 543.15-20, Kant held that the law of inertia is *a priori*.



C	$\forall X : t, on(X) \oplus off(X)$			
R				
I				
$\tau(\theta)$				
S_t	$on(a)$	$off(a)$	$on(a)$	$off(a)$
	t_1	t_2	t_3	t_4

(a) Empty theory

C	$\forall X : t, on(X) \oplus off(X)$			
R				
I	$on(a)$			
$\tau(\theta)$	$on(a)$	$on(a)$	$on(a)$	$on(a)$
S_t	$on(a)$	$off(a)$	$on(a)$	$off(a)$
	t_1	t_2	t_3	t_4

(b) One initial condition

C	$\forall X : t, on(X) \oplus off(X)$			
R	$on(X) \ni off(X)$			
I	$on(a)$			
$\tau(\theta)$	$on(a)$	$off(a)$	$off(a)$	$off(a)$
S_t	$on(a)$	$off(a)$	$on(a)$	$off(a)$
	t_1	t_2	t_3	t_4

(c) One rule

C	$\forall X : t, on(X) \oplus off(X)$			
R	$off(X) \ni on(X)$ $on(X) \ni off(X)$			
I	$on(a)$			
$\tau(\theta)$	$on(a)$	$off(a)$	$on(a)$	$off(a)$
S_t	$on(a)$	$off(a)$	$on(a)$	$off(a)$
	t_1	t_2	t_3	t_4

(d) Two rules

Figure 3.1: Four candidate theories attempting to explain the sequence (S_1, S_2, S_3, S_4) where $S_1 = S_3 = \{on(a)\}$ and $S_2 = S_4 = \{off(a)\}$. In each sub-figure, we show at the top the theory θ composed of constraints C (fixed), rules R, and initial conditions I; below, we show the trace of the theory, $\tau(\theta)$, and the state sequence(S_1, S_2, S_3, S_4). When the trace at time t fails to be a superset of the state S_t, we color the state S_t in red. Sub-figure **(a)** shows the initial theory, with empty initial conditions and rules. This fails to explain any of the sensory states. In **(b)**, we add one initial condition. The atom $on(a)$ persists throughout the time series because of the frame axiom. In **(c)**, we add one causal rule. This changes $on(a)$ at t_1 to $off(a)$ at t_2. But $off(a)$ then persists because of the frame axiom. In **(d)**, we add another causal rule. At this point, the trace $\tau(\theta)$ covers the sequence.

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 $P = \{on(s), \mathit{off}(s), p_1(s), p_2(s), p_3(s), r(s,s)\}$, 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.[5]

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

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

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

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

[5] 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 3.7.1.

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 formalize what it means for a theory to be "unified" using elements from Kant's discussion of the "synthetic unity of apperception".[6]

Definition 11. *A trace $\tau(\theta)$ is (i) a 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 the following four levels:*

1. *Objects are united via chains of binary relations (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 Object connectedness

Definition 12. *A theory θ satisfies* **object connectedness** *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 $\{r_1(x, z_1), r_2(z_1, z_2), ...r_n(z_{n-1}, z_n), r_{n+1}(z_n, y)\} \subseteq A_t$.* △

[6] In this chapter, we do not focus on Kant exegesis, but do provide some key references. "The principle of the synthetic unity of apperception is the supreme principle of all use of the understanding" [B136]; it is "the highest point to which one must affix all use of the understanding, even the whole of logic and, after it, transcendental philosophy" [B134]. For more discussion of Kant's theory of apperception, see Chapter 6.

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.[7]

Note that this notion of connectedness 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 any of these binary relations have a specifically "spatial" interpretation.[8]

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 xor constraints combine predicates into clusters of mutual incompatibility.[9]

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[10] , to provide determinacy in each state, and to ground the incompossibility relation that constrains the way information is propagated between states.[11]

[7] See [A211-5/B255-62], [B203].

[8] For a more substantial notion of spatial unity, see Section 6.5.

[9] See [A73-4/B98-9]. See also [*Jäsche Logic* 9:107n].

[10] See [A103-11]. See also: "What the form of disjunctive judgment may do is contribute to the acts of forming categorical and hypothetical judgments the perspective of their possible *systematic unity*", [Lon98, p.105]

[11] A natural question to ask at this point is: why use exclusive disjunction to represent constraints? Why not instead

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.[12] If A does not satisfy the constraints, it is not a coherent and determinate representation; it is "less even than a dream."[13] This motivates the following definition:

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.* △

Note that, from the definition of the trace in Definition 9, all the states in $\tau(\theta)$ are automatically closed

represent constraints using strong negation or negation as failure? An exclusive disjunction can always be converted into a set of extended clauses representing the predicates' exclusiveness, and one normal clause representing their exhaustiveness. For example, $\forall X : t, p(X) \oplus q(X)$ can be rendered as:

```
¬p(X):-q(X)
¬q(X):-p(X)
:- not p(X), not q(X), t(X)
```

In general, if we have an exclusive disjunction featuring n predicates, we can turn this into $n * (n-1)$ clauses (using strong negation) to capture the exclusiveness of the n predicates, and one clause (using negation as failure) to capture the exhaustiveness.

The exclusive disjunction constraint is a compact way of representing a lot of information about the connection between predicates. Although the exclusive disjunction constraint can always be translated into a set of clauses (using both negation as failure and strong negation), the representation using exclusive disjunction is much more compact.

One reason, then, for expressing the constraint as an exclusive disjunction is that it is a significantly more compact representation than the representation using negation as failure. But another, more substantial reason is that it means we can avoid the complexities involved in the semantics if we added negation as failure to our target language Datalog$^{\ni}$. There are various semantics for normal logic programs that include negation as failure (e.g. Clark completion [Cla78], stable model semantics [GL88], well-founded models), 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. (Although we do plan to extend our rules to include stratified negation, as this does not complicate the semantics in the same way that unrestricted negation does). The inner loop of our program synthesis system is the calculation of the trace $\tau(\theta)$ by executing a Datalog$^{\ni}$ program, so it is essential that the execution is as efficient as possible. Hence our strong preference for definite logic programs over normal logic programs.

Why do we not allow more complex constraints (e.g. allowing any first-order sentence to be a constraint)? If we allowed any arbitrary set of first-order formulas as constraints, then computing the incompossibility relation would become much harder, given that computing entailment in first-order logic is only semi-decidable. The reason, then, why we focus on xor constraints is that they are the simplest construct that generates the incompossibility relation needed to constrain the frame axiom.

[12] See the *schema of community* [A144/B183-4].

[13] See [A112].

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*.[14] These causal rules are *universal* in two senses: they apply to all object tuples, and they apply at all times. A causal rule $\alpha_1 \wedge ... \wedge \alpha_n \ni\!\!\!\!- \alpha_0$ 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 $t+1$). According to Kant[15], the *only thing* that can fix the succession relation between states is the universal causal rule.

Imagine that, instead, we posit a finite sequence of states extensionally (rather than intensionally via initial conditions and rules). Here, our alternative "interpretation" of the sensory sequence S is just a finite S' where $S \sqsubseteq S'$. This S' is *arbitrary* because it is not generated by rules that confer on it the "dignity of necessity"[16]. In a unified interpretation, by contrast, the states are united in a sequence by being necessitated by universal causal rules. The above discussion motivates the following:

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

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. *Object connectedness*: objects are united 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

[14] See the *schema of causality* [A144/B183].

[15] See [B233-4].

[16] See [A91/B124].

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: object connectedness, 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 object connectedness 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 $\tau(\theta) = (A_1, ...)$).

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

- 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 $\{r(b, a)\}$.
- 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 $\forall X{:}s,\ \exists! Y{:}s\ r(X, Y)$.
- 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.
- Temporal unity is automatically satisfied by the definition of the trace $\tau(\theta)$ in Definition 9. ◁

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.* △

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[17]:

[17] 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 [Kol63] of θ: the size of the smallest program that can generate θ. Or we could use Levin complexity [Lev73] 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 17. *Given a theory $\theta = (\phi, I, R, C)$, the* **cost** *of θ is*

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

Here, cost(θ) 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 chapter 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. It might be objected: why make things so complicated? 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 $\phi = (T, O, P, V)$:

$$\begin{array}{rcl} T & = & \{t\} \\ O & = & \{a{:}t\} \\ P & = & \{on(t), off(t), p(t), q(t)\} \\ V & = & \{X{:}t\} \end{array}$$

and I, R, C are defined as:

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

Here we have introduced two latent predicates p and q which are incompossible with *on* and *off* respectively. But in this interpretation, *on* and *off* are not incompossible 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. ◁

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.

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*.

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

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

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:

$$\begin{aligned} T &= \{\mathit{sensor}\} \\ O &= \{a{:}\mathit{sensor}, b{:}\mathit{sensor}\} \\ P &= \{\mathit{on}(\mathit{sensor}), \mathit{off}(\mathit{sensor}), r(\mathit{sensor}, \mathit{sensor}), p_1(\mathit{sensor}), p_2(\mathit{sensor}), p_3(\mathit{sensor})\} \\ V &= \{X{:}\mathit{sensor}, Y{:}\mathit{sensor}\} \end{aligned}$$

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:

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

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 $p_1(Y) \rightarrow \mathit{on}(X)$ 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 third constraint $\forall X{:}\mathit{sensor},\ \exists! Y{:}\mathit{sensor}\ r(X,Y)$ is used to satisfy the constraint of object connectedness.

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

$$A_{10} = \{\mathit{on}(a), \mathit{on}(b), r(a,b), r(b,a), p_2(a), p_1(b)\}$$

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).[18] 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.

$\triangleleft$

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. Dietterich et al. [DDG$^+$08] anticipated the need for it:

> It is a characteristic of many scientific domains that we need to posit the existence of hidden objects in order to achieve compact hypotheses which explain empirical observations. We will refer to this process as object invention. For instance, object invention is required when unknown enzymes produce observable effects related to a given metabolic network.

However, although the need for it has been recognised, object invention remains largely unexplored.

Our new type signature $\phi' = (T, O, P, V)$ is:

$$\begin{aligned}
T &= \{sensor\} \\
O &= \{a{:}sensor, b{:}sensor, c{:}sensor\} \\
P &= \{on(sensor), \mathit{off}(sensor), r(sensor, sensor)\} \\
V &= \{X{:}sensor, Y{:}sensor\}
\end{aligned}$$

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 $t + 1$.

[18] 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.

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:

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

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:

$$\begin{array}{rcl} T & = & \{cell, sensor\} \\ O & = & \{a : sensor, b : sensor, c_1 : cell, c_2 : cell, c_3 : cell\} \\ P & = & \{on(sensor), off(sensor), part(sensor, cell), r(cell, cell), black(cell), white(cell)\} \\ V & = & \{X : sensor, Y : cell, Y_2 : cell\} \end{array}$$

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

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

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 object connectedness 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:

$$part(a, c_3), r(c_3, c_1), part(b, c_1)$$

Object connectedness 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, a Leibnitzian monad. 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.[19]

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

[19] As Kant says, "Every motion, as object of possible experience, can be viewed arbitrarily as motion of the body in a space at rest or as the contrary motion of the space in the opposite direction with the same speed." *Metaphysical Foundations of Natural Science* 487.16, quoted in [Fri92].

complex dynamics. These are just three theories among many; there are always an infinite number of distinct theories that make sense of any sequence.

3.6 Properties of interpretations

In this section, we provide some general results about unified interpretations. We show that every unified interpretation assigns some property to each sensor in each time-step, and we show that every sensory sequence has at least one unified interpretation.

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 $q(x, y) \in A_i$. □

Theorem 3 provides some guarantee that admissible interpretations that satisfy the Kantian 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 p of the atom in which x appears is one of the predicates appearing in the sensory sequence S. It is entirely possible that p 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:

- $T = \{t\}$
- $O = \{a\}$
- $P = \{p(t), q(t)\}$
- $V = \{X : t\}$

Suppose our interpretation is (ϕ, I, R, C) where:

- $I = \{p(a)\}$
- $R = \{p(X) \ni\!\!\!- q(X)\}$
- $C = \{\forall X : t, p(X) \oplus q(X)\}$

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 shows that every sensory sequence is solvable, i.e. every sequence has some admissible interpretation that satisfies the Kantian 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. 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):

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

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 object connectedness, 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:

$$\begin{array}{ll} S_1 = \{on(a)\} & S_2 = \{off(a)\} \\ S_3 = \{on(a)\} & S_4 = \{off(a)\} \\ S_5 = \{on(a)\} & S_6 = \{off(a)\} \\ S_7 = \{on(a)\} & \end{array}$$

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:

$$\begin{array}{rcl} T' & = & \{t, t_w\} \\ O' & = & \{a, w\} \\ P' & = & P \cup \{p_1(t), p_2(t), ..., p_7(t), part(t, t_w)\} \\ V' & = & \{X : t, Y : t_w\} \end{array}$$

Our interpretation is (ϕ', I, R, C') where:

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

◁

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 needed to make sense of the sensory sequence. We can use this upper bound to define the randomness of a sensory sequence:

Definition 20. *The* **randomness** *of a sensory sequence S is the ratio of the cost of the smallest interpretation of S that satisfies the conditions divided by the cost of the unilluminating interpretation of S of Theorem 4.* △

Compare this definition with the Kolmogorov complexity of a sensory sequence:

Definition 21. *The* **Kolmogorov complexity** *of a sensory sequence S is the length of the smallest theory that provides a unified interpretation of S.*

$$K(S) = \min \left\{ \ cost(\theta) \mid S \sqsubseteq \tau(\theta), unity(\theta) \ \right\}$$

Here cost(θ) is the size of the initial conditions plus the size of the rules, and unity(θ) is true if theory θ satisfies the unity conditions. △

Note that the Kolmogorov complexity of a sequence is an integer value, while the randomness of a sequence is a real value between 0 and 1.

3.7 The computer implementation

The Apperception Engine is our system for solving apperception tasks.[20] 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 22. *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 $(\phi, N_{\rightarrow}, N_{\ni}, N_B)$ where ϕ is a type signature, $N_{\rightarrow}$ is the max number of static rules allowed in R, $N_{\ni}$ 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 $C' \supseteq C$.

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.

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

Algorithm 1: The Apperception Engine algorithm in outline

```
input  : $(S, \phi, C)$, an apperception task
output: $\theta^*$, a unified interpretation of $S$
$(s^*, \theta^*) \leftarrow (max(float), nil)$
foreach template $\chi$ extending $\phi$ 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 $\theta^*$
    end
end
```

3.7.1 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 $(O, P, V, N_{\rightarrow}, N_{\ni}, N_B)$ tuples for that particular T. We cannot, of course, enumerate *all* $(O, P, V, N_{\rightarrow}, N_{\ni}, N_B)$ 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 $(O, P, V, N_{\rightarrow}, N_{\ni}, N_B)$
end
```

In order to enumerate (T, n) pairs, we use a standard diagonalization procedure. See Table 3.1.

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.

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

Table 3.1: 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 $(O, P, V, N_{\rightarrow}, N_{\ni\!\!\!\rightarrow}, N_B)$ to enumerate. We increment n by 100. The entries in the table represent the order in which the (T, n) pairs are visited.

We use `choices` to generate 6-tuples $(O, P, V, N_{\rightarrow}, N_{\ni\!\!\!\rightarrow}, N_B)$ tuples by creating six infinite streams:

1. $\mathcal{S}_O$: an infinite list of finite lists of typed objects
2. $\mathcal{S}_P$: an infinite list of finite lists of typed predicates
3. $\mathcal{S}_V$: an infinite list of finite lists of typed variables
4. $\mathcal{S}_{\rightarrow} = \{0, 1, ...\}$: the number of static rules
5. $\mathcal{S}_{\ni\!\!\!\rightarrow} = \{0, 1, ...\}$: the number of causal rules
6. $\mathcal{S}_B = \{0, 1, ...\}$: 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 $\mathcal{S}_O \times \mathcal{S}_P \times \mathcal{S}_V \times \mathcal{S}_{\rightarrow} \times \mathcal{S}_{\ni\!\!\!\rightarrow} \times \mathcal{S}_B$.

Example 11. Recall the apperception problem from Example 1. There are two sensors a and b, and each sensor can be *on* or *off*. The sensory sequence is $S_{1:7}$ where:

$$\begin{array}{ll}
S_1 = \{on(a), on(b)\} & S_2 = \{off(a), on(b)\} \\
S_3 = \{on(a), off(b)\} & S_4 = \{on(a), on(b)\} \\
S_5 = \{off(a), on(b)\} & S_6 = \{on(a), off(b)\} \\
S_7 = \{\} &
\end{array}$$

We shall start with an initial template $\chi_0 = (\phi = (T, O, P, V), N_{\rightarrow}, N_{\ni\!\!\!\rightarrow}, N_B)$, where:

$$\begin{aligned}
T &= \{sensor, grid\} \\
O &= \{a{:}sensor, b{:}sensor, g{:}grid\} \\
P &= \{on(sensor), off(sensor), part(sensor, grid)\} \\
V &= \{X{:}sensor, Y{:}sensor\} \\
N_{\rightarrow} &= 1 \\
N_{\ni\!\!\!\rightarrow} &= 3 \\
N_B &= 2
\end{aligned}$$

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 :

$$\begin{array}{rcl}
\Delta\chi_1 & = & (\emptyset, \emptyset, \emptyset, \{V_1{:}\mathit{sensor}\}, 0, 0, 0) \\
\Delta\chi_2 & = & (\emptyset, \emptyset, \emptyset, \emptyset, 0, 0, 1) \\
\Delta\chi_3 & = & (\emptyset, \emptyset, \{p_1(\mathit{sensor})\}, \emptyset, 0, 0, 0) \\
\Delta\chi_4 & = & (\emptyset, \emptyset, \emptyset, \{V_1{:}\mathit{sensor}\}, 0, 0, 1) \\
\Delta\chi_5 & = & (\emptyset, \emptyset, \emptyset, \emptyset, 0, 0, 2) \\
\Delta\chi_6 & = & (\emptyset, \{o_1{:}\mathit{sensor}\}, \emptyset, \emptyset, 0, 0, 0) \\
\Delta\chi_7 & = & (\emptyset, \emptyset, \{p_1(\mathit{sensor})\}, \emptyset, 0, 0, 1) \\
\Delta\chi_8 & = & (\emptyset, \emptyset, \emptyset, \{V_1{:}\mathit{sensor}\}, 0, 0, 2) \\
\Delta\chi_9 & = & (\emptyset, \emptyset, \emptyset, \emptyset, 0, 0, 2) \\
\Delta\chi_{10} & = & (\emptyset, \emptyset, \{p_1(\mathit{sensor})\}, \{V_1{:}\mathit{sensor}\}, 0, 0, 0) \\
& ... &
\end{array}$$

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 resource 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):

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

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

$$I \quad = \quad \left\{ \begin{array}{lll} p_1(a) & p_2(b) & \mathit{on}(a) \\ \mathit{part}(a, g) & \mathit{part}(b, g) & \end{array} \right\}$$

$$R \quad = \quad \left\{ \begin{array}{l} p_2(S) \rightarrow \mathit{on}(S) \\ p_2(S) \ni\!\!- p_1(S) \\ p_1(S) \wedge \mathit{on}(S) \ni\!\!- \mathit{off}(S) \\ \mathit{off}(S) \wedge p_1(S) \ni\!\!- p_2(S) \end{array} \right\}$$

$$C = \left\{ \begin{array}{l} \forall X : \mathit{sensor},\ p_1(X) \oplus p_2(X) \\ \forall X : \mathit{sensor},\ \exists! Y : \mathit{grid},\ \mathit{part}(X, Y) \end{array} \right\}$$

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):

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

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

$$I = \left\{ \begin{array}{lll} \mathit{off}(o_1) & \mathit{on}(a) & \\ r_1(a, o_1) & r_1(b, a) & r_1(o_1, b) \\ \mathit{part}(a, \mathit{grid}) & \mathit{part}(b, \mathit{grid}) & \mathit{part}(o_1, \mathit{grid}) \end{array} \right\}$$

$$R = \left\{ \begin{array}{l} \mathit{off}(S) \wedge r_1(S, S2) \rightarrow \mathit{on}(S2) \\ \mathit{off}(S2) \wedge r_1(S, S2) \wedge \mathit{on}(S) \ni\!\!\!\rightarrow \mathit{off}(S) \end{array} \right\}$$

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

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

◁

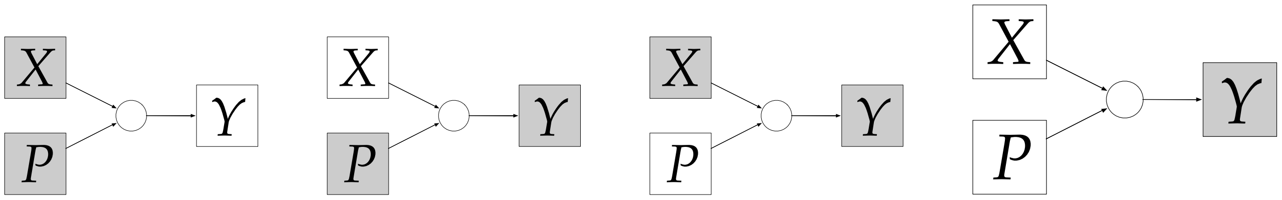
3.7.2 Finding the best theory from a template

The most complex part of Algorithm 1 is:

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

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. See Figure 3.2. Here, $X \subseteq \mathcal{G}$ is a set of facts (ground atoms), $P : \mathcal{G} \rightarrow \mathcal{G}$ is a procedure for generating the



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

Figure 3.2: Varieties of inference. Shaded elements are given, and unshaded elements are generated.

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.[21]

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 2.3 for the basic strategy, and Section 3.7.3 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 [CFG+12] 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` [GKKS14]) 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 3.7.3, we explain how Algorithm 2 is implemented in ASP. In Section 3.7.4, we evaluate the computational complexity. In Section 3.7.5, we describe the various optimisations used to prune the search. In Section 3.7.6, we compare with ILASP, a state of the art ILP system.

3.7.3 The Datalog$^{\ni}$ interpreter

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

[21] At a high level, our system is similar to XHAIL [Ray09]. 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 Kantian 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: $\theta$, the simplest unified interpretation of $S$ that conforms to $\chi$
$\pi_S \leftarrow$ gen_input($S$)
$\pi_I \leftarrow$ gen_inits($\phi$)
$\pi_R \leftarrow$ gen_rules($\phi, N_{\rightarrow}, N_{\ni}, N_B$)
$\pi_C \leftarrow$ gen_constraints($\phi, C$)
$\Pi \leftarrow \pi_\tau \cup \pi_u \cup \pi_m \cup \pi_S \cup \pi_I \cup \pi_R \cup \pi_C$
$A \leftarrow$ clingo($\Pi$)
if satisfiable($A$) then
    $\theta \leftarrow$ extract($A$)
    return $\theta$
end
return nil
```

represented by a constant `c_p`. An unground atom $p(X)$ is represented by a term `s(c_p, var_x)`. A rule is represented by a set of unground atoms for the body, and a single unground 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)).
```

Here, `c_p`, `c_q`, and `c_r` are ASP constants representing the Datalog$^{\ni}$ predicates p, q, and r, while `var_x` and `var_y` are ASP constants representing the Datalog$^{\ni}$ variables X and 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:

```
is_var_atom(atom(c_on, var_s)).
is_var_atom(atom(c_off, var_s)).
is_var_atom(atom(c_r, var_c, var_c)).
is_var_atom(atom(c_r, var_c, var_c2)).
is_var_atom(atom(c_r, var_c2, var_c))).
is_var_atom(atom(c_r, var_c2, var_c2)).
...
```

Similarly, we construct ASP terms that represent every well-typed ground atom in G_ϕ.

We also construct ASP atoms that represent every substitution in Σ_ϕ. For each substitution σ and each X/k in σ, we add an atom of the form $\mathsf{subs}(\sigma, X, k)$. For example, if $\sigma_{17} = \{X/a, Y/b\}$, then we add:

```
subs(subs_17, var_x, obj_a).
subs(subs_17, var_y, obj_b).
```

To represent that the result of applying substitution σ to unground atom α is ground atom α', we use the `ground_atom` predicate:

```
ground_atom(s(C, V), s(C, Obj), Subs) :-
    is_var_fluent(s(C, V)),
    subs(Subs, V, Obj).

ground_atom(s2(C, V, V2), s2(C, Obj, Obj2), Subs) :-
    is_var_fluent(s2(C, V, V2)),
    subs(Subs, V, Obj),
    subs(Subs, V2, Obj2).
```

The initial conditions $I \subseteq G_\phi$ are represented by the `init` predicate.

Rules and constraints are implemented differently: while rules are interpreted, constraints are compiled directly into ASP constraints. The system generates all possible constraints that are compatible with the type signature, and translates each such constraint into a set of ASP clauses. For example, if k_1 is the constraint $\forall X{:}cell, on(X) \oplus off(X)$, then k_1 is represented as:

```
:-  holds(s(c_on, X), T),
    holds(s(c_off, X), T),
    use_constraint(k_1).

:-  isa(X, t_cell),
    is_time(T),
    not holds(s(c_on, X), T),
    not holds(s(c_off, X), T),
    use_constraint(k_1).

incompossible(s(c_on, X), s(c_off, X)) :-
    isa(X, t_cell),
    use_constraint(k_1).
```

Here, the flag `use_constraint(k_1)` is used to check whether or not we wish to include this particular constraint in C. The solver chooses which particular constraints to use, just as it gets to choose the initial atoms and the update rules. For example, if there are four unary predicates $p_1, ..., p_4$, there are various possible sets of constraints that each satisfy conceptual unity:

$$\{p_1(X) \oplus p_2(X), p_3(X) \oplus p_4(X)\}$$
$$\{p_1(X) \oplus p_3(X), p_2(X) \oplus p_4(X)\}$$
$$\{p_1(X) \oplus p_4(X), p_2(X) \oplus p_3(X)\}$$
$$\{p_1(X) \oplus p_2(X) \oplus p_3(X) \oplus p_4(X)\}$$

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

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

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

-holds(S, T) :-
    holds(S2, T),
    incompossible(S, S2).

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

% arrow update
holds(GA, T) :-
    rule_head_static(R, VA),
    eval_body(R, Subs, T),
    ground_atom(VA, GA, Subs).
```

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.

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

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

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

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

The ASP program π_m minimizes the cost of the theory θ (see Definition 17) by using weak constraints [CFG+12]:

$$:\sim \texttt{rule\_body(R, A)}.\ [1@1, \texttt{R}, \texttt{A}]$$

$$:\sim \texttt{rule\_head\_static(R, A)}.\ [1@1, \texttt{R}, \texttt{A}]$$

$$:\sim \texttt{rule\_head\_causes(R, A)}.\ [1@1, \texttt{R}, \texttt{A}]$$

$$:\sim \texttt{init(A)}.\ [1@1, \texttt{A}]$$

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:

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

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

```
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 $N_{\supset\!\!-}$ and $N_{\rightarrow}$ in the template.

To allow the solver to choose which constraints to include in C, we generate all possible sets of constraints and add a cardinality constraint that exactly one such set is active for each type. For example, if there are four unary predicates $p_1(t), ...p_4(t)$ of type t, then there are various ways of collecting them into xor groups:

$$\{p_1(X) \oplus p_2(X), p_3(X) \oplus p_4(X)\}$$
$$\{p_1(X) \oplus p_3(X), p_2(X) \oplus p_4(X)\}$$
$$\{p_1(X) \oplus p_4(X), p_2(X) \oplus p_3(X)\}$$
$$\{p_1(X) \oplus p_2(X) \oplus p_3(X) \oplus p_4(X)\}$$

We remove sets that are equivalent up to renaming, so are left with:

$$\{p_1(X) \oplus p_2(X), p_3(X) \oplus p_4(X)\}$$
$$\{p_1(X) \oplus p_2(X) \oplus p_3(X) \oplus p_4(X)\}$$

Let us call these two xor sets k_1 and k_2. We use flags `use_constraint(k_1)` and `use_constraint(k_2)` to indicate which xor set to use, and add the cardinality constraint:

```
1 { use_constraint(k_1), use_constraint(k_2) } 1.
```

3.7.4 Complexity and optimisation

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 $\Sigma^P_{i+1} = NP^{\Sigma^P_i}$ be the class of problems that can be solved in polynomial time by a non-deterministic Turing machine with a Σ^P_i oracle. If Π is a Datalog program, and A and B are sets of ground atoms, then:

- the **data complexity** is the complexity of testing whether $\Pi \cup A \models B$, as a function of A and B, when Π is fixed
- the **program complexity** (also known as "expression complexity") is the complexity of testing whether $\Pi \cup A \models B$, as a function of Π and B, when A is fixed

Datalog has polynomial time data complexity but exponential time program complexity: deciding whether a ground atom is in the least Herbrand model of a Datalog program is EXPTIME-complete. The reason for this complexity is because the number of ground instances of a clause is an exponential function of the number of variables in the clause. Finding a solution to an ASP program is in NP [BED94, DEGV01], while finding an optimal solution to an ASP program with weak constraints is in Σ^P_2 [BNT03, GKS11].

Since deciding whether a non-disjunctive ASP program has a solution is in NP [BED94, DEGV01], 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^P_2 [BNT03, GKS11], our ASP encoding shows that finding the lowest cost θ is in Σ^P_2.

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[22]:

$$\begin{aligned} |G_\phi| &\leq |P| \cdot |O|^2 \\ |U_\phi| &\leq |P| \cdot |V|^2 \end{aligned}$$

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

Predicate	**Max # ground atoms**
`rule_body(R, VA)`	$\|U_\phi\| \cdot (N_\rightarrow + N_\ni)$
`rule_head_static(R, VA)`	$\|U_\phi\| \cdot N_\rightarrow$
`rule_head_causes(R, VA)`	$\|U_\phi\| \cdot N_\ni$
`holds(GA, T)`	$\|G_\phi\| \cdot t$
`subs(Subs, Var, Obj)`	$\|\Sigma_\phi\| \cdot \|V\|$
`ground_atom(VA, GA, Subs)`	$\|\Sigma_\phi\| \cdot \|U_\phi\|$
`eval_atom(VA, Subs, T)`	$\|\Sigma_\phi\| \cdot \|U_\phi\| \cdot t$
`eval_body(R, Subs, T)`	$\|\Sigma_\phi\| \cdot (N_\rightarrow + N_\ni) \cdot t$

Table 3.2: The number of ground atoms in the ASP encoding of Algorithm 2.

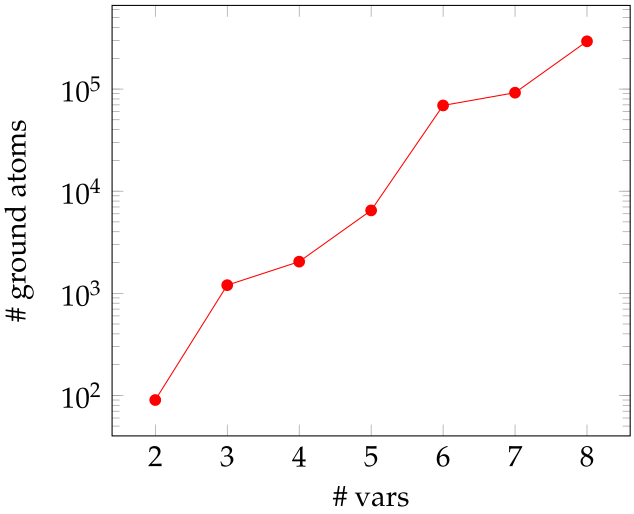


Figure 3.3: How # ground atoms grows (log-scale) as we increase # vars

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:

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

Table 3.2 shows the number of ground atoms for the most expensive predicates in the ASP encoding. Here, `R` ranges over rules, `VA` over unground atoms, `Subs` over substitutions, `Var` over variables, `Obj` over objects, `GA` over ground atoms, and `T` over time-steps $1..t$. The predicates with the largest number of groundings are those that feature a variable of type `Subs`.

From Table 3.2, we can see that the number of ground atoms is a linear function of the number of time-steps, a quadratic function of the number of objects, and an exponential function of the number of variables. Figure 3.3 shows how the number of ground atoms increases exponentially as a function of the number of variables. Here, we plot the number of ground atoms with predicate `eval_atom` as a function of the number of variables, when interpreting *Twinkle Twinkle Little Star* from the music domain.

In terms of the number of ground clauses, the most expensive rule in the ASP encoding is:

```
holds(GA, T) :-
    rule_head_static(R, VA),
    eval_body(R, Subs, T),
    ground_atom(VA, GA, Subs).
```

This rule generates $|\Sigma_\phi| \cdot |U_\phi| \cdot N_\rightarrow \cdot t$ ground instances, each containing 4 atoms. There is a similar number of ground clauses for the causal rules.

The frame axiom is less expensive.

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

This generates $|G_\phi| \cdot t$ ground clauses. There are only another $|G_\phi|$ ground clauses for the clause `holds(A, 1) :- init(A).`

Another clause that generates a large number of ground clauses is:

```
eval_body(R, Subs, T) :-
    is_rule(R),
    is_subs(Subs),
    is_time(T),
    eval_atom(V, Subs, T) : rule_body(R, V).
```

This generates $|\Sigma_\phi| \cdot (N_\rightarrow + N_\ni) \cdot t$ ground clauses, each of which contains $|U_\phi| + 4$ atoms.

Table 3.3 shows the number of ground clauses for the three most expensive clauses. The total number of ground atoms for the three most expensive clauses is approximately $5 \cdot |\Sigma_\phi| \cdot (N_\rightarrow + N_\ni) \cdot |U_\phi| \cdot t$.

3.7.5 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.

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 [MCO19].

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).`
	$\lvert\Sigma_\phi\rvert \cdot (N_\rightarrow + N_\ni) \cdot t$	$\lvert U_\phi\rvert + 4$

Table 3.3: The number of ground clauses in the ASP encoding of Algorithm 2.

Symmetry breaking

We use symmetry breaking to remove candidates that are equivalent. For example, the following two rules are equivalent up to variable renaming:

$$
\begin{aligned}
p(X, Y) &\rightarrow q(X) \\
p(Y, X) &\rightarrow q(Y)
\end{aligned}
$$

We prune the second rule by using a strict partial order on variables: $X < Y$ if X and Y are of the same type (if $\kappa_V(X) = \kappa_V(Y)$) and X is lexicographically before Y. Now we prune rules where there is a variable X in the body B, where $Y < X$ and it is not the case that either:

- there is an atom $p(Y)$ in B, or:
- there is an atom $p(Y, X)$ in B

The second form of symmetry breaking is to prune collections of rules that are equivalent up to reordering. Because we represent rules using rule-identifiers, there are multiple rule-sets that are equivalent but represented distinctly. For example, the sets R_1 and R_2 are obviously equivalent:

$$R_1 = \begin{cases} r_1 : p(X) \rightarrow q(X) \\ r_2 : q(X) \rightarrow r(X) \end{cases}$$

$$R_2 = \begin{cases} r_1 : q(X) \rightarrow r(X) \\ r_2 : p(X) \rightarrow q(X) \end{cases}$$

We define a strict total ordering $<$ on unground atoms in U_ϕ, and use this to prune duplicates. We disallow any rule-set that contains two rules $r_i : B \rightarrow \alpha$ and $r_j : B' \rightarrow \alpha'$ where $i < j$ and there exists an atom $\beta' \in B'$ that is $<$ every atom $\beta \in B$. Assuming that $p(X) < q(X) < r(X)$, this rules out R_2 in the example above.

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) [GKKS12]. 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).
```

Replacing the meta-interpreter of constraints with compiled clauses

In an earlier implementation, the *xor* and ∃! constraints were evaluated using a meta-interpreter (as the rules are). We made a significant optimization (approximately 10x speed-up) by replacing this part of the meta-interpreter with compiled clauses. Here, we used the same approach as ASPAL [CRL12] and ILASP [LRB14] to compile the constraint clauses directly into ASP. Removing the overhead of the interpreter gave us a large speedup here. But note that we did not replace the $\tau(\theta)$ interpreter with a set of compiled clauses. The reason it is worth compiling the constraint evaluator – but it is not worth compiling the update rules – is because of the relatively small number of possible constraints. The number of possible update rules grows quickly with the number of unground atoms, so compiling each possible update rule into an ASP clause is prohibitively expensive. See Section 3.7.6 for an empirical evaluation comparing the grounding sizes of the programs when meta-interpreting the update rules versus compiling the update rules.

3.7.6 A comparison with ILASP

In order to assess the efficiency of our system, we compared it to ILASP [LRB14, LRB15, LRB16, LRB18a], a state of the art Inductive Logic Programming algorithm[23].

We compared against ILASP rather than Metagol (another state-of-the-art inductive logic programming system [MLT15, CM18]) for three reasons. First, since ILASP also uses ASP we can compare the grounding size of our program with ILASP and get a fair apples-for-apples comparison. Second, ILASP achieves slightly higher performance (it achieved slightly better results than Metagol in the Inductive General Game Playing task suite [CEL19], getting 40% correct as opposed to Metagol's 36%). Third, Metagol requires positive examples of the target program in order to search, while the apperception framework does not provide any positive examples - we are simply given a trace. ILASP, by contrast, is a very general framework that is able to induce programs satisfying constraints without the need to specify positive examples.

Unlike traditional ILP systems that learn definite logic programs, ILASP learns *answer set programs*[24]. 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 [LRB15].

A **Learning from Answer Sets** task is a tuple (B, S_M, E^+, E^-) where B is a background program, S_M is the hypothesis space (a set of ASP clauses), E^+ is a set of positive examples (represented as partial interpretations) and E^- is a set of negative examples (also represented as partial interpretations). A particular ASP program $H \subseteq S_M$ is an inductive solution of the task if:

[23] Strictly speaking, ILASP is a family of algorithms, rather than a single algorithm. We used ILASP2 [LRB15] in this evaluation. I am very grateful to Mark Law for all his help in this comparative evaluation.

[24] 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 common-sense reasoning [Mue14, GK14].

1. for all e in E^+, there exists an answer set $A \in AS(B \cup H)$ such that A satisfies e
2. for all e in E^-, there is no answer set $A \in AS(B \cup H)$ such that A satisfies e

Here, $AS(B \cup H)$ is the set of answer sets of the ASP program combining the sets of clauses B and H.

A discrete apperception task (S, ϕ, C) can be expressed in this framework as $(B, H, \{e\}, \{\})$, where:

- B combines the Kantian conditions (represented as an ASP program) with the sequence sensory S and the constraints C
- H is a set of hypothesis clauses generated from the type signature ϕ
- e is a single positive empty example[25]

Note that the Kantian unity conditions of Section 3.3 are encoded as ASP constraints and provided as background knowledge to ILASP. The frame axiom (that allows atoms to persist unless there is an incompossible atom, see Definition 9) is also provided as background knowledge. Thus, the comparison between ILASP and the Apperception Engine is fair, as both systems are provided with the same inductive bias.

Because of the generality of the Learning from Answer Sets framework, we can express an apperception task within it. Of course, since ILASP was not designed specifically with this task in mind, there is no reason it would be as efficient as a program synthesis technique which was targeted specifically at apperception tasks.

In ILASP, the set of H of hypothesis clauses is defined by a set of **mode declarations**. A mode declaration specifies the sort of atoms that are allowed in the heads and bodies of clauses. For example, the declaration `#modeh(p(var(t1)))` states that an atom $p(X)$ can appear in the head of a clause, where X is some variable of type `t1`. The declaration `#modeb(2, r(var(t1), const(t2)))` states that an atom $f(X, k)$ can appear in the body of a clause, where k is a constant of type `t2`. The parameter 2 in the `modeb` declaration specifies that an atom of this form can appear at most two times in the body of any rule.

ILASP uses a similar approach to ASPAL [CRL10b, CRL11b] to generate hypothesis clauses. For example, given the mode declarations:

```
#modeh(p(var(t1))).
#modeb(1, q(var(t1), var(t1))).
```

[25] Note that we are using ILASP in a highly restricted way, with a single positive empty example and no negative examples. ILASP is a general framework for learning from positive and negative examples. It can solve tasks whose satisfiability problem is Σ_2^P-complete [LRB18a]. But if we restrict to positive examples only, it can only solve tasks whose satisfiability problem is NP-complete.

the following clauses are generated[26]:

```
p(X) :- q(X, X), in_h(1).
p(X) :- q(X, Y), in_h(2).
p(X) :- q(Y, X), in_h(3).
```

The `in_h` atoms are used to control which clauses are in the hypothesis *H* and which are not. Turning on and off the `in_h` atoms controls which atoms are included, expressed using an ASP choice rule:

```
0 { in_h(1), in_h(2), in_h(3) } 3.
```

To generate an interpretation for an apperception task, we need to generate a set of initial atoms, a set of static rules and a set of causal rules. We generate mode declarations for each type. Each potential initial atom `X` is turned into a `modeh` declaration `#modeh(init(X))`. Static rules and causal rules are generated by `modeb` and `modeh` declarations. For example:

```
#modeh(causes(s(c_on, var(t_object)), s(c_off, var(t_object)), var(t_time))).
#modeh(causes(s(c_off, var(t_object)), s(c_on, var(t_object)), var(t_time))).
#modeh(static(holds(s(c_on, var(t_object)), var(t_time)), var(t_time))).
#modeh(static(holds(s(c_off, var(t_object)), var(t_time)), var(t_time))).
#modeb(1, holds(s(const(t_pred_fluid_1), var(t_object)), var(t_time)), (positive)).
#constant(t_pred_fluid_1, c_off).
#constant(t_pred_fluid_1, c_on).
```

Evaluation

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 (by `gringo`) into propositional clauses that are then solved (by `clasp`).[27] The grounding size determines the memory usage and is strongly correlated with solution time.

We took a sample ECA, Rule 245, and looked at the grounding size as the number of cells increased from 2 to 11. The results are in Table 3.4 and Figure 3.4.

[26] This is a simplification for expository purposes; the actual clauses generated have additional complexity that is not important for this discussion.

[27] These two programs are part of the Potassco ASP toolset: https://potassco.org/clingo/

# cells	Our System	ILASP
2	0.6	60.7
3	1.8	173.7
4	4.0	376.0
5	7.8	692.8
6	13.4	1149.6
7	21.3	1771.9
8	31.7	2585.1
9	45.1	3103.4
10	61.8	4902.6
11	82.6	6464.1

Table 3.4: Like-for-like comparison between our system and ILASP. We compare the size of the ground programs (in megabytes) generated as the number of cells in the ECA increases from 2 to 11.

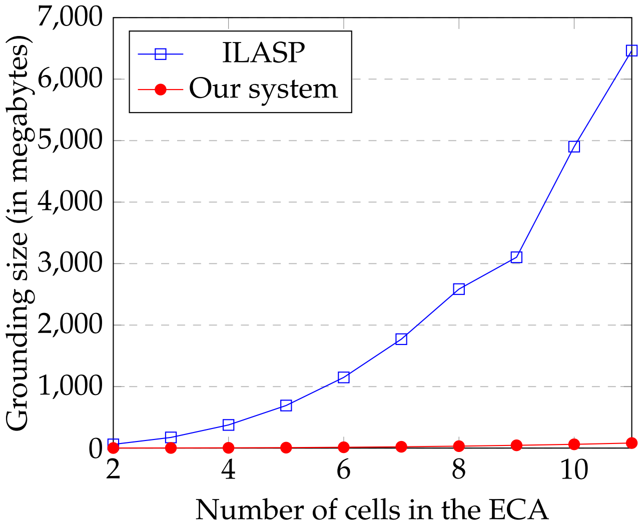


Figure 3.4: Comparing our system and ILASP w.r.t. grounding size

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).
```

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

The most expensive clauses in our encoding are analysed in Table 3.3. Recall from Section 3.7.4 that the total number of atoms in the ground clauses is approximately $5 \cdot |\Sigma_\phi| \cdot (N_\rightarrow + N_\gtrdot) \cdot |U_\phi| \cdot t$.

To compare this with ILASP, let us set $N_B = 4$ (which is representative). Then ILASP generates ground clauses with $5 \cdot |U_\phi|^{N_B+1} \cdot |\Sigma_\phi| \cdot t$ ground atoms while our system generates clauses with $5 \cdot |\Sigma_\phi| \cdot (N_\rightarrow + N_\gtrdot) \cdot |U_\phi| \cdot t$ ground atoms.

The reason, then, why our system has such lower grounding sizes than ILASP is because $(N_\rightarrow + N_\gtrdot) << |U_\phi|^{N_B}$. Intuitively, the key difference[28] is that ILASP considers *every possible* subset of the hypothesis space, while our system (by restricting to at most $N_\rightarrow + N_\gtrdot$ rules) only considers *subsets of length at most* $N_\rightarrow + N_\gtrdot$.

[28] Another difference that helps to explain the difference in grounding sizes is that ILASP's grounding multiplies the number of potential rule bodies by the number of potential heads, while the Apperception Engine splits a rule into two separate parts – one for the head, and one for the body – and therefore adds the number of rule bodies to the number of heads, rather than multiplying them. I am grateful to Mark Law for this point.

Chapter 4

Experiments

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 IQ 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). Although all the tasks are data-sparse and designed to be easy for humans but hard for machines, in other respects the domains were chosen to maximize diversity: the various domains involve different sensory modalities, and some sensors provide binary discriminators while others are multi-class.

4.1 Experimental setup

We implemented the Apperception Engine in Haskell and ASP. We used `clingo` [GKKS14] to solve the ASP programs generated by our system. We ran all experiments with a time-limit of 4 hours. We ran `clingo` in "vanilla" mode, and did not experiment with the various command-line options for optimization, although it is possible we could achieve significant speedup with judicious use of these parameters.

We ran all experiments on HTCondor, a high-throughput computing framework for distributed parallelization of computationally intensive tasks [TTL05]. Note that our experimental setup is *almost* fully deterministic: the procedure for generating an ASP program from a template and a sensory sequence is deterministic; our ASP solver `clingo` is also deterministic (in that it will always output the same optimal answer set when given the same input program). However, because we terminate after a time-limit, if two machines have different processing speeds, they may have found different local optima after the 4 hour time-limit.

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 %

Table 4.1: 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.

For each of the five domains, we provided an infinite sequence of templates (implemented in Haskell as an infinite list). Each template sequence is a form of declarative bias [DR12]. It is important to note that the domain-specific template sequence is not essential to the Apperception Engine, as our system can also operate using the domain-*independent* template iterator described in Section 3.7.1. Every template in the template sequence will eventually be found by the domain-independent iterator. However, in practice, the Apperception Engine will find results in a more timely manner when it is given a domain-specific template sequence rather than the domain-independent templates sequence[1].

4.2 Results

Our experiments (on the prediction task) are summarised in Table 4.1. 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.[2] It does not score any points for predicting most of the hidden values correctly. As can be seen from Table 4.1, our system is able to achieve good accuracy across all five domains.

In Table 4.2, we display Cohen's kappa coefficient [Coh60] for the five domains. If a is the accuracy and r is the chance of randomly agreeing with the actual classification, then the kappa metric $\kappa = \frac{a-r}{1-r}$. Since our accuracy metric for a single task is rather exacting (since the model is accurate only if *every* hidden sensor value is predicted correctly), the chance r of random accuracy is very low. For example, in the ECA domain with 11 cells, the chance of randomly predicting correctly is 2^{-11}. Similarly, in the

[1] This is analogous to the situation in Metagol, which uses *metarules* as a form of declarative bias. As shown in [CM15], there is a pair of highly general metarules which are sufficient to entail all metarules of a certain broad class. However, in practice, it is significantly more efficient to use a domain-specific set of metarules, rather than the very general pair of metarules [Cro17, CT18].

[2] 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.

Domain	**Accuracy** (a)	**Random agreement** (r)	**Kappa metric** (κ)
ECA	0.973%	0.00048	0.972
Rhythm & music	0.733%	0.00001	0.732
Seek Whence	0.767%	0.16666	0.720
Multi-modal binding	0.850%	0.00003	0.849
Occlusion	0.900 %	0.03846	0.896

Table 4.2: Cohen's kappa coefficient for the five domains. Note that the chance of random agreement (r) is very low because we define accuracy as correctly predicting *every* sensor reading. When r is very low, the κ metric closely tracks the accuracy.

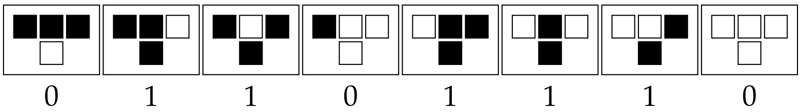


Figure 4.1: Updates for ECA rule 110. The top row shows the context: the target cell together with its left and right neighbour. The bottom row shows the new value of the target cell given the context. A cell is black if it is on and white if it is off.

music domain, if there are 8 sensors and each can have 4 loudness levels, then the chance of randomly predicting correctly is 4^{-8}. Because the chance of random accuracy is so low, the kappa metric closely tracks the accuracy.

4.2.1 Elementary cellular automata

An Elementary Cellular Automaton (ECA) [Wol83, Coo04] 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.

Figure 4.1 shows one set of ECA update rules[3]. 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. Figure 4.2 shows the state sequence for Rule 110 above from one initial starting state of length 11.

[3] This particular set of update rules is known as Rule 110. Here, 110 is the decimal representation of the binary 01101110

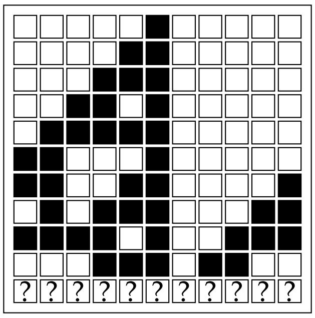


Figure 4.2: One trajectory for Rule 110. Each row represents the state of the ECA at one time-step. In this prediction task, the bottom row (representing the final time-step) is held out.

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 Figure 4.2, the sensory sequence is $(S_1, ..., S_{10})$ where:

$$\begin{aligned}
S_1 &= \{\mathit{off}(c_1), \mathit{off}(c_2), \mathit{off}(c_3), \mathit{off}(c_4), \mathit{off}(c_5), \mathit{on}(c_6), \mathit{off}(c_7), \mathit{off}(c_8), \mathit{off}(c_{(}), \mathit{off}(c_{10}), \mathit{off}(c_{11})\} \\
S_2 &= \{\mathit{off}(c_1), \mathit{off}(c_2), \mathit{off}(c_3), \mathit{off}(c_4), \mathit{on}(c_5), \mathit{on}(c_6), \mathit{off}(c_7), \mathit{off}(c_8), \mathit{off}(c_{(}), \mathit{off}(c_{10}), \mathit{off}(c_{11})\} \\
S_3 &= \{\mathit{off}(c_1), \mathit{off}(c_2), \mathit{off}(c_3), \mathit{on}(c_4), \mathit{on}(c_5), \mathit{on}(c_6), \mathit{off}(c_7), \mathit{off}(c_8), \mathit{off}(c_{(}), \mathit{off}(c_{10}), \mathit{off}(c_{11})\} \\
S_4 &= \{\mathit{off}(c_1), \mathit{off}(c_2), \mathit{on}(c_3), \mathit{on}(c_4), \mathit{off}(c_5), \mathit{on}(c_6), \mathit{off}(c_7), \mathit{off}(c_8), \mathit{off}(c_{(}), \mathit{off}(c_{10}), \mathit{off}(c_{11})\} \\
S_5 &= \{\mathit{off}(c_1), \mathit{on}(c_2), \mathit{on}(c_3), \mathit{on}(c_4), \mathit{on}(c_5), \mathit{on}(c_6), \mathit{off}(c_7), \mathit{off}(c_8), \mathit{off}(c_{(}), \mathit{off}(c_{10}), \mathit{off}(c_{11})\} \\
&...
\end{aligned}$$

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.

We provide a sequence of templates $(\chi_1, \chi_2, ...)$ for the ECA domain. Our initial template χ_1 is:

$$\begin{aligned}
\phi &= \left\{ \begin{array}{l} T = \{\mathit{cell}\} \\ O = \{c_1{:}\mathit{cell}, c_2{:}\mathit{cell}, ..., c_{11}{:}\mathit{cell}\} \\ P = \{\mathit{on}(\mathit{cell}), \mathit{off}(\mathit{cell}), r(\mathit{cell}, \mathit{cell})\} \\ V = \{X{:}\mathit{cell}, Y{:}\mathit{cell}, Z{:}\mathit{cell}\} \end{array} \right\} \\
N_{\rightarrow} &= 0 \\
N_{\ni} &= 2 \\
N_B &= 4
\end{aligned}$$

The signature includes a binary relation r on cells. This could be used as a spatial relation between neighbouring cells. But we do not, of course, insist on this particular interpretation – the system is

update rule, as shown in Figure 4.1. This rule has been shown to be Turing-complete [Coo04].

free to interpret the r relation any way it chooses. The other templates $\chi_2, \chi_3, ...$ are generated by increasing the number of static rules, causal rules, and body atoms in χ_1.

We applied our interpretation learning method to all $2^{2^3} = 256$ ECA rule-sets. For Rule 110 (see Figure 4.2 above), it found the following interpretation (ϕ, I, R, C), where:

$$
\begin{aligned}
I &= \left\{ \begin{array}{llll}
\mathit{off}(c_1) & \mathit{off}(c_2) & \mathit{off}(c_3) & \mathit{off}(c_4) \\
\mathit{off}(c_5) & \mathit{on}(c_6) & \mathit{off}(c_7) & \mathit{off}(c_8) \\
\mathit{off}(c_9) & \mathit{off}(c_{10}) & \mathit{off}(c_{11}) & r(c_1, c_{11}) \\
r(c_2, c_1) & r(c_3, c_2) & r(c_4, c_3) & r(c_5, c_4) \\
r(c_6, c_5) & r(c_7, c_6) & r(c_8, c_7) & r(c_9, c_8) \\
r(c_{10}, c_9) & r(c_{11}, c_{10}) & &
\end{array} \right\} \\
R &= \left\{ \begin{array}{l}
r(X, Y) \wedge on(X) \wedge \mathit{off}(Y) \ni\!\!- on(Y) \\
r(X, Y) \wedge r(Y, Z) \wedge on(X) \wedge on(Z) \wedge on(Y) \ni\!\!- \mathit{off}(Y)
\end{array} \right\} \\
C &= \left\{ \begin{array}{l}
\forall X{:}\mathit{cell},\ on(X) \oplus \mathit{off}(X) \\
\forall X{:}\mathit{cell},\ \exists! Y\ r(X, Y)
\end{array} \right\}
\end{aligned}
$$

The two update rules are a very compact representation of the 8 ECA updates in Figure 4.1: the first rule states that if the right neighbour is on, then the target cell switches from off to on, while the second rule states that if all three cells are on, then the target cell switches from on to off. 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.

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.

The complexities of the interpretations are shown in Table 4.3. Here, for a sample of ECA rules, we show the number of static rules, the number of causal rules, the total number of atoms in the body of all rules, the number of initial atoms, the total number of clauses (total number of static rules, causal rules, and initial atoms), and the total complexity of the interpretation. It is this final value that is minimized during search. Note that the number of initial atoms is always 22 for all ECA tasks. This is because there are 11 cells and each cell needs an initial *on* or *off*, and also each cell X needs a right-neighbour cell (a Y such that $r(X, Y)$ to satisfy the $\exists!$ constraint on the r relation). So we require 11 + 11 = 22 initial atoms.

ECA rule	**# static rules**	**# cause rules**	**# body atoms**	**# inits**	**# clauses**	**cost**
Rule #0	0	1	0	22	23	24
Rule # 143	0	3	8	22	26	36
Rule #11	0	4	9	22	26	39
Rule #110	0	4	10	22	26	40
Rule # 167	0	4	13	22	26	43
Rule # 150	0	4	16	22	26	46

Table 4.3: The complexity of the interpretations found for ECA prediction tasks

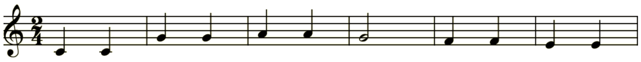


Figure 4.3: Twinkle Twinkle Little Star tune

4.2.2 Drum rhythms and nursery tunes

We also tested our system on simple auditory perception tasks. 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* tune generates the following sensor readings (assuming 8 time-steps for a bar):

$$
\begin{aligned}
S_1 &= \{v(s_c, 3), v(s_d, 0), v(s_e, 0), v(s_f, 0), v(s_g, 0), v(s_a, 0), v(s_b, 0), v(s_{c*}, 0)\} \\
S_2 &= \{v(s_c, 2), v(s_d, 0), v(s_e, 0), v(s_f, 0), v(s_g, 0), v(s_a, 0), v(s_b, 0), v(s_{c*}, 0)\} \\
S_3 &= \{v(s_c, 3), v(s_d, 0), v(s_e, 0), v(s_f, 0), v(s_g, 0), v(s_a, 0), v(s_b, 0), v(s_{c*}, 0)\} \\
S_4 &= \{v(s_c, 2), v(s_d, 0), v(s_e, 0), v(s_f, 0), v(s_g, 0), v(s_a, 0), v(s_b, 0), v(s_{c*}, 0)\} \\
S_5 &= \{v(s_c, 1), v(s_d, 0), v(s_e, 0), v(s_f, 0), v(s_g, 3), v(s_a, 0), v(s_b, 0), v(s_{c*}, 0)\} \\
S_6 &= \{v(s_c, 0), v(s_d, 0), v(s_e, 0), v(s_f, 0), v(s_g, 2), v(s_a, 0), v(s_b, 0), v(s_{c*}, 0)\} \\
&\ldots
\end{aligned}
$$

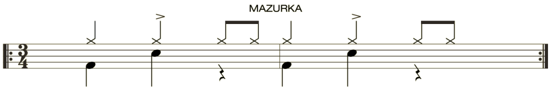


Figure 4.4: Mazurka rhythm

We provided the following initial template χ_1:

$$\begin{array}{rcl} \phi & = & \left\{ \begin{array}{l} T = \{\mathit{sensor}, \mathit{finger}, \mathit{loudness}\} \\ O = \{f{:}\mathit{finger}, s_c{:}\mathit{sensor}, s_d{:}\mathit{sensor}, s_e{:}\mathit{sensor}, ..., 0{:}\mathit{loudness}, 1{:}\mathit{loudness}, ...\} \\ P = \{h(\mathit{finger}, \mathit{sensor}), v(\mathit{sensor}, \mathit{loudness}), r(\mathit{sensor}, \mathit{sensor}), \\ \mathit{succ}(\mathit{loudness}, \mathit{loudness}), \mathit{max}(\mathit{loudness}), \mathit{min}(\mathit{loudness})\} \\ V = \{F{:}\mathit{finger}, L{:}\mathit{loudness}, S{:}\mathit{sensor}\} \end{array} \right\} \\ N_{\rightarrow} & = & 2 \\ N_{\ni} & = & 3 \\ N_B & = & 2 \end{array}$$

To generate the rest of the template sequence $(\chi_1, \chi_2, \chi_3, ...)$, we added additional unary predicates $p_i(\mathit{sensor})$, $q_i(\mathit{finger})$ and relational predicates $r_i(\mathit{sensor}, \mathit{sensor})$, as well as additional variables $L_i{:}\mathit{loudness}$ and $S_i{:}\mathit{sensor}$. We also provided domain-specific knowledge of the *succ* relation on loudness levels (e.g. $\mathit{succ}(0, 1), \mathit{succ}(1, 2), ...$), and we provide the spatial relation r on notes: $r(s_c, s_d), r(s_d, s_e), ..., r(s_b, s_{c*})$. This is the only domain-specific knowledge given.

We tested our system on some simple rhythms (*Pop Rock, Samba,* etc.) and tunes (*Twinkle Twinkle, Three Blind Mice*, etc). On the first two bars of *Twinkle Twinkle*, it finds an interpretation with 6 rules and 26 initial atoms. One of the rules states that when sensor S satisfies predicate p_1, then the value of the sensor S is set to the *max* loudness level:

$$p_1(S) \wedge \mathit{max}(L) \wedge v(S, L_2) \ni v(S, L)$$

This rule states that when sensor S satisfies p_2, then the value decays:

$$p_2(S) \wedge \mathit{succ}(L, L2) \wedge v(S, L_2) \ni v(S, L)$$

Clearly, p_1 and p_2 are exclusive unary predicates used to determine whether a note is currently being pressed or not.

The next rule states that when the finger F satisfies predicate q_1, then the note which the finger is on is pressed:

$$q_1(F) \wedge h(F, S) \wedge p_2(S) \ni p_1(S)$$

Here, the system is using q_1 to indicate whether or not the finger is down. It uses the other predicates $q_2, q_3, ...$ to indicate which state the finger is in (and hence which note the finger should be on), and the other rules to indicate when to transition from one state to another.

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

Task	# static rules	# cause rules	# atoms	# inits	# clauses	complexity
Twinkle Twinkle	2	4	9	26	32	45
Eighth Note Drum Beat	4	8	29	13	25	62
Stairway to Heaven	4	8	30	13	25	63
Three Blind Mice	2	8	17	34	44	69
Twist	4	12	40	16	32	84
Mazurka	4	12	44	14	30	86

Table 4.4: The complexity of the interpretations found for rhythm and tune prediction tasks

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. The complexities of the interpretations are shown in Table 4.4. Note that the interpretations found are large and complex programs by the standards of state of the art ILP systems. In *Mazurka*, for example, the interpretation contained 16 update rules with 44 body atoms. In *Three Blind Mice*, the interpretation contained 10 update rules and 34 initialisation atoms making a total of 44 clauses.

In the 8 cases where the Apperception Engine failed to predict correctly, this was because the system failed to find a unified interpretation of the sensory sequence. It was not that the system found an interpretation which produced the wrong prediction. Rather, in the 8 failure cases, it was simply unable to find a unified interpretation within the memory and time limits. In the ECA tasks, by contrast, the system always found some unified interpretation for each of the 256 tasks, but some of these interpretations produced the wrong prediction.

4.2.3 *Seek Whence* and C-test sequence induction IQ tasks

Hofstadter introduced the **Seek Whence**[4] domain in [Hof95]. 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[5]:

- **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.

[4] The name is a pun on "sequence". See also the related Copycat domain [Mit93].

[5] Hofstadter used natural numbers, but we transpose the sequences to letters, to bring them in line with the Thurstone letter completion problems [TT41] and the C-test [HOMC98].

It is important to note that these tasks, unlike the tasks in the ECA domain or in the rhythm and music domain, have a certain subjective element. There are always many different ways of interpreting a finite sequence. Given that these different interpretations will provide different continuations, why privilege some continuations over others?

When Hernández-Orallo introduced the C-test [HOMC98, HO00, HOMPS+16, HO17], one of his central motivations was to address this "subjectivity" objection via the concept of *unquestionability*. If we are given a particular programming language for generating sequences, then a sequence $s_{1:T}$ is **unquestionable** if it is not the case that the smallest program π that generates $s_{1:T}$ is rivalled by another program π' that is almost as small, where π and π' have different continuations after T time-steps.

Consider, for example, the sequence **a, b, b, c, c, ...** This sequence is highly questionable because there are two interpretations which are very similar in length (according to most programming languages): one parses the sequence as $(a), (b, b), (c, c, c), (d, d, d, d), ...$, and the other parses the input as a sequence of pairs $(a, b), (b, c), (c, d), ...$ Hernández-Orallo generated the C-test sequences by enumerating programs from a particular domain-specific language, executing them to generate a sequence, and then restricting the computer-generated sequences to those that are unquestionable.

For our set of sequence induction tasks, we combined sequences from Hofstadter's *Seek Whence* dataset (transposed from numbers to letters) together with sequences from the C-test. The C-test sequences are unquestionable by construction, and we also observed (by examining the size of the smallest interpretations) that Hofstadter's sequences were unquestionable with respect to Datalog$^{\ni}$. This goes some way to answering the "subjectivity" objection[6].

There have been a number of attempts to implement sequence induction systems using domain-specific knowledge of the types of sequence to be encountered. Simon et al [SK63] implemented the first program for solving Thurstone's letter sequences [TT41]. Meredith [Mer86] and Hofstadter [Hof95] also used domain-specific knowledge: after observing various types of commonly recurring patterns in the *Seek Whence* sequences, they hand-crafted a set of procedures to detect the patterns. Although their search algorithm is general, the patterns over which it is searching are hand-coded and domain-specific.

If solutions to sequence induction or IQ tasks are to be useful in general models of cognition, it is essential that we do not provide domain-specific solutions to those tasks. As Hernández-Orallo et al [HOMPS+16] argue, " In fact, for most approaches the system does not learn to solve the problems but it is programmed to solve the problems. In other words, the task is hard-coded into the program and it can be easier to become 'superhuman' in many specific tasks, as happens with chess, draughts, some kinds of planning, and many other tasks. But humans are not programmed to do intelligence tests." What we want is a *general-purpose domain-agnostic* perceptual system that can solve sequence

[6] Some may still be concerned that the definition of unquestionability is relative to a particular domain-specific language, and the Kolmogorov complexity of a sequence depends on the choice of language. Hernández-Orallo [HO17] discusses this issue at length.

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,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, ...**

Figure 4.5: Sequences from *Seek Whence* and the C-test

induction tasks "out of the box" *without hard-coded domain-specific knowledge* [BD15].

The Apperception Engine described in this chapter is just such a general-purpose domain-agnostic perceptual system. We tested it on 30 sequences (see Figure 4.5), and it got 76.6% correct (23/30 correct, 3/30 incorrect and 4/30 timeout).

For the letter sequence induction problems, we provide the initial template χ_1:

$$\begin{array}{rcl}
\phi & = & \left\{ \begin{array}{l}
T = \{sensor, cell, letter\} \\
O = \{s{:}sensor, c_1{:}cell, l_a{:}letter, l_b{:}letter, l_c{:}letter, ...\} \\
P = \{value(sensor, letter), h(sensor, cell), p(cell, letter), q_1(cell), r(cell, cell)\} \\
V = \{X{:}sensor, Y{:}cell, Y_2{:}cell, L{:}letter, L_2{:}letter\}
\end{array} \right\} \\
N_{\rightarrow} & = & 1 \\
N_{\ni} & = & 2 \\
N_B & = & 3
\end{array}$$

As we iterate through the templates ($\chi_1, \chi_2, \chi_3, ...$), we increase the number of objects, the number of fluent and permanent predicates, the number of static rules and causal rules, and the number of atoms allowed in the body of a rule.

The one piece of domain-specific knowledge we inject is the successor relation between the letters l_a, l_b, l_c, ... We provide the *succ* relation with $succ(l_a, l_b), succ(l_b, l_c), ...$ Please note that this knowledge does not *have* to be given to the system. We verified it is possible for the system to learn the successor relation on a simpler task and then *reuse* this information in subsequent tasks. We plan to do more continual learning from curricula in future work.

We illustrate our system on the "theme song" of the *Seek Whence* project: **b, a, b, b, b, b, b, c, b, b, d,**

b, b, e, b, b, ... Let the sensory sequence be $S_{1:16}$ where:

$$\begin{array}{lll}
S_1 = \{value(s, l_b)\} & S_2 = \{value(s, l_a)\} & S_3 = \{value(s, l_b)\} \\
S_4 = \{value(s, l_b)\} & S_5 = \{value(s, l_b)\} & S_6 = \{value(s, l_b)\} \\
S_7 = \{value(s, l_b)\} & S_8 = \{value(s, l_c)\} & S_9 = \{value(s, l_b)\} \\
S_{10} = \{value(s, l_b)\} & S_{11} = \{value(s, l_d)\} & S_{12} = \{value(s, l_b)\} \\
S_{13} = \{value(s, l_b)\} & S_{14} = \{value(s, l_e)\} & S_{15} = \{value(s, l_b)\} \\
S_{16} = \{value(s, l_b)\} & ... & ...
\end{array}$$

When our system is run on this sensory input, the first few templates are unable to find a solution. The first template that is expressive enough to admit a solution is one where there are three latent objects c_1, c_2, c_3. The first interpretation found is (ϕ, I, R, C) where:

$$\begin{array}{lll}
I & = & \left\{ \begin{array}{llll} p(c_1, l_b) & p(c_2, l_b) & p(c_3, l_a) & q_1(c_3) \\ q_2(c_1) & q_2(c_2) & r(c_1, c_3) & r(c_3, c_2) \\ r(c_2, c_1) & h(s, c_1) & & \end{array} \right\} \\
R & = & \left\{ \begin{array}{l} h(X, Y) \wedge p(Y, L) \rightarrow value(X, L) \\ r(Y, Y_2) \wedge h(X, Y) \ni h(X, Y_2) \\ h(X, Y) \wedge q_1(Y) \wedge succ(L, L_2) \wedge p(Y, L) \ni p(Y, L_2) \end{array} \right\} \\
C & = & \left\{ \begin{array}{l} \forall X{:}sensor,\ \exists! L\ value(X, L) \\ \forall Y{:}cell,\ \exists! L\ p(Y, L) \\ \forall Y{:}cell\ q_1(Y) \oplus q_2(Y) \\ \forall Y{:}cell,\ \exists! Y_2\ r(Y, Y_2) \end{array} \right\}
\end{array}$$

In this interpretation, the sensor moves between the three latent objects in the order $c_1, c_3, c_2, c_1, c_3, c_2, ...$ The two unary predicates q_1 and q_2 are used to divide the latent objects into two types. Effectively, q_1 is interpreted as an "object that increases its letter" while q_2 is interpreted as a "static object". The static rule states that the sensor inherits its value from the p-value of the object it is on. The causal rule $r(Y, Y_2) \wedge h(X, Y) \ni h(X, Y_2)$ states that the sensor moves from left to right along the latent objects. The causal rule $h(X, Y) \wedge q_1(Y) \wedge succ(L, L_2) \wedge p(Y, L) \ni p(Y, L_2)$ states that q_1 objects increase their p-value when a sensor is on them. This is an intelligible and satisfying interpretation of the sensory sequence. See Diagram 4.6.

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. The complexities of the interpretations are shown in Table 4.5. The first key

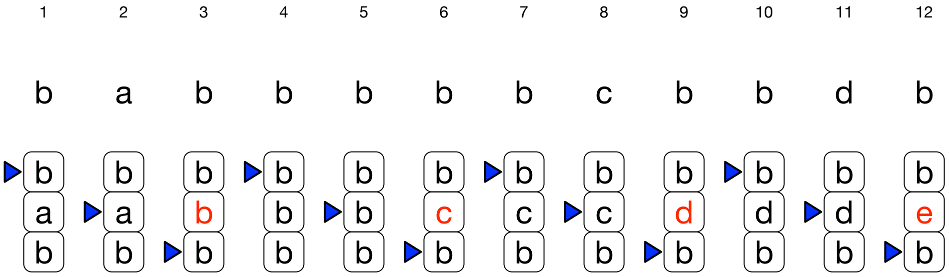


Figure 4.6: A visualization of the Apperception Engine's interpretation of the "theme song" *Seek Whence* sequence **b, a, b, b, b, b, b, c, b, b, d, b, b, e, b, b, ...** We show the trace $\tau(\theta) = A_1, A_2, ...,$ of this interpretation for the first 12 time steps. The t'th column represents the state at time t. Each column contains the time index t, the sensor reading S_t, the values of the three latent objects c_1, c_2, c_3 at time t, and the position of the sensor s at t. The only moving object is the sensor, represented by a triangle, that moves between the three latent objects from top to bottom and then repeats. Note that the middle object c_2's value changes when the sensor passes over it; we change the color of the object's letter to indicate when the object's value has changed.

Sequence	**# static rules**	**# cause rules**	**# body atoms**	**# inits**	**# clauses**	**complexity**
abcde...	0	1	1	7	8	10
fafbfcfdf...	1	2	6	7	10	18
babbbbbcbbdbbe...	1	2	6	14	17	25
aababcabcdabcde...	3	5	19	7	15	39
abccddeeefff...	3	5	21	8	16	42
baabbbaaaabbbbb...	3	5	23	7	15	43

Table 4.5: The complexity of the interpretations found for *Seek Whence* prediction tasks

point we want to emphasise here is that our system was able to achieve human-level performance[7] on these tasks without hand-coded domain-specific knowledge. This is a *general* system for making sense of sensory data that, when applied to the *Seek Whence* domain[8], 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[9]. 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.

4.2.4 Binding tasks

We wanted to see whether our system could handle traditional problems from cognitive science "out of the box", without needing additional task-specific information. We used probe tasks to evaluate two key issues: binding and occlusion.

The binding problem [Hol09] 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 and a siren in your auditory field and you see an insect and an ambulance in your visual field. How do you associate the buzzing and the insect-appearance as aspects of one object, and the siren and the ambulance appearance as aspects of a separate 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 Figure 4.2, 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 Figure 4.7.

Suppose we insist that the type signature contains no binary relations connecting any of the sensors together. Suppose there is no relation in the given type signature between light sensors, no relation between touch sensors, and no relation between light sensors and touch sensors. Now, in order to satisfy the constraint of object connectedness, there must be some indirect connection between any two sensors. But if there are no direct relations between the sensors, *the only way our system can satisfy the constraint of object connectedness is by positing latent objects, directly connected to each other, that the sensors are connected to*. Thus the latent objects are the intermediaries through which the various sensors are indirectly connected.

[7] See Meredith [Mer86] for empirical results with 25 students on the"Blackburn dozen" *Seek Whence* problems.

[8] The only domain-specific information provided is the *succ* relation on letters.

[9] Machine learning approaches to these tasks need thousands of examples before they can learn to predict. See for example [BHS+18].

For the binding tasks, we started with the initial template χ_1:

$$
\begin{array}{rcl}
\phi & = & \left\{ \begin{array}{l}
T = \{cell, light, touch, int\} \\
O = \{c_1{:}cell, c_2{:}cell, ..., c_{11}{:}cell, l_1{:}light, l_2{:}light, ..., l_{11}{:}light, t_1{:}touch, \\
t_2{:}touch, 0{:}int, 1{:}int, ...\} \\
P = \{black(light), white(light), value(touch, int), on(cell), \\
off(cell), r(cell, cell), in_L(light, cell), in_T(touch, cell), min(int), max(int), succ(int, int)\} \\
V = \{C{:}cell, X{:}touch, Y{:}light, L{:}int\}
\end{array} \right\} \\
N_{\rightarrow} & = & 4 \\
N_{\ni} & = & 4 \\
N_B & = & 4
\end{array}
$$

We provided as background knowledge information about the predicates *min*, *max*, and *succ*, e.g. $succ(2, 3)$.

As we iterate through the templates $(\chi_1, \chi_2, \chi_3, ...)$, we increase the number of predicates, the number of variables, the number of static rules and causal rules, and the number of atoms allowed in the body of a rule.

Given the template sequence $(\chi_1, \chi_2, \chi_3, ...)$, our system found the following interpretation (ϕ, I, R, C), where:

$$
I = \left\{ \begin{array}{llll}
off(c_1) & off(c_2) & off(c_3) & off(c_4) \\
off(c_5) & on(c_6) & off(c_7) & off(c_8) \\
off(c_9) & off(c_{10}) & off(c_{11}) & r(c_1, c_{11}) \\
r(c_2, c_1) & r(c_3, c_2) & r(c_4, c_3) & r(c_5, c_4) \\
r(c_6, c_5) & r(c_7, c_6) & r(c_8, c_7) & r(c_9, c_8) \\
r(c_{10}, c_9) & r(c_{11}, c_{10}) & in_L(l_1, c_1) & in_L(l_2, c_2) \\
... & in_L(l_{11}, c_{11}) & in_T(t_1, c_3) & in_T(t_2, c_{11})
\end{array} \right\}
$$

$$
R = \left\{ \begin{array}{l}
r(C_1, C_2) \wedge on(C_2) \wedge off(C_1) \ni on(C_1) \\
r(C_1, C_2) \wedge r(C_2, C_3) \wedge on(C_1) \wedge on(C_3) \wedge on(C_2) \ni off(C_2) \\
touch(X, L_1) \wedge min(L_1) \wedge p(X, L_2) \ni p(X, L_1) \\
touch(X, L_1) \wedge succ(L_2, L_1) \wedge p(X, L_1) \ni p(X, L_2) \\
in_T(X, C) \wedge on(C) \wedge max(L) \rightarrow value(X, L) \\
in_T(X, C) \wedge off(C) \wedge p(X, L) \rightarrow value(X, L) \\
in_L(Y, C) \wedge on(C) \rightarrow black(Y) \\
in_L(Y, C) \wedge off(C) \rightarrow white(Y)
\end{array} \right\}
$$

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

Figure 4.7: 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.

$$C \;=\; \left\{ \begin{array}{l} \forall C{:}cell,\ on(C) \oplus off(C) \\ \forall C{:}cell,\ \exists! C_2\ r(C, C_2) \\ \forall X{:}touch,\ \exists! C\ in_T(X, C) \\ \forall X{:}touch,\ \exists! L\ value(X, L) \\ \forall Y{:}light,\ black(Y) \oplus white(Y) \\ \forall Y{:}light,\ \exists! C\ in_L(X, C) \end{array} \right\}$$

Here, the cells $c_1, ..., c_{11}$ are directly connected via the r relation, and the light and touch sensors are connected to the cells via the in_L and in_T relations. Thus all the sensors are indirectly connected. For example, light sensor l_1 is indirectly connected to touch sensor t_1 via the chain of relations $in_L(l_1, c_1), r(c_1, c_2), r(c_2, c_3), in_T(t_1, c_3)$. We stress that we did not need to write special code in order to get the system to satisfy the binding problem. Rather, *the binding problem is satisfied automatically, as a side-effect of satisfying the object connectedness condition*.

We ran 20 multi-modal binding experiments, with different ECA rules, different initial conditions, and the touch sensors attached to different cells. The results are shown in Table 4.6.

4.2.5 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.

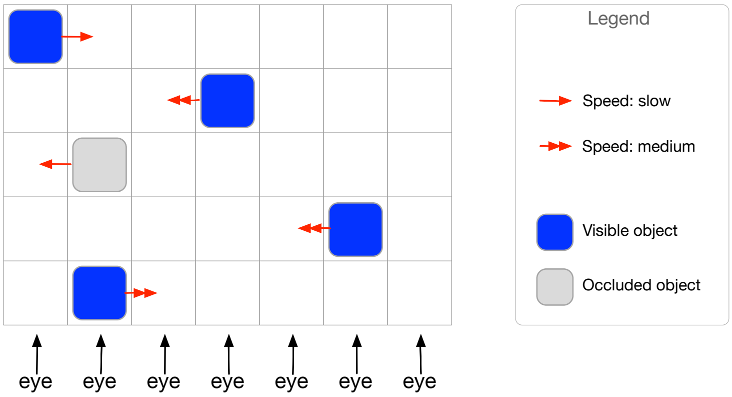


Figure 4.8: An occlusion task

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 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 Figure 4.8.

The system receives a sensory sequence consisting of the positions of the moving objects whenever they are visible. The positions 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.

For the occlusion tasks, we provide the initial template χ_1:

- $T = \{cell, mover\}$
- $O = \{c_{1,1}{:}cell, ..., c_{7,5}{:}cell, m_1{:}mover, ..., m_5{:}mover\}$
- $P = \{in(mover, cell), right(cell, cell), below(cell, cell),$
  $p_1(mover), p_2(mover), p_3(mover), p_4(mover)\}$

We provide the *right* and *below* predicates defining the 2D relation between grid cells. The system is free to interpret the p_i predicates any way it desires.

Here is a sample of the rules generated to solve one of the tasks:

$$
\begin{aligned}
&p_1(X) \wedge right(C_1, C_2) \wedge in(X, C_1) \ni in(X, C_2) \\
&p_2(X) \wedge p_3(X) \ni p_4(X) \\
&p_2(X) \wedge p_4(X) \ni p_3(X) \\
&p_2(X) \wedge right(C_1, C_2) \wedge p_4(X) \wedge in(X, C_2) \ni in(X, C_1)
\end{aligned}
$$

Domain	# Tasks	Memory	Time	% Correct
Multi-modal binding	20	1003.2	2.4	85.0%
Occlusion	20	604.3	2.3	90.0 %

Table 4.6: The two types of probe task. We show mean memory in megabytes and mean solution time in hours.

The rules describe the behaviour of two types of moving objects. An object of type p_1 moves right one cell every time-step. An object of type p_2 moves left every two time-steps. It uses the state predicates p_3 and p_4 as *counters* to determine when it should move left and when it should remain where it is.

We generated 20 occlusion tasks by varying the size of the grid, the number of moving objects, their direction and speed. Our system was able to solve these tasks without needing additional domain-specific information. The results are shown in Table 4.6.

4.3 Empirical comparisons with other approaches

In this section, we evaluate our system experimentally and attempt to establish the following claims. First, we claim the test domains of Section 5.5 represent a challenging set of tasks. We show that these domains are challenging by providing baselines that are unable to interpret the sequences. Second, we claim our system is general in that it can handle retrodiction and imputation as easily as it can handle prediction tasks. We show in extensive tests that results for retrodicting earlier values and imputing intermediate values are comparable with results for predicting future values. Third, we claim that the various features of the system (the Kantian unity conditions and the cost minimization procedure) are essential to the success of the system. In ablation tests, where individual features are removed, the system performs significantly worse.

4.3.1 Our domains are challenging for existing baselines

To evaluate whether our domains are indeed sufficiently challenging, we compared our system against four baselines.[10] 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) [Mur12] that looks at a window of earlier time-steps to predict the next time-step. The fourth **LSTM** baseline is a recurrent neural net based on the long short-term memory (LSTM) architecture [HS97].

We also considered using a hidden Markov model (HMM) as a baseline. However, as Ghahramani emphasizes ([Gha01], Section 5), a HMM represents each of the exponential number of propositional

[10] I am very grateful to Johannes Welbl for designing and implementing the neural baselines.

states separately, and thus fails to generalize in the way that a first-order rule induction system does. Thus, although we did not test it, we are confident that a HMM would not perform well on our tasks.

Although the neural network architectures are very different from our system, we tried to give the various systems access to the same amount of information. This means in particular that:

- Since our system interprets the sequence without any knowledge of the other sequences, *we do not allow the neural net baselines to train on any sequences other than the one they are currently given*. Each neural net baseline is only allowed to look at the single sensory sequence it is given. This extreme paucity of training data is unusual for data-hungry methods like neural nets, and explains their weak results. But we stress that this is the only fair comparison, given that the Apperception Engine, also, only has access to a single sequence.
- Since our system interprets the sequence without knowing anything about the relative spatial position of the sensors (it does not know, in the ECA examples, the spatial locations of the cells), we do not give the neural nets a (1-dimensional) convolutional structure, even though this would help significantly in the ECA tasks.

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 $\mathbf{x}$ of relevant context sensors, and a feed-forward network is trained to predict the correct state $\mathbf{y}$ of a given sensor in question. In the prediction task, the feature representation comprises one-hot[11] 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 [KB14] 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 extremely small number of data points (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.

[11] 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.

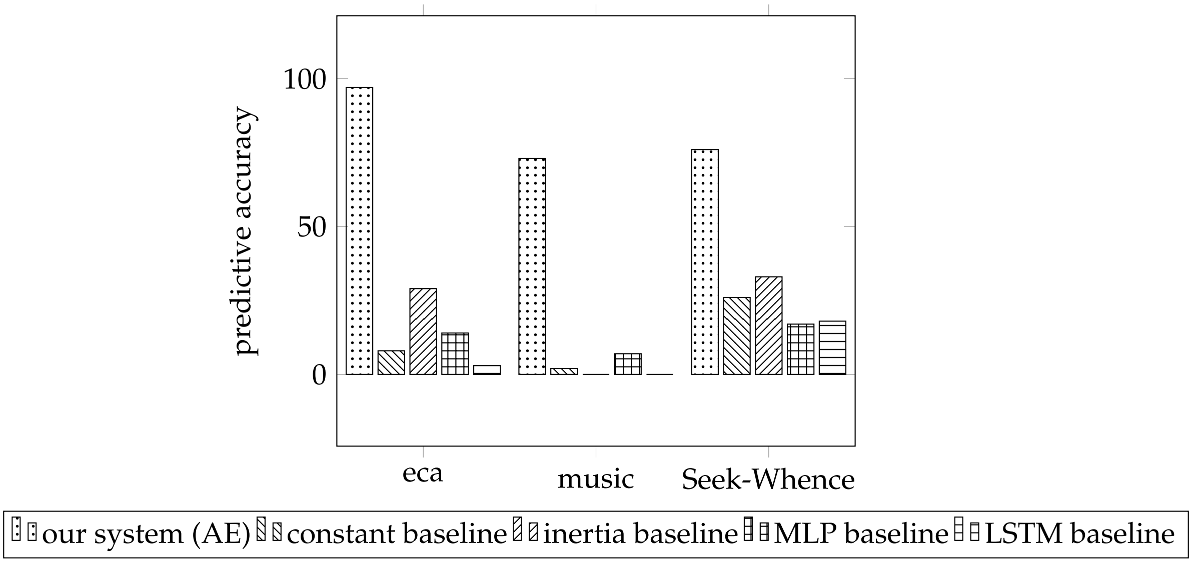


Figure 4.9: Comparison with baselines. We display predictive accuracy on the held-out final time-step.

	ECA	Rhythm & Music	Seek Whence
Our system (AE)	97.3%	73.3%	76.7%
Constant baseline	8.6%	2.5%	26.7%
Inertia baseline	29.2%	0.0%	33.3%
Neural MLP	15.5%	1.3%	17.9%
Neural LSTM	3.3%	0.0%	18.7%

Table 4.7: Comparison with baselines. We display predictive accuracy on the held-out final time-step.

Why didn't we give the LSTM significantly longer sequences, to give them a more reasonable chance of success? It has been shown, in a variety of situations, that humans are able to make sense of short sequences [Mit93, Hof95, Mar18a, LUTG17]. My aim in this thesis was to build a machine with the right inductive bias that could also learn from short sequences. The aim was not to see if these problems could be solved with an unrealistic amount of data – rather, the aim was to see whether it is possible for a machine to solve problems in sparse data regimes. Thus, since we give the Apperception Engine short sequences, it is only fair to give the same length sequences to the LSTM.

Figure 4.9 shows the results. Clearly, the tasks are very challenging for all four baseline systems.

Table 4.7 shows a comparison with four baselines: a constant baseline (that always predicts the same thing), the inertia baseline (that predicts the final time-step equals the penultimate time-step), a simple neural baseline (a fully connected MLP), and a recurrent neural net (an LSTM [HS97]). The results for the neural MLP and LSTM are averaged over 5 reruns.

Table 4.8 shows the McNemar test [McN47] for the four baselines. For each baseline, we assess the null hypothesis that its distribution is the same as the distribution of the Apperception Engine. If b is the proportion of tasks on which the Apperception Engine is inaccurate, and c is the proportion of

	ECA	Rhythm & Music	Seek Whence
AE vs constant baseline	216.6	11.9	7.8
AE vs inertia baseline	164.2	12.7	6.3
AE vs neural MLP	200.6	11.1	7.0
AE vs neural LSTM	242.1	12.7	6.9

Table 4.8: The McNemar test comparing our system (AE) to each baseline. The McNemar test statistic generates a χ^2 distribution with 1 degree of freedom. For each entry in the table, the null hypothesis (that the baseline's distribution is the same as our system's distribution) is extremely unlikely.

tasks in which the baseline is inaccurate, then the McNemar test statistic is

$$\chi^2 = \frac{(b-c)^2}{b+c}$$

In comparison with the Apperception Engine, the LSTM baseline has very little inductive bias to help it solve the apperception tasks. The LSTM has no equivalent of the frame axiom, no equivalent of the Kantian unity conditions, and is not able to represent latent unobserved information. While the Apperception Engine is able to posit latent properties and objects to explain the surface information, the LSTM operates purely at the surface level [McC06]. But it should be possible to design a more complex LSTM baseline that is able to induce latent information: let each state be a pair (X, L), where X is the explicit surface information and L is the latent information, and let the LSTM update from state (X, L) to (X', L'). We add weights to the network that are interpreted as representing the initial condition L_0 of the latent information. To calculate the loss, we extract the X state from the (X, L) pair and compare with the observed result. Implementing this more complex neural baseline is an exercise for future work.

Although it is possible to add latent unobserved information to the LSTM, it is much less clear how to add a frame axiom or the Kantian unity conditions to the LSTM. The frame axiom states that a fact persists until some other fact becomes true that is incompossible with it. This clearly relies on the notion of incompossibility between atoms. But if atoms are represented implicitly, in a vector of activations in a neural network, it is not clear how to detect when two atoms are incompossible.

Consider, next, what would be involved in adding the requirement of conceptual unity to a neural network. The conceptual unity condition insists that every predicate must appear in some xor constraint. But in a neural model, xor constraints are represented only implicitly in the weights of a network. Thus, it is hard to see how to detect whether or not a particular predicate features in some constraint, when the constraints are hidden in the network's weights, and thus it is not clear how to detect whether a neural network is respecting the requirement of conceptual unity.

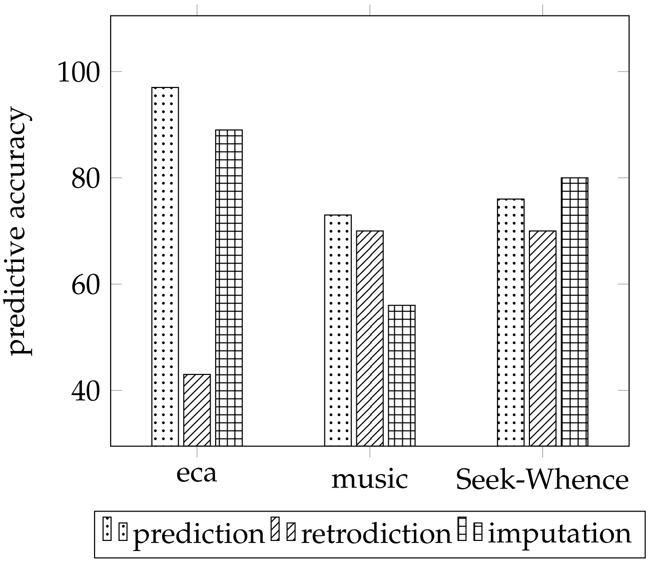


Figure 4.10: Comparing prediction with retrodiction and imputation. In retrodiction, we display accuracy on the held-out initial time-step. In imputation, a random subset of atoms are held-out; the held-out atoms are scattered throughout the time-series. In other words, there may be different held-out atoms at different times. The number of held-out atoms in imputation matches the number of held-out atoms in prediction and retrodiction.

4.3.2 Our system handles retrodiction and imputation just as easily as prediction

To verify that 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). We made sure that the number of hidden sensor values was the same for prediction, retrodiction, and imputation.

Figure 4.10 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. So, for example, in Rule # 0, one trajectory is shown in Figure 4.11. Here, although it is possible to predict the future state from earlier states, it is not possible to retrodict the initial state given subsequent states.

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 (see Figure 4.10).

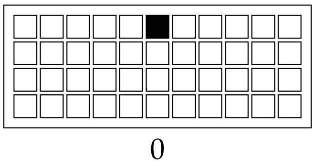


Figure 4.11: One trajectory for ECA rule # 0. This trajectory shows how information is lost as we progress through time. Here, clearly, retrodiction (where the first row is held-out) is much harder than prediction (where the final row is held-out).

4.3.3 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 object connectedness. 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 4.9.

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 object connectedness 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

	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 object connectedness	95.7%	73.3%	73.3%
No cost minimization	96.7%	56.6%	73.3%

Table 4.9: Ablation experiments. We display predictive accuracy on the final held-out time-step.

identical for the rhythm tasks is because the background knowledge provided (the r relation on notes, see Section 4.2.2) means that the object connectedness constraint is guaranteed to be satisfied. The reason why the results are lower for ECA tasks is because interpretations that fail to satisfy object connectedness 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 Seek Whence tasks is because the lowest cost unified interpretation for most of these tasks also happens to satisfy object connectedness. In future work, we shall test the Apperception Engine in a much wider variety of domains, to understand when object connectedness is[12] and is not[13] important.

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, since shorter interpretations are more human-readable. On the other hand, it is significantly more expensive to compute the lowest cost theory than it is to just find any unified theory (see the complexity results in Section 3.7.4). So in some domains, where the difference in accuracy is minimal, the cost minimization step can be avoided.

4.4 Discussion

This chapter is an attempt to answer a key question of unsupervised learning: what does it mean to "make sense" of a (discretised) sensory sequence? Our answer is broadly Kantian [CFH92]: making sense means positing a collection of objects that persist over time, with attributes that change over time, according to intelligible laws. 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

[12] Recently, we have found other cases where this object connectedness constraint is necessary. Andrew Cropper has some recent unpublished work using object invention in which the object connectedness constraint was found to be essential.

[13] Some philosophers (e.g. Strawson [Str18]) have questioned whether spatial unity is, in fact, necessary to make sense of the sensory stream.

system achieves good results across the board.

Of particular note is that it is able to achieve human performance on challenging sequence induction IQ tasks. We stress, once more, that the system was not hard-coded to solve these tasks. Rather, it is 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[14]. 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*. We also stress that the interpretations produced are human readable and can be used to provide explanations and justifications of the decisions taken: when the Apperception Engine produces an interpretation, we can not only see what it predicts will happen next, but we can also understand *why* it thinks this is the right continuation. We believe these results are highly suggestive, and shows that a sense-making component such as this will be a key aspect of any general intelligence.

Our architecture, an unsupervised program synthesis system, is a purely symbolic system, and as such, it inherits two key advantages of ILP systems [EG18]. First, the interpretations produced are *interpretable*. Because the output is symbolic, it can be read and verified by a human[15]. Second, it is very *data-efficient*. Because of the language bias of the Datalog$^{\ni}$ language, and the strong inductive bias provided by the Kantian 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 clear limitations. First, it does not currently handle noise in the sensory input. All sensory information is assumed to be significant, and the system will strive to find an explanation of *every* sensor reading. There is no room for the idea that some sensor readings are inaccurate.

Second, 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 classes.

The first limitation is addressed in Section 4.5, and the second limitation is addressed in Chapter 5.

4.5 Noisy apperception

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

[14] Raven's progressive matrices [CJS90] are spatial reasoning tasks requiring inductive reasoning. Barrett et al [BHS+18] train a neural network to learn to solve Raven's progressive matrices, but their method requires millions of training examples.

[15] 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 [MSZ+18] for an extensive discussion.

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

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

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:

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

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 mislabeled 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 $i = 1..T$, using the following simple argument.

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

$$\arg\max_{\theta}\ p\left(\theta \mid S_{1:T}\right) \tag{4.1}$$

By Bayes' rule, this is equivalent to

$$\arg\max_{\theta}\ \frac{p(\theta) \cdot p\left(S_{1:T} \mid \theta\right)}{p(S_{1:T})} \tag{4.2}$$

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

$$\arg\max_{\theta}\ p(\theta) \cdot p\left(S_{1:T} \mid \theta\right) \tag{4.3}$$

Since the probability of the state S_i is conditionally independent of the previous state S_{i-1} given θ

(this is the assumption of the Hidden Markov Model), the above is equivalent to:

$$\arg\max_{\theta} \; p(\theta) \cdot \prod_{i=1}^{T} p\left(S_i \mid \theta\right) \tag{4.4}$$

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

$$\arg\max_{\theta} \; p(\theta) \cdot \prod_{i=1}^{T} p\left(S_i \mid \tau(\theta)_i\right) \tag{4.5}$$

Let the probability of θ be $2^{-len(\theta)}$. Let[16] 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:

$$\arg\max_{\theta} \; 2^{-len(\theta)} \cdot \prod_{i=1}^{T} 2^{-|S_i - \tau(\theta)_i|} \tag{4.6}$$

Since log_2 is monotonic, we can take logs and rewrite to:

$$\arg\max_{\theta} \; -len(\theta) + \sum_{i=1}^{T} -|S_i - \tau(\theta)_i| \tag{4.7}$$

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

$$cost_{noise} = len(\theta) + \sum_{i=1}^{T} |S_i - \tau(\theta)_i| \tag{4.8}$$

and search for the theory with lowest cost.

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

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

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

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

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

[16] If we want to weight unexplained ground atoms differently from unground atoms of the theory, we could add a β parameter and use the more general formula $p\left(S_i \mid \tau(\theta)_i\right) = 2^{-\beta \cdot |S_i - \tau(\theta)_i|}$.

of a "proper" explanation over a degenerate solution becomes more and more apparent. Consider, for example, the following extension $S'_{1:30}$:

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

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

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

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. $\triangleleft$

4.5.1 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,b,a,a,b,b,a,a,b,b, ...
a,b,b,a,a,b,b,a,a,b,b,a, ...
a,b,c,b,a,a,b,c,b,a,a,b,c,b,a, ...
a,b,c,c,a,b,c,c,a,b,c,c, ...
a,a,b,a,a,b,a,a,b,a,a,b, ...
a,a,a,b,a,a,a,b,a,a,a,b, ...
a,b,c,a,b,c,a,b,c,a,b,c, ...
a,b,a,c,a,b,a,c,a,b,a,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. Figure 4.12 shows the results. We plot mean

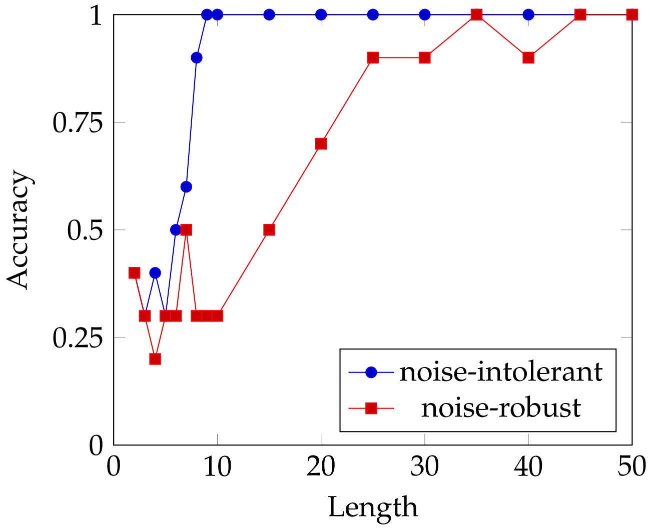


Figure 4.12: Comparing the data-efficiency of the noise-robust version of the Apperception Engine with the noise-intolerant version on noise-free sequences. We plot mean percentage accuracy against length of the sequence. The noise-intolerant version achieves 100% accuracy when the sequence is length 10 or over, while the noise-robust version only achieves this level of accuracy when the length is over 30.

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 45.

In the second experiment, we evaluate how much better the noise-robust version of the Apperception Engine is at handling mislabeled data. We take the same ten sequences above, extended to length 100, and consider various perturbations of the sequence where we randomly mislabel a certain number of entries. Figure 4.13 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 mislabeled.

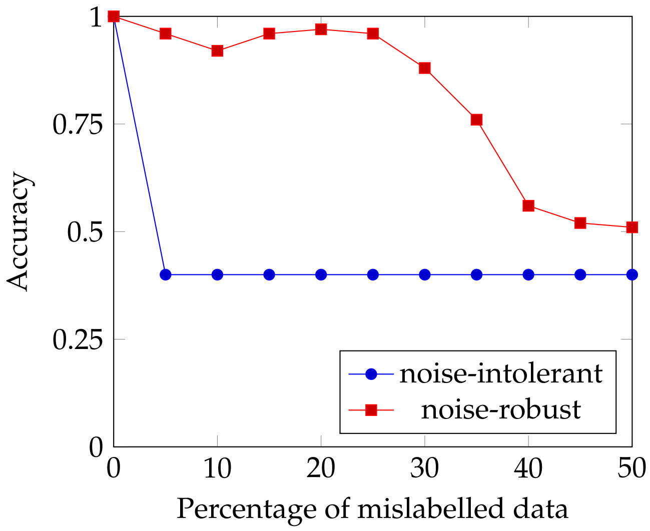


Figure 4.13: Comparing the accuracy of the noise-robust version of the Apperception Engine with the noise-intolerant version. We plot mean percentage accuracy against the number of mislabellings. The noise-intolerant version deteriorates to random as soon as any noise is introduced, while the noise-robust version is able to maintain reasonable (88%) accuracy with up to 30% of the sequence mislabelled.

Chapter 5

Making sense of raw input

This material is based on "Making sense of raw input", which is in review for *Artificial Intelligence*.[1] It is also based on my article "Apperception", in *Human-Like Machine Intelligence*, Oxford University Press, 2020 (forthcoming).

In this chapter, we extend the Apperception Engine so that it can handle raw unprocessed sensory input. First, we shall define what it means to make sense of a disjunctive sensory sequence. Second, we shall show how to use a neural network to transform raw unprocessed sensory input into a disjunctive sensory sequence.

5.1 Making sense of disjunctive symbolic input

In this section, we extend the Apperception Engine to handle disjunctive sensory input.

Definition 23. *A* **disjunctive input sequence** *is a sequence of sets of disjunctions of ground atoms.* △

A disjunctive input sequence generalises the input sequence of Definition 1 to handle uncertainty. Now if we are unsure if a sensor a satisfies predicate p or predicate q, we can express our uncertainty as $p(a) \vee q(a)$.

Example 13. Consider, for example, the following sequence $D_{1:10}$. This is a disjunctive variant of the unambiguous sequence from Example 1. Here there are two sensors a and b, and each sensor can be *on* or *off*.

$$\begin{array}{lll} D_1 = \{\} & D_2 = \{\mathit{off}(a), \mathit{on}(b)\} & D_3 = \{\mathit{on}(a), \mathit{off}(b)\} \\ D_4 = \{\mathit{on}(a), \mathit{on}(b)\} & D_5 = \{\mathit{off}(a) \vee \mathit{on}(a), \mathit{on}(b)\} & D_6 = \{\mathit{on}(a), \mathit{off}(b)\} \\ D_7 = \{\mathit{on}(a), \mathit{on}(b)\} & D_8 = \{\mathit{off}(a), \mathit{on}(b)\} & D_9 = \{\mathit{off}(a) \vee \mathit{on}(a)\} \\ D_{10} = \{\} & & \end{array}$$

[1] The paper is co-authored with Matko Bosnjak, Lars Buesing, Kevin Elllis, Pushmeet Kohli, and Marek Sergot. Matko Bosnjak implemented the neural net baselines in Sections 5.5. Lars Buesing helped with the related work. Kevin Ellis helped with the derivation of the formulas in Section 5.3. Pushmeet Kohli is my advisor at DeepMind.

$D_{1:10}$ contains less information than $S_{1:10}$ from Example 1, since D_9 is unsure whether a is *on* or *off*, while in S_9 a is *on*. ◃

Recall that the $\sqsubseteq$ relation describes when one (finite) sequence is covered by another. We extend the $\sqsubseteq$ relation to handle disjunctive input sequences in the first argument.

Definition 24. *Let $D = (D_1, ..., D_T)$ be a (finite) disjunctive input sequence and S be an input sequence. $D \sqsubseteq S$ if S contains a finite subsequence $(S_1, ..., S_T)$ such that $S_i \models D_i$ for all $i = 1..T$.* △

Example 14. The theory θ of Example 3 explains the disjunctive sequence D of Example 13, since the trace $\tau(\theta)$ (as shown in Example 3) covers D. ◃

The disjunctive apperception task generalises the simple apperception task of Definition 18 to disjunctive input sequences.

Definition 25. *The input to a disjunctive apperception task is a triple (D, ϕ, C) consisting of a disjunctive input sequence D, a suitable type signature ϕ, and a set C of (well-typed) constraints such that (i) for each disjunction featuring predicates $p_1, ..., p_n$ there exists a constraint in C featuring each of $p_1, ..., p_n$. (ii) D can be extended to satisfy C.*

Given such an input triple (D, ϕ, C), the **disjunctive apperception task** *is to find a lowest cost theory $\theta = (\phi', I, R, C')$ such that ϕ' extends ϕ, $C' \supseteq C$, $D \sqsubseteq \tau(\theta)$, and θ satisfies the four unity conditions of Definition 11.* △

5.2 Making sense of raw input

The reason for introducing the disjunctive apperception task is as a stepping stone to the real task of interest: making sense of sequences of raw uninterpreted sensory input.

Definition 26. *Let $\mathbf{R}$ be the set of all possible raw inputs. A* **raw input sequence** *of length T is a sequence $(\mathbf{r}_1, ..., \mathbf{r}_T)$ in $\mathbf{R}^T$. Here $\mathbf{R}$ is the set of all possible raw inputs for a single time step, e.g. the set of all 20×20 binary pixel arrays.* △

A raw apperception framework uses a neural network $\pi_\mathbf{w}$, parameterised by weights $\mathbf{w}$, to map subregions of each $\mathbf{r}_i$ into subsets of classes $\{1, ..., n\}$. Then the results of the neural network are transformed into a disjunction of ground atoms, transforming the raw input sequence into a disjunctive input sequence.

Definition 27. *A* **raw apperception framework** *is a tuple $(\pi_\mathbf{w}, n, \Delta, \phi, C)$, where:*

- *$\pi_\mathbf{w}$ is a neural network, a multilabel classifier mapping subregions of $\mathbf{r}_i$ to subsets of $\{1, ..., n\}$; π is parameterised by weight vector $\mathbf{w}$*

- *n is the number of classes that the perceptual classifier $\pi_{\mathbf{w}}$ uses*
- *Δ is a "disjunctifier" that converts the results of the neural network $\pi_{\mathbf{w}}$ into a set of disjunctions; it takes as input the result of repeatedly applying[2] $\pi_{\mathbf{w}}$ to the N subregions $\{\mathbf{p}_i^1, ..., \mathbf{p}_i^N\}$ of $\mathbf{r}_i$, and produces as output a set of N disjunctions of ground atoms*
- *ϕ is a type signature*
- *C is a set of constraints*

The input to a raw apperception task is a raw sequence together with a raw apperception framework. Given sequence $\mathbf{r} = (\mathbf{r}_1, ..., \mathbf{r}_T)$ and framework $(\pi_{\mathbf{w}}, n, \Delta, \phi, C)$, the **raw apperception task**[3] *is to find the lowest cost weights w and theory θ such that θ is a solution to the disjunctive apperception task $((D_1, ..., D_T), \phi, C)$ where $D_i = \Delta(\pi_{\mathbf{w}}(\mathbf{p}_i^1), ..., \pi_{\mathbf{w}}(\mathbf{p}_i^N))$.*

The best $(\theta, \mathbf{w})$ pair is:

$$\arg\max_{\theta,\, \mathbf{w}} \; \log p(\theta) + \sum_{i=1}^{T} \sum_{j=1}^{N} \log \frac{1}{|\{\mathbf{p} \in \mathbf{P} \mid k_i^j \in \pi_{\mathbf{w}}(\mathbf{p})\}|}$$

where $\mathbf{P} = \{\mathbf{p}_i^j \mid i = 1..T, j = 1..N\}$ is the union of the subregions appearing at each position j and at each time-step i, and k_i^j is the atom in the i'th state of $\tau(\theta)$ that represents the class of the object in region j.

△

The intuition here is that $p(\theta) \propto 2^{-cost(\theta)}$ represents the prior probability of the theory θ, while the second term penalises the neural network $\pi_{\mathbf{w}}$ for mapping many elements to the same class. In other words, it prefers highly selective classes, minimising the number of elements that are assigned by $\pi_{\mathbf{w}}$ to the same class. This particular optimisation can be justified using Bayes theorem, as we now show.

5.3 Finding the most probable interpretation

We are given a raw sequence $\mathbf{r} = (\mathbf{r}_1, ..., \mathbf{r}_T)$ together with a raw framework $(\pi_{\mathbf{w}}, n, \Delta, \phi, C)$, where neural network π is parameterised by weight vector $\mathbf{w}$. We want to find the most probable theory θ and weights $\mathbf{w}$ given our raw input sequence $\mathbf{r}$:

$$\arg\max_{\theta,\, \mathbf{w}} \; p\,(\theta, \mathbf{w} \mid \mathbf{r}) \tag{5.1}$$

[2] This repeated application of the same neural net to each subregion is inspired by the convolutional neural network [LB+95].

[3] For concrete examples of this rather abstract definition, see Sections 5.5.2 and 5.5.3.

By Bayes rule, this is equivalent to

$$\arg\max_{\theta,\,\mathbf{w}} \frac{p\left(\mathbf{r} \mid \theta, \mathbf{w}\right) \cdot p(\theta, \mathbf{w})}{p\left(\mathbf{r}\right)} \tag{5.2}$$

Since the denominator does not depend on θ or $\mathbf{w}$, this is equivalent to:

$$\arg\max_{\theta,\,\mathbf{w}} \; p\left(\mathbf{r} \mid \theta, \mathbf{w}\right) \cdot p(\theta, \mathbf{w}) \tag{5.3}$$

Assuming the priors of θ and $\mathbf{w}$ are independent, $p(\theta, \mathbf{w})$ can be decomposed to get:

$$\arg\max_{\theta,\,\mathbf{w}} \; p\left(\mathbf{r} \mid \theta, \mathbf{w}\right) \cdot p(\theta) \cdot p(\mathbf{w}) \tag{5.4}$$

Let us assume the prior $p(\mathbf{w})$ on the weight vector is uniform, so can be dropped:

$$\arg\max_{\theta,\,\mathbf{w}} \; p\left(\mathbf{r} \mid \theta, \mathbf{w}\right) \cdot p(\theta) \tag{5.5}$$

Let the trace $\tau(\theta) = (A_1, A_2, ...)$. As each $\mathbf{r}_i$ is conditionally independent of $\mathbf{r}_{i-1}$ given θ, $p(\mathbf{r} \mid \theta, \mathbf{w}) = p(\tau(\theta) \mid \theta, \mathbf{w}) \cdot \prod_{i=1}^{T} p(\mathbf{r}_i \mid A_i, \theta, \mathbf{w})$, we can rewrite to get:

$$\arg\max_{\theta,\,\mathbf{w}} \; p(\theta) \cdot p(\tau(\theta) \mid \theta, \mathbf{w}) \cdot \prod_{i=1}^{T} p(\mathbf{r}_i \mid A_i, \theta, \mathbf{w}) \tag{5.6}$$

Since the latent symbolic trace $(A_1, A_2, ...)$ is deterministically generated from the theory θ, $p(\tau(\theta) \mid \theta, \mathbf{w}) = 1$, and we can remove this term to get:

$$\arg\max_{\theta,\,\mathbf{w}} \; p(\theta) \cdot \prod_{i=1}^{T} p(\mathbf{r}_i \mid A_i, \theta, \mathbf{w}) \tag{5.7}$$

Since $\mathbf{r}_i$ is conditionally independent of θ given A_i and $\mathbf{w}$, we can rewrite to:

$$\arg\max_{\theta,\,\mathbf{w}} \; p(\theta) \cdot \prod_{i=1}^{T} p(\mathbf{r}_i \mid A_i, \mathbf{w}) \tag{5.8}$$

Assuming raw data $\mathbf{r}_i$ can be decomposed into independent subregions $\mathbf{p}_i^1, ..., \mathbf{p}_i^N$, we can rewrite to:

$$\arg\max_{\theta,\,\mathbf{w}} \; p(\theta) \cdot \prod_{i=1}^{T} \prod_{j=1}^{N} p(\mathbf{p}_i^j \mid A_i, \mathbf{w}) \tag{5.9}$$

Assume that subregion $\mathbf{p}_i^j$ is stochastically sampled conditioned on the latent k_i^j in A_i. Here, k_i^j is an atom featuring a class label from $\{1, ..., n\}$ representing the type of object in region j at time i. Assume the raw subregions are sampled uniformly. Then the probability of the particular subregion $\mathbf{p}_i^j$ is 1 divided by the number of subregions that are mapped to class k_i^j:

$$p(\mathbf{p}_i^j \mid A_i, \mathbf{w}) = p(\mathbf{p}_i^j \mid k_i^j, \mathbf{w}) = \frac{1}{|\{\mathbf{p} \in \mathbf{P} \mid k_i^j \in \pi_w(\mathbf{p})\}|} \tag{5.10}$$

where $\mathbf{P} = \{\mathbf{p}_i^j \mid i = 1..T, j = 1..N\}$ is the union of the subregions appearing at each position j and at each time-step i.

Substituting Equation 5.10 in Formula 5.9 gives:

$$\underset{\theta,\, \mathbf{w}}{\arg\max}\ \ p(\theta) \cdot \prod_{i=1}^{T} \prod_{j=1}^{N} \frac{1}{|\{\mathbf{p} \in \mathbf{P} \mid k_i^j \in \pi_{\mathbf{w}}(\mathbf{p})\}|} \tag{5.11}$$

Since $log(.)$ is monotonic, we can rewrite to:

$$\underset{\theta,\, \mathbf{w}}{\arg\max}\ \ \log p(\theta) + \sum_{i=1}^{T} \sum_{j=1}^{N} \log \frac{1}{|\{\mathbf{p} \in \mathbf{P} \mid k_i^j \in \pi_{\mathbf{w}}(\mathbf{p})\}|} \tag{5.12}$$

5.4 Applying the Apperception Engine to raw input

Recall from Section 2.4 that a binary neural network (BNN) is a neural network in which the node activations and weights are all binary values in $\{0, 1\}$. Because the activations and weights are binary, the state of the network can be represented by a set of atoms, and the dynamics of the network can be defined as a logic program. This means we can combine the low-level perception task (of mapping raw data to concepts) and the high-level apperception task (of combining concepts into rules) *into a single logic program* in ASP, and solve both *simultaneously*.

5.4.1 Implementing a binary neural network in ASP

The network is configured by specifying the number of nodes in each layer. For example, to specify a network with 25 input nodes, 15 hidden units, and 10 output nodes, we write:

```
nodes(1, 25).
nodes(2, 15).
nodes(3, 10).
```

The output layer is defined to be the final layer:

```
is_output_layer(L) :-
```

```
    is_layer(L),
    not is_layer(L+1).

is_layer(L) :- nodes(L, _).
```

Each layer except the output layer has an additional node, the bias node. If layer `L` has `N` nodes with indices `1...N`, then the bias node has index `0`:

```
is_bias(node(L, 0)) :- is_layer(L).

is_node(node(L, I)) :-
    is_layer(L),
    not is_output_layer(L),
    nodes(L, N),
    I = 0 .. N.

is_node(node(L, I)) :-
    is_output_layer(L),
    nodes(L, N),
    I = 1 .. N.
```

Each node is represented by a term `node(L, I)` where `L` is the layer and `I` is the index. Each node has a set of input nodes with associated weights.

Choosing weights

We use `weight(X, Y, B)` to represent that the weight from input node `X` to node `Y` has binary value `B`. The assignment of weights is implemented by the choice rule:

```
0 { weight(node(L, I), node(L+1, J), B) : binary(B) } 1 :-
    is_node(node(L, I)),
    is_node(node(L+1, J)),
    J > 0.

binary(0).
binary(1).
```

This code makes two choices: first, whether or not to create a connection between `node(L, I)` and `node(L+1, J)`; second, if there is such a connection, the binary value of the weight. Note that if there is no weight from node `X` to node `Y` then `X` is not an input node to `Y`. Thus we can, if we wish, use a weak constraint to minimise the number of connections:

```
:~ weight(N, N2, B). [1@1, N, N2]
```

This is an extreme form of regularisation, where the ASP solver is guaranteed to find the *minimum* number of connections.

Calculating activations

The activation values of the input nodes are determined by the input:

```
value(E, N, B) :- bnn_input(E, N, B).
```

The bias nodes are always 1:

```
value(E, N, 1) :- example(E), is_bias(N).
```

For the rest of the nodes in the network, the node is activated if the total sum of inputs x_i that are equal to their weight w_i is greater than or equal to half the number of inputs:

$$\sum_{i=1}^{n} \mathbb{1}[x_i = w_i] \geq \left\lceil \frac{n}{2} \right\rceil$$

This is implemented as:

```
value(E, N, 1) :-
    count_1s(E, N, C),
    threshold_count(N, T),
    C >= T.

value(E, N, 0) :-
    count_1s(E, N, C),
    threshold_count(N, T),
    C < T.
```

The threshold_count predicate checks if $\sum_{i=1}^{n} \mathbb{1}[x_i = w_i] \geq \left\lceil \frac{n}{2} \right\rceil$:

```
threshold_count(N, C/2) :-
    num_inputs(N, C),
    C\2 < C/2.

threshold_count(N, C/2+1) :-
    num_inputs(N, C),
    C\2 >= C/2.
```

The following code counts how many inputs each node has:

```
has_input(N, N2) :- weight(N2, N, _).

count_inputs(node(L, N), node(L-1, 0), 1) :-
    has_input(node(L, N), node(L-1, 0)).

count_inputs(node(L, N), node(L-1, 0), 0) :-
    is_node(node(L, N)),
    is_node(node(L-1, 0)),
    not has_input(node(L, N), node(L-1, 0)).

count_inputs(Output, N, C) :-
    count_inputs(Output, N2, C),
    next_node(N2, N),
    not has_input(Output, N).

count_inputs(Output, N, C+1) :-
    count_inputs(Output, N2, C),
    next_node(N2, N),
    has_input(Output, N).

num_inputs(Output, C) :-
    count_inputs(Output, N, C),
    last_node(N).
```

The following code calculates $\sum_{i=1}^{n} \mathbb{1}[x_i = w_i]$:

```
count_1(E, node(L, N), node(L-1, 0), 1) :-
    example(E),
    is_node(node(L, N)),
    is_node(node(L-1, 0)),
    weight(node(L-1, 0), node(L, N), 1).

count_1(E, node(L, N), node(L-1, 0), 0) :-
    example(E),
    is_node(node(L, N)),
    is_node(node(L-1, 0)),
    not weight(node(L-1, 0), node(L, N), 1).
```

```
count_1(E, Output, N, C) :-
    count_1(E, Output, N2, C),
    next_node(N2, N),
    not check_equality(E, N, Output, 1).

count_1(E, Output, N, C+1) :-
    count_1(E, Output, N2, C),
    next_node(N2, N),
    check_equality(E, N, Output, 1).

count_1s(E, N, C) :-
    count_1(E, N, N2, C),
    last_node(N2).

check_equality(E, N1, N2, B) :-
    weight(N1, N2, W),
    value(E, N1, B2),
    nxor(W, B2, B).

next_node(node(L, N), node(L, N+1)) :-
    is_node(node(L, N)),
    is_node(node(L, N+1)).

last_node(node(L, N)) :-
    is_node(node(L, N)),
    not is_node(node(L, N+1)).

nxor(0, 0, 1).
nxor(0, 1, 0).
nxor(1, 0, 0).
nxor(1, 1, 1).
```

This code uses *linearization* to efficiently calculate the sum $\sum_{i=1}^{n} \mathbb{1}[x_i = w_i]$. Linearization is much more efficient than using ASP's `#count` mechanism [GKKS12].

The source code for the binary neural network is available here:

`https://github.com/RichardEvans/apperception/blob/master/asp/bnn.lp`

5.5 Experiments

We present three groups of experiment: *Seek Whence* from noisy images, *Sokoban*, and fuzzy sequences.

5.5.1 *Seek Whence* with noisy images

The *Seek Whence* dataset is a set of challenging sequence induction problems designed by Douglas Hofstadter [Hof95]. In each problem, you are given a sequence of symbols, and have to predict the next symbol in the sequence. See Section 4.2.3 for details (but here we use sequences of digits rather than sequences of letters).

The data

In Hofstadter's original dataset, the sequences are lists of discrete symbols. In our modified dataset, we replaced each discrete symbol with a corresponding MNIST image.

To make it more interesting (and harder), we deliberately chose particularly ambiguous images. Consider Figure 5.1a. Here, the leftmost image could be a 0 or a 2, while the next could be a 5 or possibly a 6. Of course, we humans are unphased by these ambiguities because the low Kolmogorov complexity [LV08] of the high-level symbolic sequence helps us to resolve the ambiguities in the low-level perceptual input. We would like our machines to do the same.

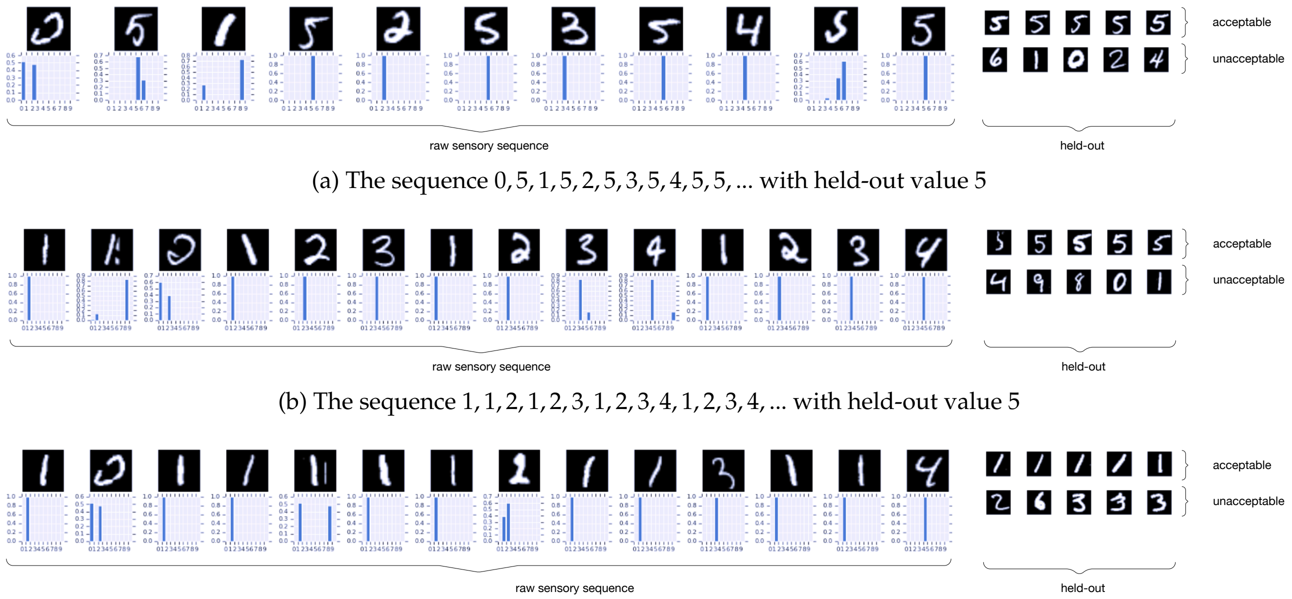
For each sequence, the held-out data used for evaluation is a set of acceptable images, and a set of unacceptable images, for the final held-out time step. See Figure 5.1. We provide a slice of the sequence as input, and use a held-out time step for evaluation. If the correct symbol at the held-out time step is s, then we sample a set of unambiguous images representing s for our set of acceptable next images, and we sample a set of unambiguous images representing symbols other than s for our set of unacceptable images.

The model

In this experiment, we combined the Apperception Engine with a three-layer perceptron with dropout that had been *pre-trained* to classify images into ten classes representing the digits $0 - 9$.[4] For each image, the network produced a probability distribution over the ten classes.

We chose a threshold (0.1), and stipulated that if the probability of a particular digit exceeded the threshold, then the image possibly represents that digit. According to this threshold, some of the images (the first, second, and sixth) of Figure 5.2a are ambiguous, while others (the third, fourth, and fifth) are not.

[4] For experiments in which the network's weights are learned simultaneously with rule-induction, see Section 5.5.2 below.



(a) The sequence $0, 5, 1, 5, 2, 5, 3, 5, 4, 5, 5, ...$ with held-out value 5

(b) The sequence $1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, ...$ with held-out value 5

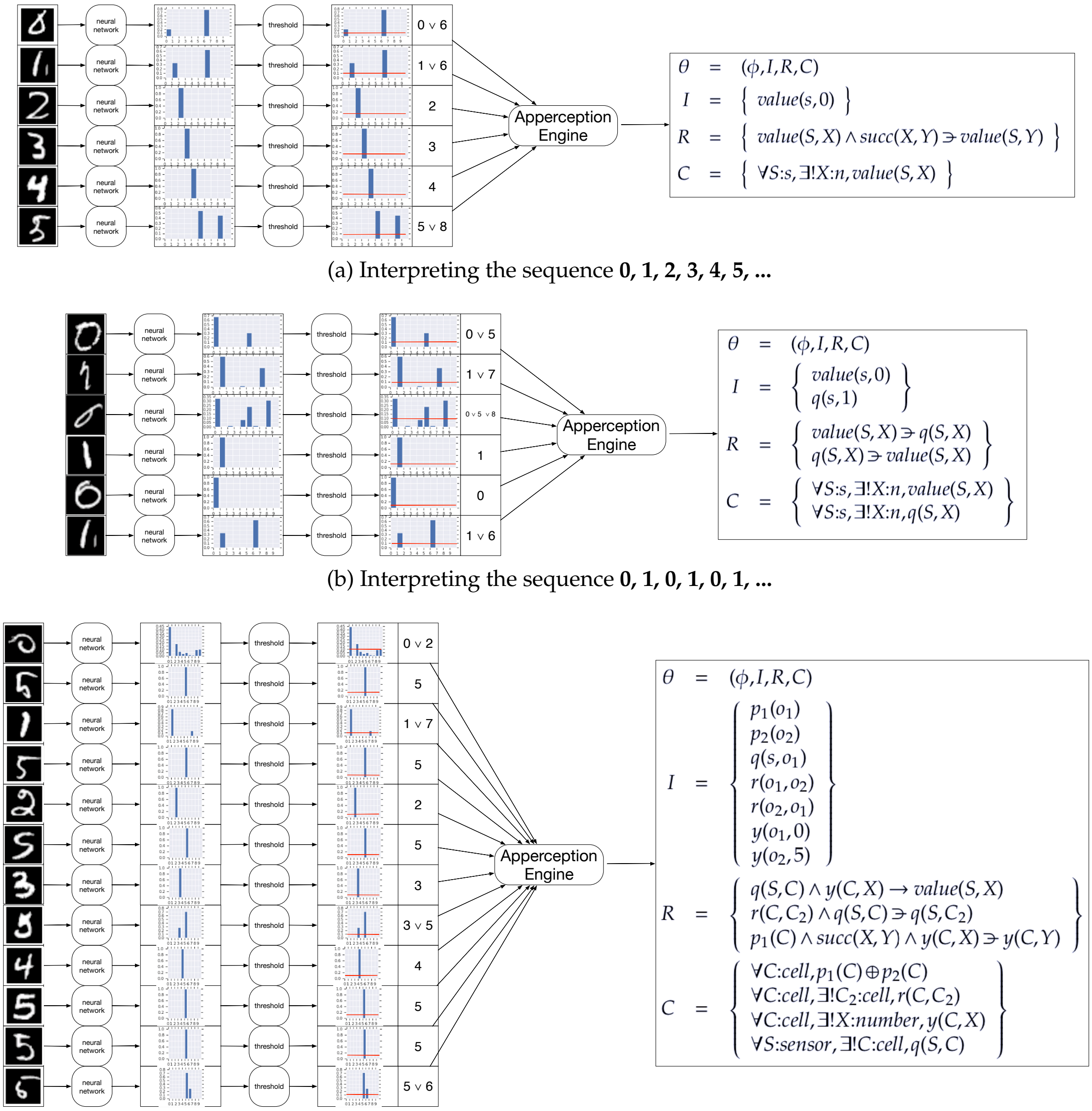
(c) The sequence $1, 0, 1, 1, 1, 1, 1, 2, 1, 1, 3, 1, 1, 4, ...$ with held-out value 1

Figure 5.1: Three *Seek Whence* tasks using MNIST images. The left section of each diagram shows the given sensory sequence, while the right section shows the held-out time step. At the final held-out time step, there is a set of acceptable images, and a set of unacceptable images.

Our pre-trained neural network MNIST classifier has effectively turned the raw apperception task into a disjunctive apperception task. Once the input has been transformed into a sequence of disjunctions, we apply the Apperception Engine to resolve the disjunctions and find a unified theory that explains the sequence.

In terms of the formalism of Section 5.2, the raw input $r = (r_1, ..., r_T)$ is a sequence of MNIST images from $[0, 1]^{28 \times 28}$. The framework $(\pi_{\mathbf{w}}, n, \Delta, \phi, C)$ consists of:

- $\pi_{\mathbf{w}}$, a pre-trained MNIST classifier with frozen weights $\mathbf{w}$
- $n = 10$, representing the digits '0'–'9'
- Δ takes the output of the pre-trained MNIST classifier, and produces a single disjunction of all the classes for which the network outputs a probability that is above the threshold
- A type signature $\phi = (T, O, P, V)$ consisting of two types: sensors and integers, and two predicates: $value(sensor, int)$ representing the numeric value of a sensor, and $succ(int, int)$ representing the successor relation
- C contains one constraint that insists that every sensor always has exactly one numeric value



$$
\begin{aligned}
\theta &= (\phi, I, R, C) \\
I &= \{\ value(s,0)\ \} \\
R &= \{\ value(S,X) \wedge succ(X,Y) \mathrel{\supset\!\!-} value(S,Y)\ \} \\
C &= \{\ \forall S{:}s, \exists ! X{:}n, value(S,X)\ \}
\end{aligned}
$$

(a) Interpreting the sequence **0, 1, 2, 3, 4, 5, ...**

$$
\begin{aligned}
\theta &= (\phi, I, R, C) \\
I &= \left\{ \begin{array}{l} value(s,0) \\ q(s,1) \end{array} \right\} \\
R &= \left\{ \begin{array}{l} value(S,X) \mathrel{\supset\!\!-} q(S,X) \\ q(S,X) \mathrel{\supset\!\!-} value(S,X) \end{array} \right\} \\
C &= \left\{ \begin{array}{l} \forall S{:}s, \exists ! X{:}n, value(S,X) \\ \forall S{:}s, \exists ! X{:}n, q(S,X) \end{array} \right\}
\end{aligned}
$$

(b) Interpreting the sequence **0, 1, 0, 1, 0, 1, ...**

$$
\begin{aligned}
\theta &= (\phi, I, R, C) \\
I &= \left\{ \begin{array}{l} p_1(o_1) \\ p_2(o_2) \\ q(s,o_1) \\ r(o_1,o_2) \\ r(o_2,o_1) \\ y(o_1,0) \\ y(o_2,5) \end{array} \right\} \\
R &= \left\{ \begin{array}{l} q(S,C) \wedge y(C,X) \rightarrow value(S,X) \\ r(C,C_2) \wedge q(S,C) \mathrel{\supset\!\!-} q(S,C_2) \\ p_1(C) \wedge succ(X,Y) \wedge y(C,X) \mathrel{\supset\!\!-} y(C,Y) \end{array} \right\} \\
C &= \left\{ \begin{array}{l} \forall C{:}cell, p_1(C) \oplus p_2(C) \\ \forall C{:}cell, \exists ! C_2{:}cell, r(C,C_2) \\ \forall C{:}cell, \exists ! X{:}number, y(C,X) \\ \forall S{:}sensor, \exists ! C{:}cell, q(S,C) \end{array} \right\}
\end{aligned}
$$

(c) Interpreting the sequence **0, 5, 1, 5, 2, 5, 3, 5, 4, 5, 5, 5, ...**

Figure 5.2: Interpreting *Seek Whence* sequences from raw images. Each MNIST image is passed to a pre-trained neural network classifier that emits a distribution over the digits 0–9. A threshold of 0.1 is applied to the probability distribution, generating a disjunction over the values of the sensor. For example, the disjunction 0 ∨ 6 is short-hand for $value(s,0) \vee value(s,6)$. The sequence of disjunctions is passed to the Apperception Engine, which produces a unified theory that resolves the disjunctions and explains the sequence. The predicates *value* and *succ* are provided to the system, while all other predicates are invented.

The input type signature ϕ and initial constraints C are:

$$\begin{aligned}
\phi &= \begin{cases} T = \{sensor, int\} \\ O = \{s{:}sensor, 0{:}int, 1{:}int, 2{:}int, ...\} \\ P = \{value(sensor, int), succ(int, int)\} \\ V = \{X{:}sensor, N{:}int\} \end{cases} \\
C &= \left\{ \ \forall X{:}sensor,\ \exists! N{:}int\ value(X, N) \ \right\}
\end{aligned}$$

We ran the Apperception Engine on a standard Unix desktop, allowing 4 hours for each sequence. Figure 5.2 shows some results.

Understanding the interpretations

Figure 5.3a shows the unified theory found for the "theme song"' sequence, while Figure 5.3b shows the interpretation in detail.

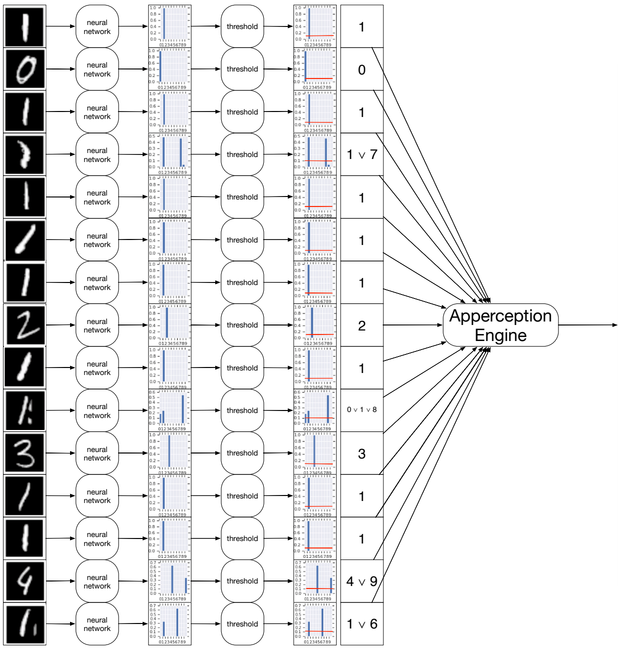
Let us try to understand, in detail, why the Apperception Engine believes the bottom MNIST image in Figure 5.3a (at time step 15) should be interpreted as a '1', rather than a '6'. According to the neural network, the image could either be classified as a '1' or a '6'. In fact, the network thinks it is rather more likely to be a '6'. Nevertheless, the overall assessment of the Apperception Engine is that the image represents a '1'. Why is this?

At a high level, the explanation for this interpretation is that the whole sequence exhibits a particular regularity described by a single general pattern with low Kolmogorov complexity, and that, given this overall structure, the best way to read the final symbol is as a '1' rather than as a '6'.

More specifically, Figure 5.3a describes the following simple process: the sensor is a read-write head that moves between three cells, in a cycle. These cells are o_1, o_2, and o_3, which are placed in the order: o_1, o_3, o_2. Initially, cells o_1 and o_2 have value 1, while cell o_3 has value 0. The head reads the value of the current cell, and writes it onto an output tape. When the head moves over the middle cell (o_3), it increments the value of that cell. When it moves over either of the other two cells, its value remains unchanged.

Note that the only predicates that are given to the Apperception Engine are *value* (provided by the neural network) and the *succ* relation (provided as prior knowledge). Every other predicate is *invented*, its meaning entirely determined by its inferential role in the rules and constraints of the theory in which it is embedded.

Now, at this particular moment (time step 15 of Figure 5.3a), the read-write head is on the cell o_2. This cell has value 1. So the sensor must be reading a '1' rather than a '6'. There is no comparably simple theory that makes sense of the data which resolves the final image to a '6'. So, given the plausibility



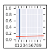
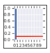
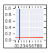
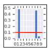
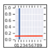
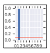
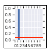
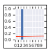
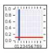
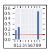
$$\theta = (\phi, I, R, C)$$

$$I = \left\{ \begin{array}{l} p_1(o_3) \\ p_2(o_1) \\ p_2(o_2) \\ q(s, o_1) \\ r(o_1, o_3) \\ r(o_3, o_2) \\ r(o_2, o_1) \\ y(o_1, 1) \\ y(o_2, 1) \\ y(o_3, 0) \end{array} \right\}$$

$$R = \left\{ \begin{array}{l} q(S,C) \wedge y(C,N) \rightarrow value(S,N) \\ p_1(C) \wedge succ(N,N_2) \wedge q(S,C) \wedge y(C,N) \ni y(C,N_2) \\ r(C,C_2) \wedge q(S,C) \ni q(S,C_2) \end{array} \right\}$$

$$C = \left\{ \begin{array}{l} \forall C{:}cell, p_1(X) \oplus p_2(X) \\ \forall C{:}cell, \exists! C_2{:}cell, r(C,C_2) \\ \forall C{:}cell, \exists! N{:}number, y(C,N) \\ \forall S{:}sensor, \exists! C{:}cell, q(S,C) \\ \forall S{:}sensor, \exists! N{:}number, value(S,N) \end{array} \right\}$$

(a) Generating a theory to make sense of the sequence.

	raw input	network output	disjunctive state	overt state	latent state	rules firing
t_1			$value(s,1)$	$value(s,1)$	$q(s,o_1)$ $y(o_1,1)$ $y(o_2,1)$ $y(o_3,0)$	$q(S,C) \wedge y(C,N) \rightarrow value(S,N)$ $r(C,C_2) \wedge q(S,C) \ni q(S,C_2)$
t_2			$value(s,0)$	$value(s,0)$	$q(s,o_3)$ $y(o_1,1)$ $y(o_2,1)$ $y(o_3,0)$	$q(S,C) \wedge y(C,N) \rightarrow value(S,N)$ $p_1(C) \wedge succ(N,N_2) \wedge q(S,C) \wedge y(C,N) \ni y(C,N_2)$ $r(C,C_2) \wedge q(S,C) \ni q(S,C_2)$
t_3			$value(s,1)$	$value(s,1)$	$q(s,o_2)$ $y(o_1,1)$ $y(o_2,1)$ $y(o_3,1)$	$q(S,C) \wedge y(C,N) \rightarrow value(S,N)$ $r(C,C_2) \wedge q(S,C) \ni q(S,C_2)$
t_4			$value(s,1) \oplus value(s,7)$	$value(s,1)$	$q(s,o_1)$ $y(o_1,1)$ $y(o_2,1)$ $y(o_3,1)$	$q(S,C) \wedge y(C,N) \rightarrow value(S,N)$ $r(C,C_2) \wedge q(S,C) \ni q(S,C_2)$
t_5			$value(s,1)$	$value(s,1)$	$q(s,o_3)$ $y(o_1,1)$ $y(o_2,1)$ $y(o_3,1)$	$q(S,C) \wedge y(C,N) \rightarrow value(S,N)$ $p_1(C) \wedge succ(N,N_2) \wedge q(S,C) \wedge y(C,N) \ni y(C,N_2)$ $r(C,C_2) \wedge q(S,C) \ni q(S,C_2)$
t_6			$value(s,1)$	$value(s,1)$	$q(s,o_2)$ $y(o_1,1)$ $y(o_2,1)$ $y(o_3,2)$	$q(S,C) \wedge y(C,N) \rightarrow value(S,N)$ $r(C,C_2) \wedge q(S,C) \ni q(S,C_2)$
t_7			$value(s,1)$	$value(s,1)$	$q(s,o_1)$ $y(o_1,1)$ $y(o_2,1)$ $y(o_3,2)$	$q(S,C) \wedge y(C,N) \rightarrow value(S,N)$ $r(C,C_2) \wedge q(S,C) \ni q(S,C_2)$
t_8			$value(s,2)$	$value(s,2)$	$q(s,o_3)$ $y(o_1,1)$ $y(o_2,1)$ $y(o_3,2)$	$q(S,C) \wedge y(C,N) \rightarrow value(S,N)$ $p_1(C) \wedge succ(N,N_2) \wedge q(S,C) \wedge y(C,N) \ni y(C,N_2)$ $r(C,C_2) \wedge q(S,C) \ni q(S,C_2)$
t_9			$value(s,1)$	$value(s,1)$	$q(s,o_2)$ $y(o_1,1)$ $y(o_2,1)$ $y(o_3,3)$	$q(S,C) \wedge y(C,N) \rightarrow value(S,N)$ $r(C,C_2) \wedge q(S,C) \ni q(S,C_2)$
t_{10}			$value(s,0) \oplus value(s,1) \oplus value(s,8)$	$value(s,1)$	$q(s,o_1)$ $y(o_1,1)$ $y(o_2,1)$ $y(o_3,3)$	$q(S,C) \wedge y(C,N) \rightarrow value(S,N)$ $r(C,C_2) \wedge q(S,C) \ni q(S,C_2)$

(b) The left column shows the raw MNIST images, while the second column shows the output layer of the pre-trained neural network when given as input the MNIST image on the left. The third column shows the disjunctive sensor state, and the fourth column shows the overt state after the disjunctions have been resolved. The fifth column shows the latent state imputed by the Apperception Engine. The sixth column shows the rules that fire at each time step.

Figure 5.3: Interpreting the sequence **1, 0, 1, 1, 1, 1, 1, 2, 1, 1, 3, 1, 1, 4, 1, ...**

(simplicity) of the whole theory that explains all the data, we are compelled to interpret the image as a '1'.

The baseline

Given the raw input to the Apperception Engine, neural models are the most appropriate baselines for comparison.[5] However, the modes of operation of these two systems differ greatly. The Apperception Engine outputs a compact theory which aims to fully explain the sequence, making these rules useful for prediction, imputation, retrodiction as well as explanation. With a neural model, it is hard to induce a verifiably correct and explainable theory. However, we can compare it to the Apperception Engine in terms of their predictive capabilities.

In order to make a fair like-for-like comparison between the neural baseline and the Apperception Engine, we impose the following requirements on the baseline:

- It must learn using *self-supervision*, predicting future time-steps from earlier time-steps.
- It must be able to work with *variable-length data*, since the different trajectories are different lengths.
- It must be able to handle *noisy or ambiguous data*, since the raw data in all three of our experiments is noisy and ambiguous.
- It must be able to work with a *small amount of data*, since the Apperception Engine is able to learn from a handful of data.
- Its inner workings should be *interpretable*. The Apperception Engine outputs a fully explainable model, which we cannot achieve with a neural model. However, we can design a neural model to induce an almost symbolic representation of the next state. This provides explainability at the level of the state, though the state transition function itself remains opaque and inscrutable.

Following these desiderata brings us to the class of auto-regressive models with a relaxed discrete distribution as a bottleneck [MMT16]. We will abide by these desiderata, making slight adjustments per task, to ensure fair comparison and same testing conditions for both systems.

We follow the proposed desiderata for the *Seek Whence* task, though in this instance we do not use a relaxed discrete distribution as a bottleneck. Concretely, this baseline (i) uses a pre-trained MNIST model to classify digits, as does the Apperception Engine, ii) utilises an LSTM model [HS97] as a prediction engine over these digits, and importantly iii) does not represent the distribution of the next state, but directly predicts the next digit. We opted for this approach on the *Seek Whence* task only, since we can utilise the pre-trained MNIST model to produce the target for the next sequence

[5] I am very grateful to Matko Bosnjak for his help in designing and implementing the neural baselines.

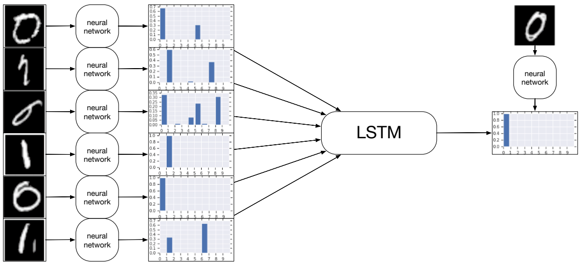


Figure 5.4: Neural baseline for the *Seek Whence* task, utilising a pre-trained digit-recognition network and an LSTM predicting the output representation of the following digit.

element. This is possible because we do not retrain the MNIST model, as training would otherwise lead to unstable learning, leading to degenerate solutions.

The model, depicted in Figure 5.4, is using the same pre-trained MNIST model and an LSTM with 10 hidden units. It is optimised with the Adam optimiser with a learning rate of 0.01. Every experiment is repeated 10 times on different random seeds.

Results

Our *Seek Whence* experiments contained 10 sequences:

0, 0, 0, 0, 0, 0, ...	**0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, ...**
0, 1, 2, 3, 4, 5, ...	**0, 5, 1, 5, 2, 5, 3, 5, 4, 5, 5, 5, ...**
5, 4, 3, 2, 1, 0, ...	**1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, ...**
0, 1, 0, 1, 0, 1, ...	**1, 0, 1, 1, 1, 2, 1, 3, 1, 4, 1, 5, ...**
0, 0, 1, 1, 2, 2, 3, 3, 4, 4, ...	**1, 0, 1, 1, 1, 1, 1, 2, 1, 1, 3, 1, 1, 4, 1, ...**

For each symbolic sequence, we generated multiple MNIST image sequences. To generate an MNIST image sequence, we chose α, the number of ambiguities, and then sampled an image sequence with exactly α ambiguous images. We let α range from 0 to 10. An image counts as ambiguous relative to our threshold of 0.1 if two or more classes are assigned a probability of higher than 0.1 by our pre-trained neural network.

Figure 5.5 shows how accuracy deteriorates as we increase the number of ambiguous images. The interpretations are very robust to a small number of ambiguous images. Eventually, once we have 10 ambiguous images (for sequences of average length 12), the results begin to degenerate, as we would expect. But the key point here is that the Apperception Engine's accuracy is robust to a number of ambiguities.

Comparing with the neural baseline, we can see that the baseline performance also drops with the increasing number of ambiguous images, when trained on a single example, though not as significantly as the Apperception Engine. This problem is fixed with an increasing number of

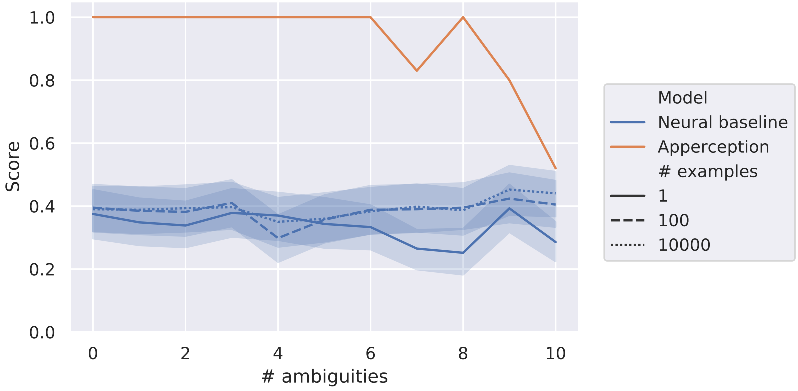


Figure 5.5: The evaluation for the noisy *Seek Whence* sequences from MNIST, for the Apperception Engine and the baseline models. The horizontal axis records the number of ambiguous images in the sequence while the vertical axis records the mean percentage accuracy over the ten sequences. The neural baseline is trained on an increasing number of training examples. The shaded area is the 95% confidence interval across all the sequences and the 10 runs with different random seeds.

training examples, as expected for a neural model, which perform well with noisy inputs. Qualitative analysis of models per sequence shows that the neural baseline can easily learn to predict elements of easy sequences such as the all-zero and the zero-one sequences. However, it struggles with other sequences, correctly predicting only static elements of a sequence, but failing to learn the approximation to the *succ* relation. Though seemingly unfair—requiring a neural model to learn the *succ* relation—we emphasise that any background knowledge needs to be explicitly hard-coded into the architecture of the model necessitating non-trivial modifications per task, as opposed to the Apperception Engine where the addition of background knowledge is straightforward. In addition, model performance is highly dependent on the parameter initialisations, shown by the confidence intervals in Figure 5.5.

5.5.2 *Sokoban*

In Section 5.5.1, we used a hybrid architecture where the output of a *pre-trained* neural network was fed to the Apperception Engine. We assumed that we already knew that the images fell into exactly ten classes (representing the digits $0-9$), and that we had access to a network that already knew how to classify images.

But what if these assumptions fail? What if we are doing pure unsupervised learning and don't know how many classes the inputs fall into? What if we want to jointly train the neural network and solve the apperception problem *at the same time*?

In this next experiment, we combined the Apperception Engine with a neural network, simultaneously learning the weights of the neural network and also finding an interpretable theory that explains the sensory sequence.

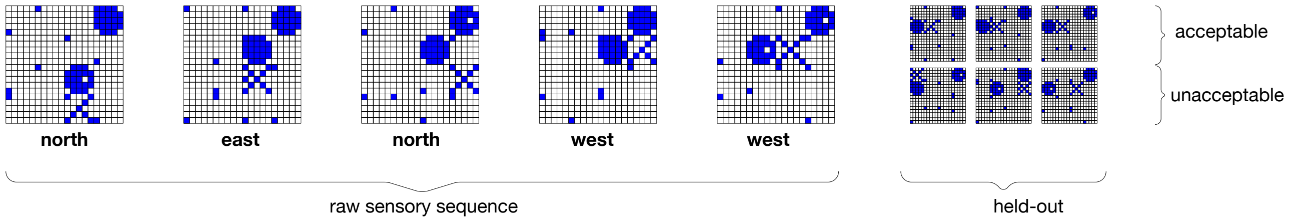


Figure 5.6: The *Sokoban* task. The input is a sequence of (image, action) pairs. For the held-out time step, there is a set of acceptable images, and a set of unacceptable images.

We used *Sokoban* as our domain. This is a puzzle game where the player controls a man who moves around a two-dimensional grid world, pushing blocks onto designated target squares. We generate traces of human play, and ask our system to make sense of the sequence. We chose Sokoban because it is a challenging domain for next-step neural network predictors [BWR+18].

In our version, the system is not given a symbolic representation of the state, but is presented with a sequence of noisy pixel images together with associated actions. The system must jointly (i) parse the noisy pixel images into a set of persistent objects, and (ii) construct a set of rules that explain how the properties of those objects change over time as a result of the actions being performed. We wanted the learned dynamics to be 100% correct. Although next-step prediction models based on neural networks are able, with sufficient data, to achieve accuracy of 99% [BWR+18], this is insufficient for our purposes. If a learned dynamics model is going to be used for long-term planning, 99% is insufficiently accurate, as the roll-outs will become increasingly untrustworthy as we progress through time, since 0.99^t quickly approaches 0 as t increases.

The data

In this task, the raw input is a sequence of pairs containing a binarised 20×20 image together with a player action from $\mathcal{A} = \{north, east, south, west\}$. In other words, $\mathcal{R} = \mathbb{B}^{20 \times 20} \times \mathcal{A}$, and $(r_1, ..., r_T)$ is a sequence of (image, action) pairs from $\mathcal{R}$.

Each array is generated from a 4×4 grid of 5×5 sprites. Each sprite is rendered using a certain amount of noise (random pixel flipping), and so each 20×20 pixel image contains the accumulated noise from the various noisy sprite renderings.

The player actions are treated as *exogenous*: although they can be used by the system to predict the next state, they do not themselves need to be explained.

Each trajectory contains a sequence of (image, action) pairs, plus held-out data for evaluation. Because of the noisy sprite rendering process, there are many possible acceptable pixel arrays for the final held-out time step. These acceptable pixel arrays were generated by taking the true underlying symbolic description of the *Sokoban* state at the held-out time step, and producing many alternative renderings. A set of unacceptable pixel arrays was generated by rendering from various symbolic states distinct from the true symbolic state. Figure 5.6 shows an example.

In our evaluation, a model is considered accurate if it accepts every acceptable pixel array at the held-out time step, and rejects every unacceptable pixel array. This is a stringent test. We do not give partial scores for getting some of the predictions correct.

The model

In outline, we convert the raw input sequence into a disjunctive input sequence by imposing a grid on the pixel array and repeatedly applying a binary neural network to each sprite in the grid. In detail:

1. We choose a sprite size k, and assume the pixel array can be divided into squares of size $k \times k$. We assume all objects fall exactly within grid cell boundaries. In this experiment[6], we set $k = 5$.

2. We choose a number m of persistent objects $o_1, ..., o_m$. We choose a number n of distinct types of objects $v_1, ..., v_n$, and add an additional type v_0 (where v_0 is a distinguished identifier that will be used to indicate that there is *nothing* at a grid square). We choose a total map $\kappa : \{o_1, ..., o_m\} \rightarrow \{v_1, ..., v_n\}$ from objects to types. For example, we might choose three objects ($m = 3$) and two types ($n = 2$), where o_1 is of type v_1, while both o_2 and o_3 are of type v_2.

3. We apply a binary neural network (**BNN**) to each $k \times k$ sprite in the grid. The BNN implements a mapping $\mathbb{B}^{k \times k} \rightarrow \{v_0, v_1, ..., v_n\}$. If sprite s is at (x, y), then $BNN(s) = v_i$ can be interpreted as: it looks as if there is *some object* of type v_i at grid cell (x, y), for $i > 0$. If $BNN(s) = v_0$, it means that there is *nothing* at (x, y). See Figure 5.7. For each time step, for each grid cell, we convert the output of the BNN into a disjunction of ground atoms: if sprite s is at (x, y), and $BNN(s) = v_i$, then we create a disjunction featuring each object o of type v_i stating that *any* of them could be at (x, y). See Figure 5.8.

4. We use the Apperception Engine to solve the disjunctive apperception task generated by steps 1–3.

[6] Giving the system the grid cell boundaries is a substantial piece of information that helps the Apperception Engine overcome the combinatorial complexity of the problem. But for a series of experiments in which we do *not* provide any spatial information, see Section 5.5.3.

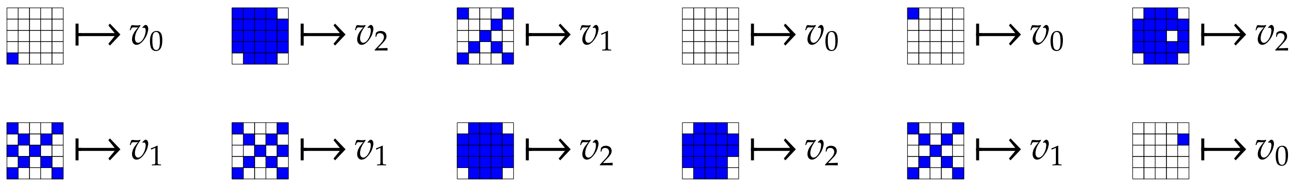
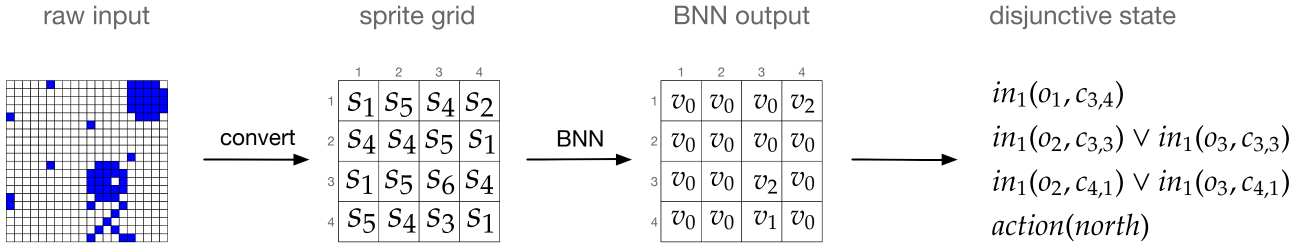


Figure 5.7: A binary neural network maps sprite pixel arrays to types $\{v_0, v_1, v_2\}$.



	1	2	3	4
1	s_1	s_5	s_4	s_2
2	s_4	s_4	s_5	s_1
3	s_1	s_5	s_6	s_4
4	s_5	s_4	s_3	s_1

	1	2	3	4
1	v_0	v_0	v_0	v_2
2	v_0	v_0	v_0	v_0
3	v_0	v_0	v_2	v_0
4	v_0	v_0	v_1	v_0

$in_1(o_1, c_{3,4})$
$in_1(o_2, c_{3,3}) \lor in_1(o_3, c_{3,3})$
$in_1(o_2, c_{4,1}) \lor in_1(o_3, c_{4,1})$
$action(north)$

Figure 5.8: A binary neural network converts the raw pixel input into a set of disjunctions. There is one object o_1 of type v_1 and two objects o_2, o_3 of type v_2. If sprite s is at (x, y), and $BNN(s) = v_i$, then we create a disjunction featuring each object o of type v_i stating that any of them could be at (x, y).

The input type signature ϕ and initial constraints C are:

$$\phi = \begin{cases} T = \{cell, v_1, ..., v_n, d\} \\ O = \begin{cases} c_{x,y} : cell \mid (x, y) \in \{1, 2, 3, 4\} \times \{1, 2, 3, 4\} \\ o_1{:}v_1, ..., o_m{:}v_n \\ north{:}d, east{:}d, south{:}d, west{:}d \end{cases} \\ P = \begin{cases} in_i(v_i, cell) \mid i = 1..n \\ action(d) \\ right(cell, cell) \\ below(cell, cell) \end{cases} \\ V = \{C{:}cell, A{:}d\} \cup \{X_i{:}v_i \mid i = 1..n\} \end{cases}$$

$$C = \begin{cases} \forall X_i{:}v_i,\ \exists! C{:}cell,\ in_i(X, C) \mid i = 1..n \\ \exists! A{:}d, action(A) \end{cases}$$

As background knowledge, we provide the spatial arrangement of the grid cells: $right(c_{1,1}, c_{2,1})$, $below(c_{1,1}, c_{1,2})$, etc.

For each *Sokoban* trajectory, we gave the Apperception Engine 48 hours running on a standard Unix desktop to find the lowest cost interpretation according to the score of Definition 27.

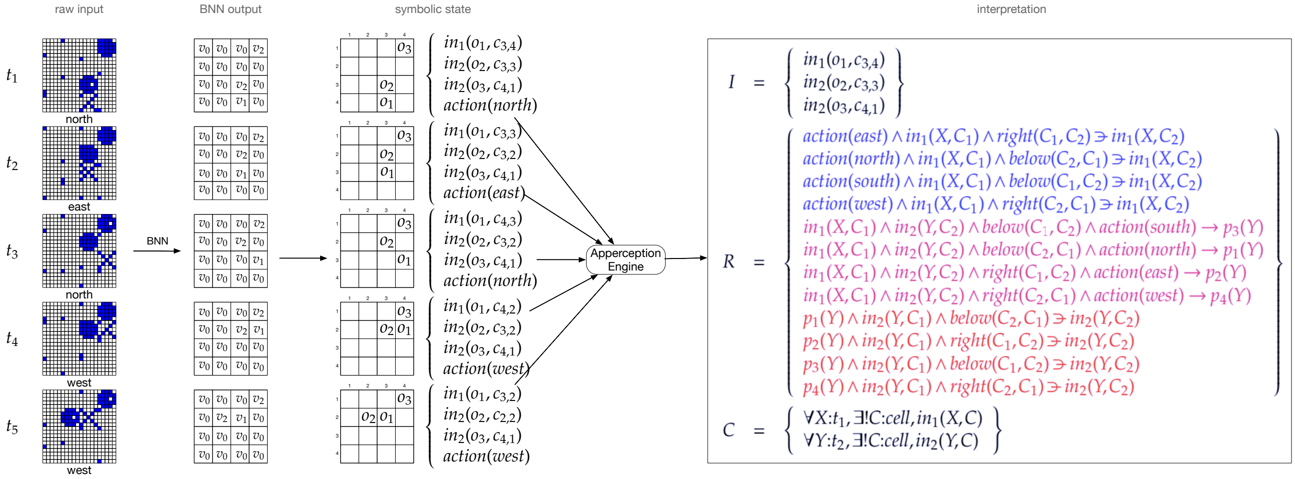

raw input
BNN output
symbolic state
interpretation
t_1
t_2
t_3
t_4
t_5
north
east
north
west
west
BNN
$in_1(o_1, c_{3,4})$ $in_2(o_2, c_{3,3})$ $in_2(o_3, c_{4,1})$ $action(north)$
$in_1(o_1, c_{3,3})$ $in_2(o_2, c_{3,2})$ $in_2(o_3, c_{4,1})$ $action(east)$
$in_1(o_1, c_{4,3})$ $in_2(o_2, c_{3,2})$ $in_2(o_3, c_{4,1})$ $action(north)$
$in_1(o_1, c_{4,2})$ $in_2(o_2, c_{3,2})$ $in_2(o_3, c_{4,1})$ $action(west)$
$in_1(o_1, c_{3,2})$ $in_2(o_2, c_{2,2})$ $in_2(o_3, c_{4,1})$ $action(west)$
Apperception Engine
$I = \{ in_1(o_1, c_{3,4}), in_2(o_2, c_{3,3}), in_2(o_3, c_{4,1}) \}$
$R =$
$action(east) \wedge in_1(X, C_1) \wedge right(C_1, C_2) \ni in_1(X, C_2)$
$action(north) \wedge in_1(X, C_1) \wedge below(C_2, C_1) \ni in_1(X, C_2)$
$action(south) \wedge in_1(X, C_1) \wedge below(C_1, C_2) \ni in_1(X, C_2)$
$action(west) \wedge in_1(X, C_1) \wedge right(C_2, C_1) \ni in_1(X, C_2)$
$in_1(X, C_1) \wedge in_2(Y, C_2) \wedge below(C_1, C_2) \wedge action(south) \rightarrow p_3(Y)$
$in_1(X, C_1) \wedge in_2(Y, C_2) \wedge below(C_2, C_1) \wedge action(north) \rightarrow p_1(Y)$
$in_1(X, C_1) \wedge in_2(Y, C_2) \wedge right(C_1, C_2) \wedge action(east) \rightarrow p_2(Y)$
$in_1(X, C_1) \wedge in_2(Y, C_2) \wedge right(C_2, C_1) \wedge action(west) \rightarrow p_4(Y)$
$p_1(Y) \wedge in_2(Y, C_1) \wedge below(C_2, C_1) \ni in_2(Y, C_2)$
$p_2(Y) \wedge in_2(Y, C_1) \wedge right(C_1, C_2) \ni in_2(Y, C_2)$
$p_3(Y) \wedge in_2(Y, C_1) \wedge below(C_1, C_2) \ni in_2(Y, C_2)$
$p_4(Y) \wedge in_2(Y, C_1) \wedge right(C_2, C_1) \ni in_2(Y, C_2)$
$C = \{ \forall X{:}t_1, \exists! C{:}cell, in_1(X, C); \forall Y{:}t_2, \exists! C{:}cell, in_2(Y, C) \}$

Figure 5.9: Interpreting *Sokoban* from raw pixels. Raw input is converted into a sprite grid, which is converted into a grid of types v_0, v_1, v_2. The grid of types is converted into a disjunctive apperception task. The Apperception Engine finds a unified theory explaining the disjunctive input sequence, a theory which explains how objects' positions change over time. The top four rules of R (in blue) describe how the man X moves when actions are performed. The middle four rules (in magenta) define four invented predicates $p_1, ...p_4$ that are used to describe when a block is being pushed in one of the four cardinal directions. The bottom four rules (in red) describe what happens when a block is being pushed in one of the four directions.

Understanding the interpretations

Figure 5.9 shows the best theory[7] found by the Apperception Engine from one trajectory of 17 time steps. When neural network next-step predictors are applied to these sequences, the learned dynamics typically fail to generalise correctly to different-sized worlds or worlds with a different number of objects [BWR+18]. But the theory leaned by the Apperception Engine applies to all *Sokoban* worlds, no matter how large, no matter how many objects. Not only is this learned theory correct, but it is provably correct.[8]

Figure 5.10 shows the evolving state over time. The grid on the left is the raw perceptual input, a grid of 20 × 20 pixels. The second element is the output of the binary neural network: a 4×4 grid of predicates v_0, v_1, v_2. If v_i is at (x, y), this means "it looks as if there is *some* object of type i at (x, y)" (but we don't yet know which particular object). So, for example, the grid in the top row states that there is some object of type 1 at $(3, 4)$, and some object of type 2 at $(4, 1)$. Here, v_0 is a distinguished predicate meaning there is *nothing* at this grid square.

[7] The rules in R describe the state transitions conditioned on actions being performed. They do not describe the conditions under which the particular actions are available. For example, in *Sokoban*, you cannot push a block left if there is another block to the left of the block that you are trying to push. Action availability is not represented explicitly in the theory $\theta = (\phi, I, R, C)$.

[8] The fixed rules of Sokoban determine a deterministic state transition function $tr : S \times A \rightarrow S$ from board states and actions to board states. We can show that, for any board state S and action A, if $\tau(\theta)$ contains $S \cup A$ at time t, then $\tau(\theta)$ contains $tr(S, A)$ at time $t + 1$.

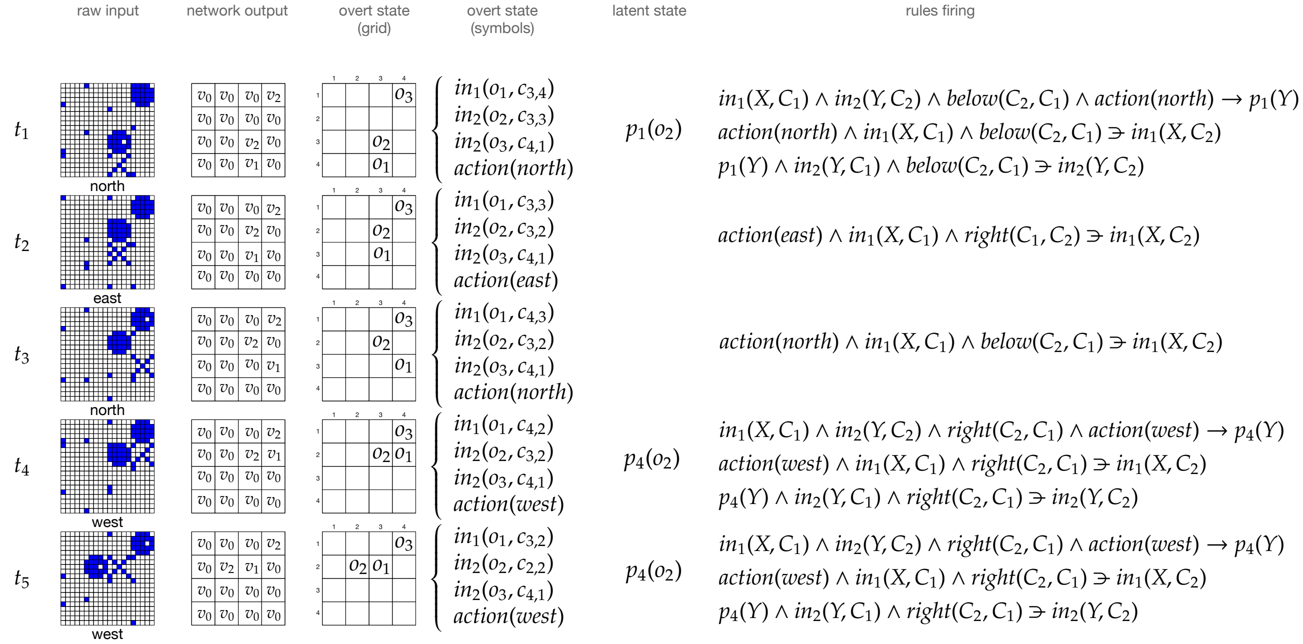

raw input
network output
overt state (grid)
overt state (symbols)
latent state
rules firing
t_1
north
$v_0\ v_0\ v_0\ v_2$ / $v_0\ v_0\ v_0\ v_0$ / $v_0\ v_0\ v_2\ v_0$ / $v_0\ v_0\ v_1\ v_0$
o_3 o_2 o_1
$in_1(o_1, c_{3,4})$
$in_2(o_2, c_{3,3})$
$in_2(o_3, c_{4,1})$
$action(north)$
$p_1(o_2)$
$in_1(X, C_1) \land in_2(Y, C_2) \land below(C_2, C_1) \land action(north) \rightarrow p_1(Y)$
$action(north) \land in_1(X, C_1) \land below(C_2, C_1) \ni in_1(X, C_2)$
$p_1(Y) \land in_2(Y, C_1) \land below(C_2, C_1) \ni in_2(Y, C_2)$
t_2
east
$v_0\ v_0\ v_0\ v_2$ / $v_0\ v_0\ v_2\ v_0$ / $v_0\ v_0\ v_1\ v_0$ / $v_0\ v_0\ v_0\ v_0$
o_3 o_2 o_1
$in_1(o_1, c_{3,3})$
$in_2(o_2, c_{3,2})$
$in_2(o_3, c_{4,1})$
$action(east)$
$action(east) \land in_1(X, C_1) \land right(C_1, C_2) \ni in_1(X, C_2)$
t_3
north
$v_0\ v_0\ v_0\ v_2$ / $v_0\ v_0\ v_2\ v_0$ / $v_0\ v_0\ v_0\ v_1$ / $v_0\ v_0\ v_0\ v_0$
o_3 o_2 o_1
$in_1(o_1, c_{4,3})$
$in_2(o_2, c_{3,2})$
$in_2(o_3, c_{4,1})$
$action(north)$
$action(north) \land in_1(X, C_1) \land below(C_2, C_1) \ni in_1(X, C_2)$
t_4
west
$v_0\ v_0\ v_0\ v_2$ / $v_0\ v_0\ v_2\ v_1$ / $v_0\ v_0\ v_0\ v_0$ / $v_0\ v_0\ v_0\ v_0$
o_3 o_2 o_1
$in_1(o_1, c_{4,2})$
$in_2(o_2, c_{3,2})$
$in_2(o_3, c_{4,1})$
$action(west)$
$p_4(o_2)$
$in_1(X, C_1) \land in_2(Y, C_2) \land right(C_2, C_1) \land action(west) \rightarrow p_4(Y)$
$action(west) \land in_1(X, C_1) \land right(C_2, C_1) \ni in_1(X, C_2)$
$p_4(Y) \land in_2(Y, C_1) \land right(C_2, C_1) \ni in_2(Y, C_2)$
t_5
west
$v_0\ v_0\ v_0\ v_2$ / $v_0\ v_2\ v_1\ v_0$ / $v_0\ v_0\ v_0\ v_0$ / $v_0\ v_0\ v_0\ v_0$
o_3 o_2 o_1
$in_1(o_1, c_{3,2})$
$in_2(o_2, c_{2,2})$
$in_2(o_3, c_{4,1})$
$action(west)$
$p_4(o_2)$
$in_1(X, C_1) \land in_2(Y, C_2) \land right(C_2, C_1) \land action(west) \rightarrow p_4(Y)$
$action(west) \land in_1(X, C_1) \land right(C_2, C_1) \ni in_1(X, C_2)$
$p_4(Y) \land in_2(Y, C_1) \land right(C_2, C_1) \ni in_2(Y, C_2)$

Figure 5.10: The state evolving over time. Each row shows one time step. We show the raw pixel input, the output of the binary neural network, the set of ground atoms that are currently true, and the rules that fire.

The third element is a 4×4 grid of persistent objects: if o_i is at (x, y) this means: the particular persistent object o_i is at (x, y). The fourth element is a set of ground atoms. This is a re-representation of the persistent object grid (the fourth element) together with an atom representing the player's action. The fifth element shows the latent state. In *Sokoban*, the latent state stores information about which objects are being pushed in which directions. Here, in the top row, $p_1(o_2)$ means that persistent object o_2 is being pushed up. The sixth element shows which rules fire in which situations. In the top row, three rules fire. The first rule describes how the man moves when the *north* action is performed. The second rule concludes that a block is pushed northwards if a man is below the block and the man is moving north. The third rule describes how the block moves when it is pushed northwards.

Looking at how the engine interprets the sensory sequence, it is reasonable—in fact, we claim, inevitable—to attribute *beliefs* to the system. In the top row of Figure 5.9, for example, the engine believes that the object at $(3, 3)$ is the same type of thing as the object at $(4, 1)$, while the object at $(3, 4)$ is not the same type of thing as the object at $(4, 1)$. As well as beliefs about particular situations, the system also has general beliefs that apply to all situations. For example, whenever the *north* action is performed, and the man is below a block, then the block is pushed upwards. One of the reasons for using a purely declarative language such as Datalog$^{\ni}$ is that individual atoms and clauses can be interpreted as beliefs. If, on the other hand, the program that generated the trace had been a procedural program, it would have been much less clear what beliefs, if any, the procedure represented.

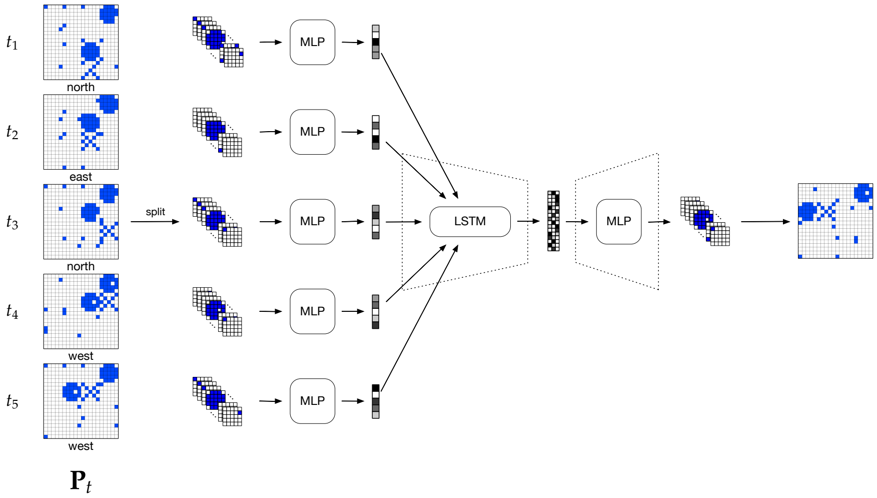


Figure 5.11: The baseline model for the Sokoban task. The perceptual input data is per-sprite fed into an array of parameter-sharing MLPs and then concatenated with the action data. The result is fed into an LSTM which predicts the parameters of an array of Gumbel-Softmaxes [JGP16, MMT16], one per sprite. These distributions approximate a symbolic state of the board, which is then decoded back into perceptual data of the following step.

The baseline

The baseline we construct for the Sokoban task is an auto-regressive model with a continuously-relaxed discrete bottleneck [MMT16], fully following the desiderata of Section 5.5.1.[9]

The model applies an array of parameter-sharing multilayer perceptrons (MLPs) to each block of the game state, and concatenates the result with the one-hot representation of the actions before feeding it into an LSTM. The LSTM, combined with a dense layer, produces the parameters of Gumbel-Softmax [JGP16, MMT16] continuous approximations of the categorical distribution, one per each block of the state. These distributions, when the model is learned well, can encode a close-to-symbolic representation of the current state without direct supervision. The step following is a decoder network consisting of a two-layer perceptron which targets the next raw state of the sequence.

Given that the presented model is a purely generative model over a large state space, in order to compare it to the Apperception Engine, we add a density estimation classifier on its output. The classifier fits a Gaussian per class, trained on log-probabilities of independently sampled acceptable and unacceptable test states calculated over the Bernoulli distribution outputted by the model.

We trained the baseline with the Adam optimizer, varying the learning rate in $[0.05, 0.01, 0.005, 0.001]$, batch size in $[512, 1024]$ and executing each experiment 10 times. We selected the best set of hyper-parameters by choosing the parameters with the best development set performance, and averaged the performance across 10 repetitions with different random seeds. During training, we annealed

[9] I am very grateful to Matko Bosnjak for his help in designing and implementing the neural baselines.

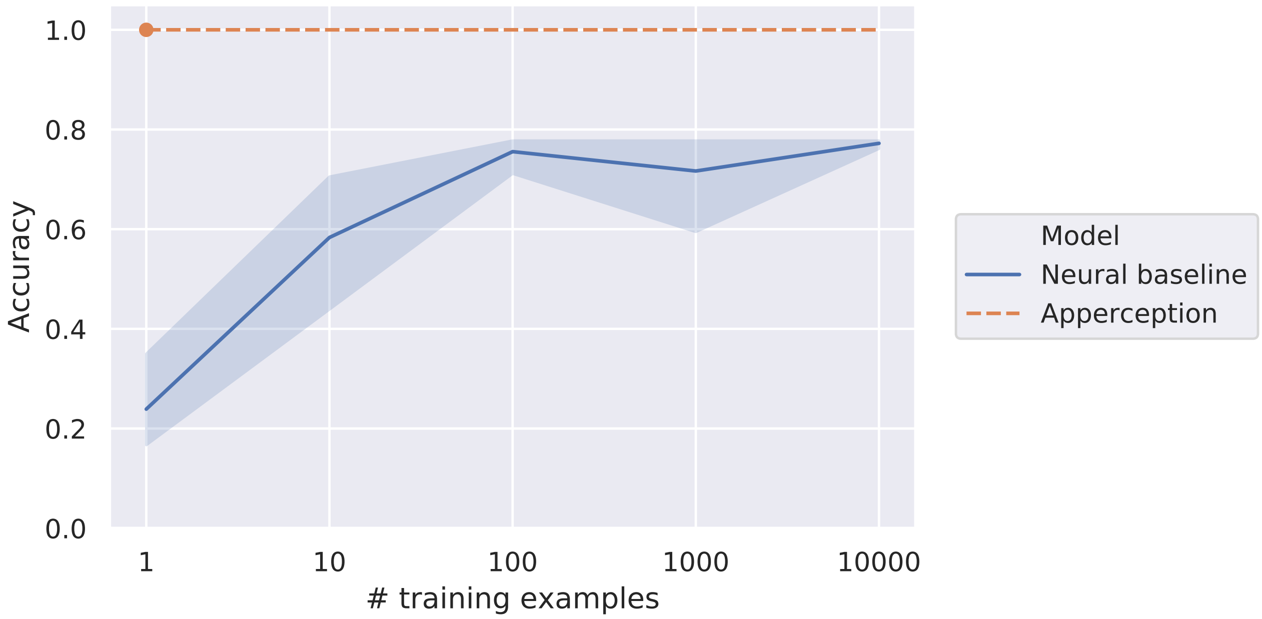


Figure 5.12: The results on the Sokoban task. Apperception is trained on only a single example and the dashed line represents the apperception results on only that example. The neural baseline is trained on an increasing number of training examples. The shaded area is the 95% confidence interval on 10 runs with different random seeds.

the temperature of the Gumbel-Softmax with an exponential decay, from 2.0 to 0.5 with a per-epoch decay of 0.0009.

Results

We took ten trajectories of length 22. For each trajectory, we evaluated on eight subsequences of lengths 3 to 17 in increments of 2. For each subsequence of length n, we used the remaining $22 - n$ time-steps for evaluation. The results are shown in Figure 5.13. While most of the trajectories do not contain enough information for the engine to extract a correct theory, three of them are able to achieve 100% accuracy on the held-out portion of the trajectory. Of course, getting complete accuracy on the held-out portion of a single trajectory is necessary, but not sufficient, to confirm that the induced theory is actually correct on all possible Sokoban configurations. We checked each of the three accurate induced theories, and verified by inspection that one of the three theories was correct on all possible Sokoban maps, no matter how large, and no matter how many objects.[10]

Next, we compare the Apperception Engine to the neural baseline. We train both models on a single trajectory containing enough information to extract the correct theory. In addition, we train the neural baseline on an increasingly large training set.

[10] Note that state of the art ILP systems are unable to learn the correct dynamics of Sokoban given hundreds of trajectories [CEL19].

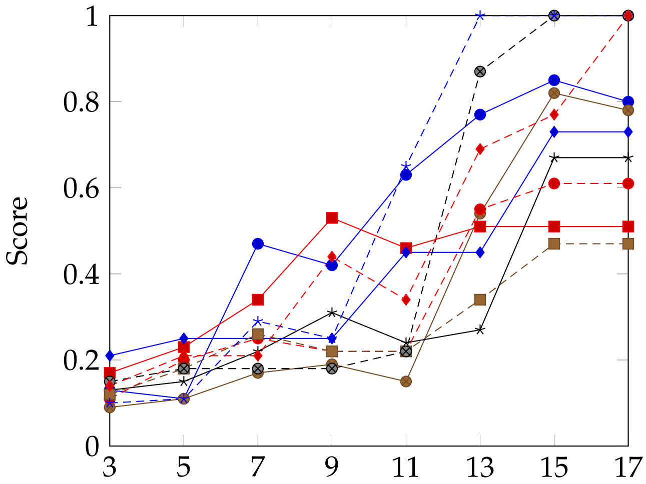


Figure 5.13: The results for *Sokoban* on ten trajectories. The horizontal axis records the number of time-steps provided as input. The vertical axis records the mean percentage accuracy over the held-out time-steps.

The baseline model is not able to absolutely correctly distinguish between acceptable and unacceptable next steps, neither from the single example, nor a large number of examples. However, as expected, the accuracy of the baseline increases with increasing size of the training set, though it shows the tendency to plateau without reaching the maximum. By inspecting the latent distributions, we see that the model learns to approximate the symbolic state of the board well—the resulting distribution roughly corresponds to the state, though the visual inspection of the decoded state shows that the model largely focuses on the large objects (such as the block O) the best, while possibly ignoring smaller objects (such as the man X). An important thing to emphasise here is that the performance of the model is highly dependent on the initial random seed: with some random initialisations, the performance is acceptable, with others unacceptable. From these findings, we conclude that the neural networks can somewhat learn to predict the next state, and even induce a near-to-symbolic representation of the state, though the model requires a larger number of training instances and the performance of the model is not fully reliable.

In contrast, the Apperception Engine is able to learn a fully explainable theory from a single example.

5.5.3 Fuzzy sequences

In the *Sokoban* experiments described in Section 5.5.2 above, the system jointly solved low-level perception and high-level apperception. It performed low-level perception by finding the weights of the binary neural network and it performed high-level apperception by finding a unified theory that solves the apperception task. Because both tasks were encoded as a single SAT problem, and solved jointly, information could flow in both directions, both bottom-up and top-down.

But there were two pieces of domain-specific knowledge that we injected: the dimensions of the sprite

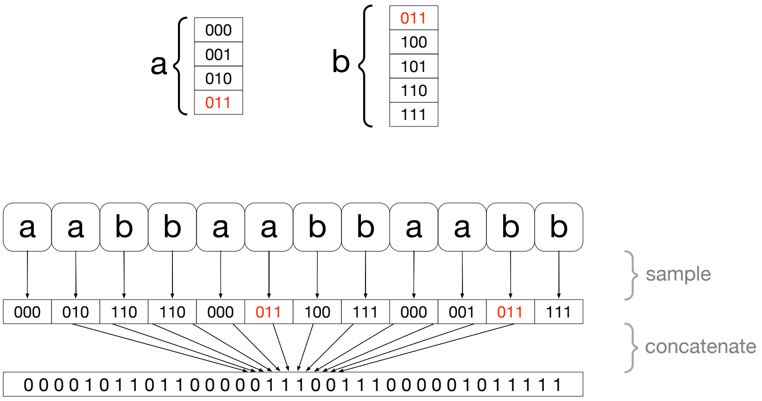


Figure 5.14: Generating fuzzy sequences. We start with a symbolic sequence, and convert into a binary vector. We use a map from discrete symbols to sets of binary vectors. Some of the binary vectors are ambiguous between different symbols; these are shown in red. For each symbol in the original symbolic sequence, we sample one of the corresponding vectors using the map. Finally, we concatenate the binary vectors to produce one large sequence where the segmentations have been thrown away.

grid and the number of distinct types of objects. In this final set of experiments, we investigate what happens when we jointly solve low-level perception and high-level apperception *without* providing a spatial structure or any hint as to the number of classes.

The data

In these experiments, the inputs are binary sequences that were generated by a stochastic process from an underlying symbolic sequence with low Kolmogorov complexity. See Figure 5.14. We start with a simple symbolic sequence, e.g., *aabbaabbaabb*... We generate a map from symbols to sets of binary vectors. This map contains some ambiguities, some binary vectors that are associated with multiple symbols: in Figure 5.14, for example, 011 is ambiguous between *a* and *b*. We convert the sequence of symbols into a sequence of binary vectors by sampling (uniformly randomly) for each symbol in the sequence one of the corresponding vectors. Then we concatenate the binary vectors into one large sequence, thus throwing away the information about where the sequence is segmented. Figure 5.15 shows six example sequences.

We want the Apperception Engine to recover the underlying symbolic structure from this fuzzy, ambiguous sequence, without giving it privileged access to the segmentation information—we want the system to recover the segmentation information as part of the perceptual process.

The held-out data

To evaluate the accuracy of the models, we consider what the model predicts about a held-out portion of the sequence. Because the sequence is ambiguous, there are many different acceptable

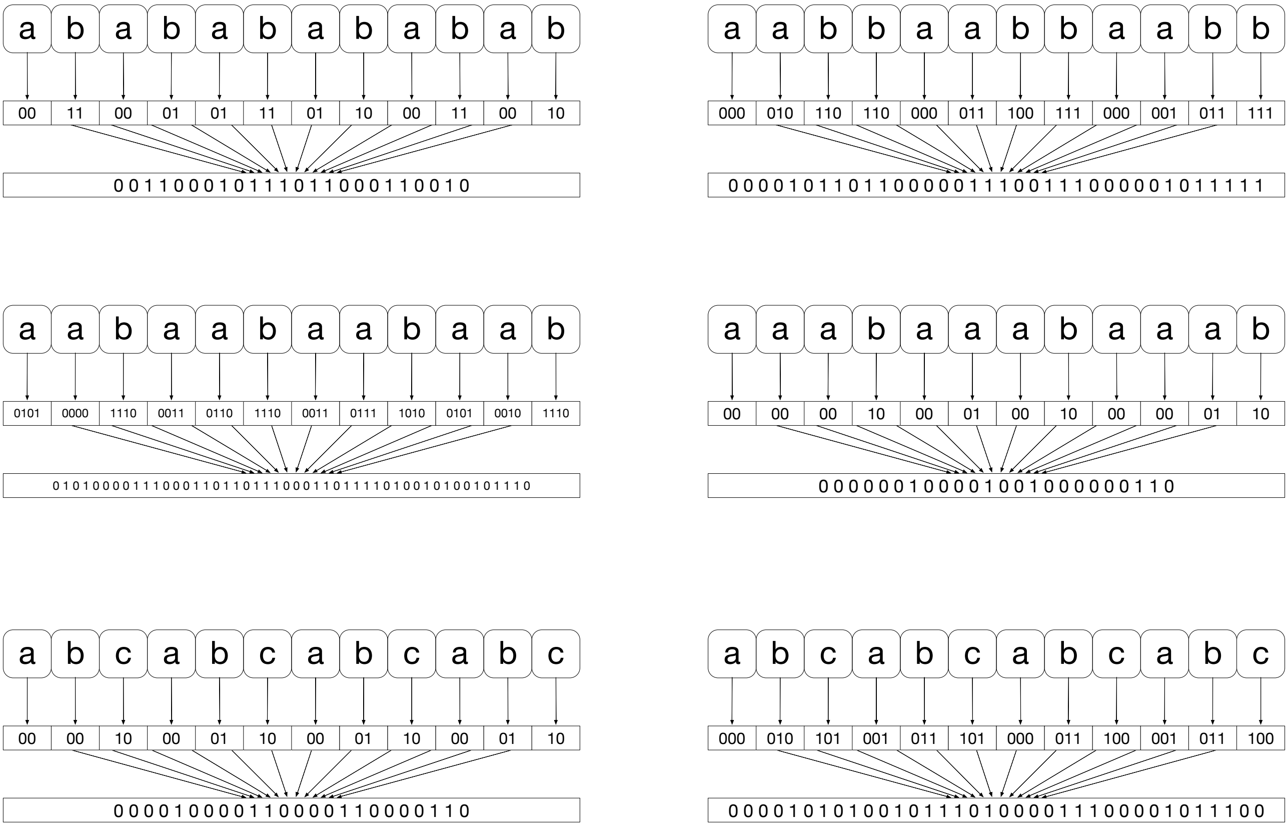


Figure 5.15: Six example sequences. In each, we show the original symbolic sequence, the randomly sampled vectors, and the final concatenated result.

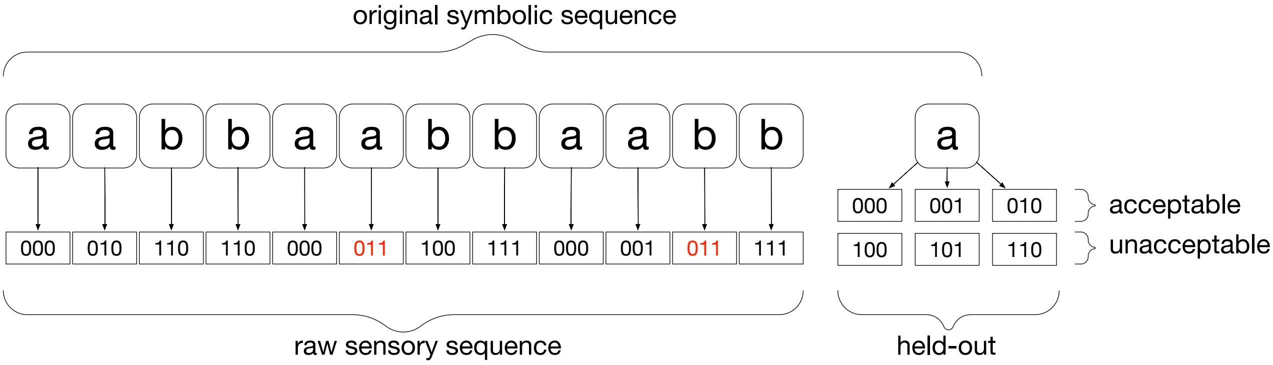


Figure 5.16: A fuzzy sequence with held-out data. Ambiguous vectors are shown in red.

continuations of it (see Figure 5.16).

We evaluate a model as accurate on the sequence if it accepts *every* correct continuation and rejects *every* incorrect one, stringently giving no partial credit.

The models

To find the best interpretation of a fuzzy sequence, we consider a set of models, and find the one with the highest probability (see Definition 27). Each model is an Apperception Engine combined with a binary neural network.

Recall that the given binary sequence is formed by starting with a symbolic sequence $(S_1, ..., S_T)$ from an alphabet of size n, then sampling, for each S_i, a binary vector of length k, and then concatenating the vectors together to produce a single binary vector of size $T \times k$.

We withhold certain crucial information from our model: we do not provide the size k of the constituent binary vectors, and we do not provide the number n of symbols in the alphabet. Instead, we perform a grid search over pairs (k_g, n_g), where k_g is the guessed value of k and n_g is the guessed value of n, and choose the pair with the best score.

For a particular (k_g, n_g) pair, we divide the given binary sequence into $T \cdot k/k_g$ vectors $\mathbf{v}_1, ..., \mathbf{v}_{T \cdot k/k_g}$ each of size k_g, and create a binary neural network with output layer of size n_g. We apply the binary neural network to each vector. We use a type signature with n_g unary predicates $p_1, ..., p_{n_g}$.

We apply the network to each vector $\mathbf{v}_1, ..., \mathbf{v}_{T \cdot k/k_g}$ and generate, for each $\mathbf{v}_t$, a disjunction $p_{x_1}(s) \vee ... \vee p_{x_m}(s)$ holding at time t, where $\{p_{x_1}, ..., p_{x_m}\}$ are the subset of predicates $\{p_1, ..., p_{n_g}\}$ such that the network's x_i'th output is 1. For example, if $n_g = 5$ and the neural network's output layer is $(1, 0, 1, 1, 0)$ on input $\mathbf{v}_t$, then the disjunction $p_1(s) \vee p_3(s) \vee p_4(s)$ is added at time t. The s is a distinguished constant representing the (single) sensor.

In terms of the formalism of Section 5.2, the raw input sequence is a sequence $(r_1, ..., r_T)$ of binary vectors from $\mathbb{B}^k$. The framework $(\pi_{\mathbf{w}}, n, \Delta, \phi, C)$ consists of:

- A binary neural network $\pi_{\mathbf{w}}$ mapping $\mathbb{B}^k \rightarrow \mathbb{B}^n$
- A number n of classes
- A "disjunctifier" Δ that translates the output of $\pi_{\mathbf{w}}$ into a single disjunction of ground atoms
- A type signature $\phi = (T, O, P, V)$ consisting of one type, *sensor*, one object s of type sensor, and predicates $p_1, .., .p_n$ for each of the output classes.
- C contains the constraint that every sensor satisfies exactly one of the p_i predicates

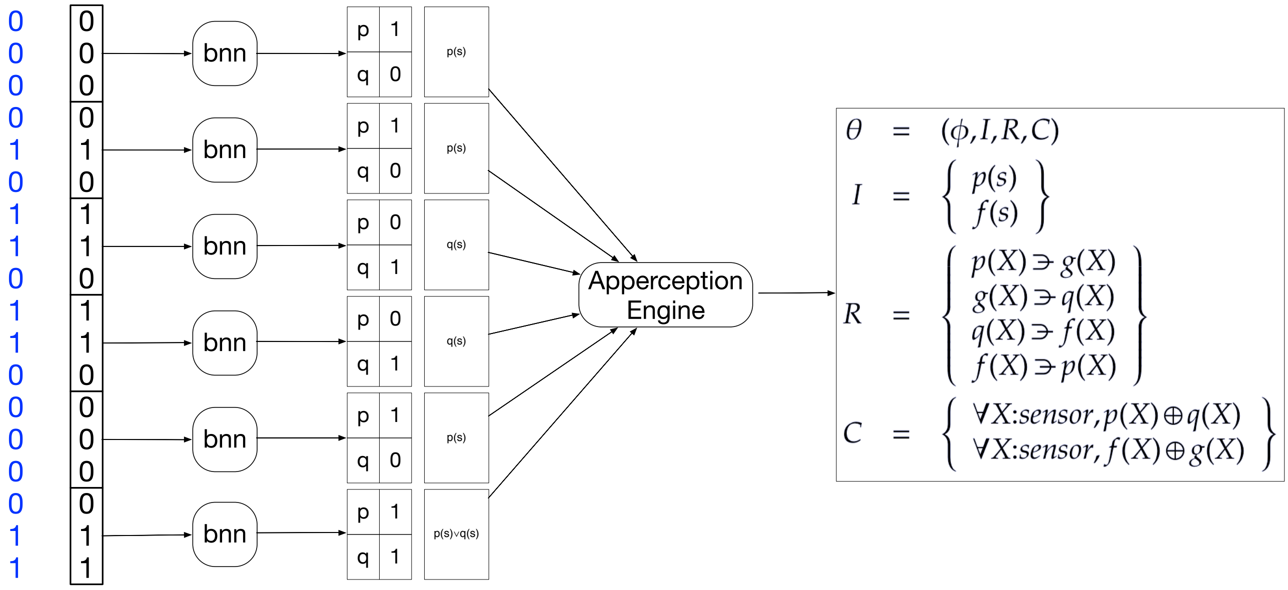

0 0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 1 1
0 0 0
0 1 0
1 1 0
1 1 0
0 0 0
0 1 1
bnn
bnn
bnn
bnn
bnn
bnn
p 1 q 0
p 1 q 0
p 0 q 1
p 0 q 1
p 1 q 0
p 1 q 1
p(s)
p(s)
q(s)
q(s)
p(s)
p(s)∨q(s)
Apperception Engine
$\theta = (\phi, I, R, C)$
$I = \left\{ \begin{array}{l} p(s) \\ f(s) \end{array} \right\}$
$R = \left\{ \begin{array}{l} p(X) \ni g(X) \\ g(X) \ni q(X) \\ q(X) \ni f(X) \\ f(X) \ni p(X) \end{array} \right\}$
$C = \left\{ \begin{array}{l} \forall X{:}sensor, p(X) \oplus q(X) \\ \forall X{:}sensor, f(X) \oplus g(X) \end{array} \right\}$

Figure 5.17: Solving the fuzzy sequence with $k_g = 3$ and $n_g = 2$ (the correct guesses). The interpretation discerns the underlying pattern *ppqqppqqppqq*... which is isomorphic to the original symbolic sequence *aabbaabbaabb*...

In a little more detail, the binary neural network $\pi_{\mathbf{w}}$, parameterised by weights $\mathbf{w}$, takes a binary vector of length k and maps it to a binary vector of length n. Now the disjunctifier Δ uses the binary neural network's output to generate a disjunction: if the i'th output is 1, then the sensor s could satisfy p_i:

$$\Delta = \left\{ \ \bigvee \{ p_i(s) \mid \pi_{\mathbf{w}}(r)[i] = 1 \} \ \right\}$$

Figures 5.17 and 5.18 provide two examples. In Figure 5.17, the guesses are correct as $k_g = k$ and $n_g = n$. Here, each vector $\mathbf{v}_i$ is of length 3, just as in the true generative process, and there are two predicates p and q corresponding to the two symbols of the original symbolic sequence. In Figure 5.18, the guesses are incorrect as $k_g = 2 \neq k$ and $n_g = 3 \neq n$.

Repeated application of the binary neural network to the vectors $\mathbf{v}_1, ..., \mathbf{v}_{T \cdot k/k_g}$ produces $T \cdot k/k_g$ disjunctions of the form $p_{x_1}(s) \vee ... \vee p_{x_m}(s)$. See the fifth column in Figures 5.17 and 5.18. The disjunctive sensory sequence is passed to the Apperception Engine which attempts to resolve the disjunctions and find a unified interpretation. (Note that strictly speaking there is no temporal sequence here: the weights of the binary neural network, the resolutions of the disjunctions, and the unified theory are found jointly and simultaneously. But for expository purposes, it can be helpful to think of the binary neural network as operating before the Apperception Engine).

Once we have chosen k_g and n_g, we create an initial type signature with n_g unary predicates $p_1, ..., p_{n_g}$, and then iterate through increasingly complex templates, with an increasingly large number of invented predicates and additional rules. This iterative procedure produces a set of theories that need to be compared. From each pair of guesses of k_g and n_g, we find a vector w of network weights

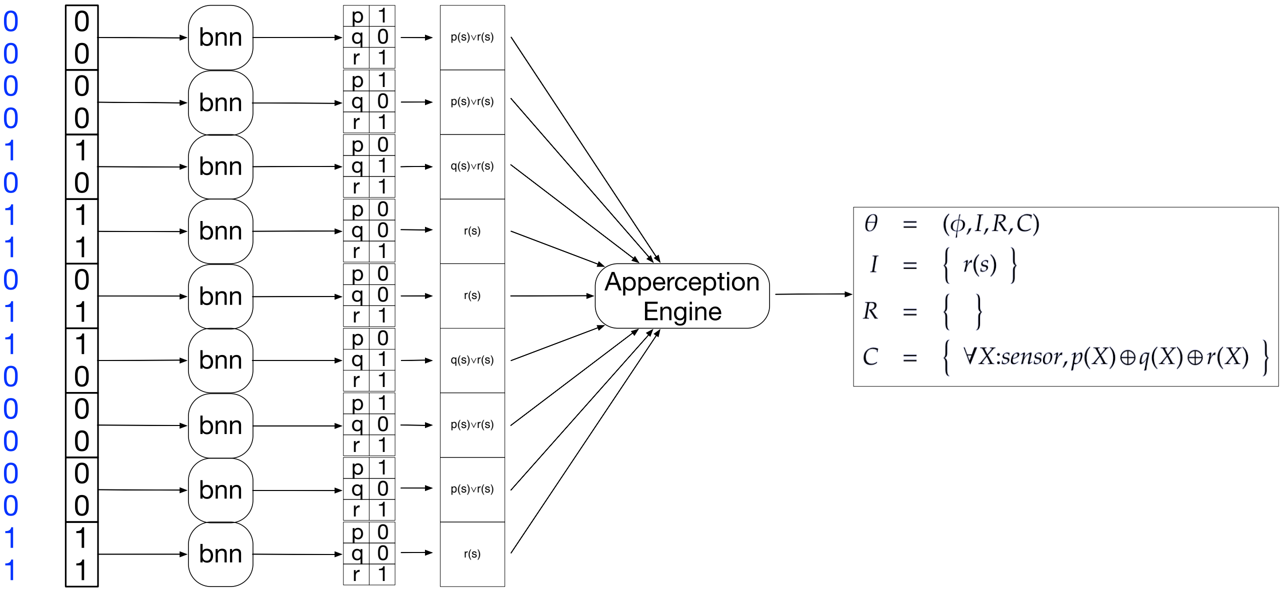


Figure 5.18: Solving the fuzzy sequence with $k_g = 2$ and $n_g = 3$ (the wrong guesses). The interpretation maps all vectors to c and produces a degenerate interpretation in which $r(s)$ remains true and nothing changes.

plus a theory θ that satisfies the unity conditions. We score (w, θ) using the function of Definition 27.

Understanding the interpretations

Figure 5.18 shows one interpretation of the fuzzy sequence of Figure 5.14. In this interpretation, the guessed vector length $k_g = 2$ is wrong as the actual vector length is 3. The guessed number of predicates $n_g = 3$ is also wrong as the fuzzy sequence was generated from the symbol sequence *aabbaabbaabb*... that uses 2 symbols.

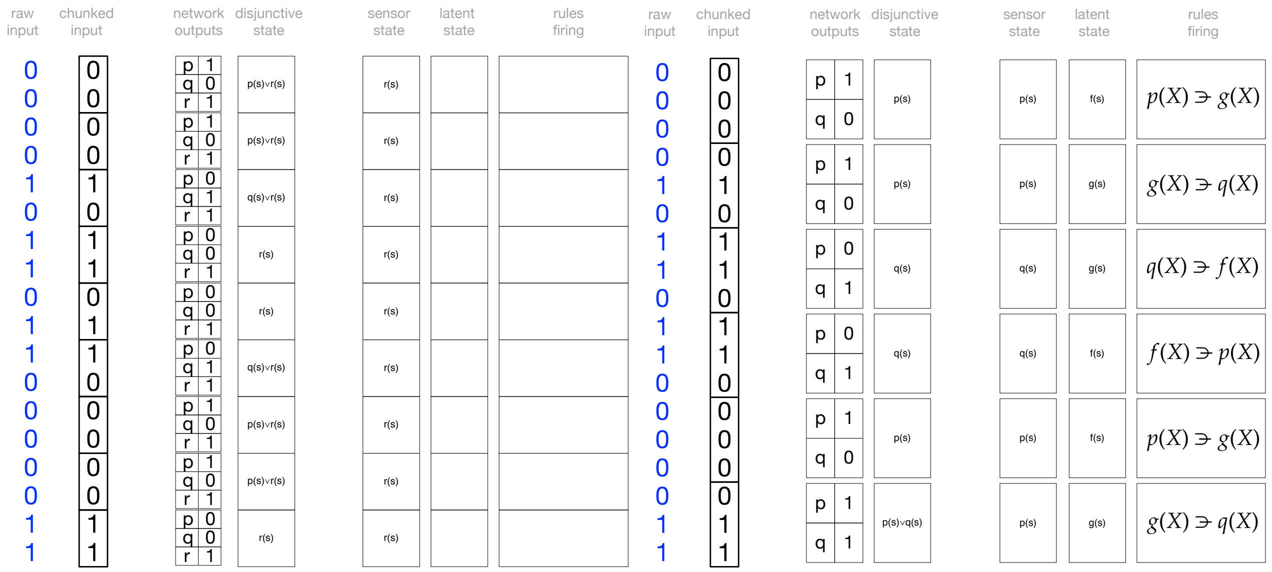
In this interpretation, vectors are mapped to concepts as follows:

$$(0,0) \mapsto p(s) \vee r(s) \quad (0,1) \mapsto r(s) \quad (1,0) \mapsto q(s) \vee r(s) \quad (1,1) \mapsto r(s)$$

Note that *every* vector is mapped to r.

The interpretation found is very simple. The atom $r(s)$ is initially true, and then remains true forever. Nothing changes. Because the guessed vector size is wrong, the system is unable to discern any distinctions in the input, and maps everything to the single concept r. This interpretation is the great leveller, blurring all distinctions. It is inaccurate on the held-out data.

Figure 5.17 shows another interpretation of the same noisy sequence. In this case, the guessed vector size $k_g = 3$ is correct, as is the guessed number of predicates $n_g = 2$. In this interpretation, vectors are



(a) Incorrect guess with $k_g = 2$ and $n_g = 3$

(b) Correct guess with $k_g = 3$ and $n_g = 2$

Figure 5.19: Two interpretations of a sequence generated from *aabbaabbaabb*... with $k = 3$. In both (a) and (b), we show the raw concatenated input, the input divided into chunks of size k_g, the output of the binary neural network, and the disjunction generated by the multiclass classifier. The sensor state column shows how each disjunction is resolved, while the latent state shows the ground atoms that were invented to explain the surface sequence. The final column shows all the rules whose preconditions are satisfied at that moment.

mapped to concepts as follows:

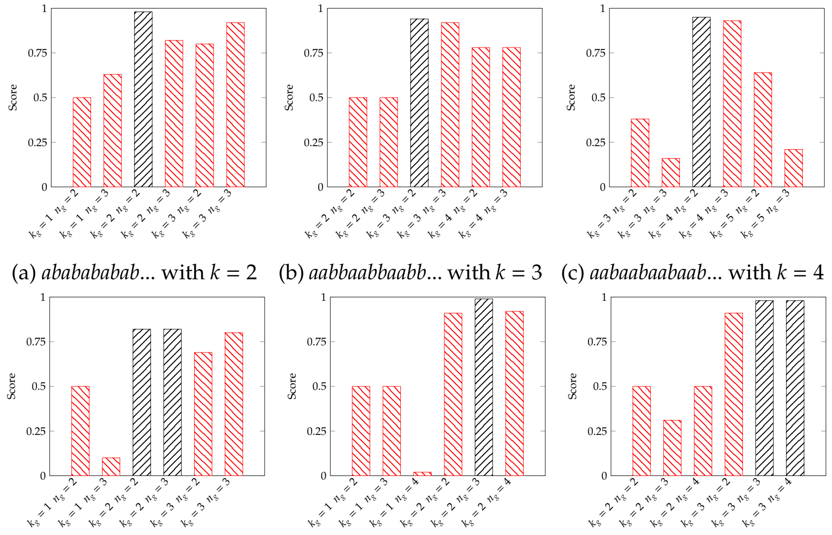
$$
\begin{array}{llll}
(0,0,0) \mapsto p(s) & (0,0,1) \mapsto p(s) & (0,1,0) \mapsto p(s) & (0,1,1) \mapsto p(s) \vee q(s) \\
(1,0,0) \mapsto q(s) & (1,0,1) \mapsto q(s) & (1,1,0) \mapsto q(s) & (1,1,1) \mapsto q(s)
\end{array}
$$

Here, the mapping has one ambiguity on vector $(0,1,1)$. Note that this ambiguity is unavoidable given the original ambiguous mapping in Figure 5.14.

Note that the system has discerned the underlying symbolic sequence *ppqqppqqppqq*..., isomorphic to the original symbolic sequence *aabbaabbaabb*.... of Figure 5.14 that was used to generate the fuzzy sequence. The rules R use f and g as invented predicates to count how many times we are in the two states of p and q.

It is pleasing that the system is able to recover the underlying symbolic sequence as well as the low-level mapping from vectors to concepts, from fuzzy ambiguous sequences. This interpretation is accurate on all the held-out data (see Section 5.5.3).

The two interpretations are compared in Figure 5.19. The left figure (a) shows the interpretation of Figure 5.18 which blurs all distinctions. The right figure (b) shows the interpretation of Figure 5.17, which correctly discerns the underlying symbolic structure.



(a) *abababab*... with $k = 2$ (b) *aabbaabbaabb*... with $k = 3$ (c) *aabaabaabaab*... with $k = 4$

(d) *aaabaaabaaab*... with $k = 2$ (e) *abcabcabcabc*... with $k = 2$ (f) *abcabcabcabc*... with $k = 3$

Figure 5.20: The results of the Apperception Engine on the *Fuzzy Sequences* task. Interpretations that are accurate (on *all* held-out data) are shown in black, while inaccurate interpretations are shown in red. In all our experiments, the highest-scoring interpretations are always accurate.

The probability of the accurate interpretation (see Definition 27) is significantly higher than the probability of the inaccurate interpretation. In general, throughout our experiments, *the most probable interpretations (according to Definition 27) coincide with the accurate interpretations*. This means we are able to retrieve the correct values of k_g and n_g by taking the interpretation with the highest probability. See Section 5.5.3.

The baseline

The baseline we construct for the *Fuzzy Sequences* task is a slightly modified version of the *Sokoban* baseline, an auto-regressive model which fully follows the baseline desiderata of Section 5.5.1.

The model applies a two-layer perceptron to each input element, and passes the result to the LSTM tasked with predicting the distribution (Gumbel-Softmax [JGP16, MMT16]) of the next element. Following the Gumbel-Softmax is a two-layer perceptron which decodes the samples from the distribution into the following element of the sequence.

We trained the baseline with the Adam optimizer, set the learning rate to 0.01 and trained on only the single example. After noticing that the model struggles with producing a crisp distribution, we introduced the KL-weighing beta parameter [HMP+17] and set it to $\beta = 0.1$ to produce better representations. We ran the model on each instance of the task 10 times on different random seeds.

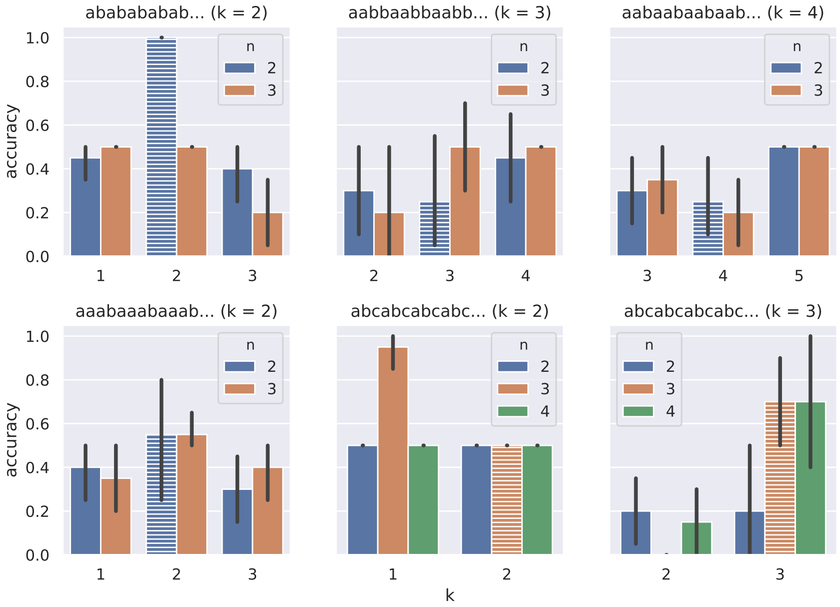


Figure 5.21: The results of the neural baseline on the *Fuzzy Sequences* task. Striped bar denotes the correct (n, k) choice.

Results

Figure 5.20 shows the results for the Apperception Engine, while Figure 5.21 shows the results for the neural baseline.

Figure 5.20 shows, for six fuzzy sequences, an evaluation of different theories with different guesses for k_g and n_g. The accurate theories (those that correctly predict *all* held-out data) are shown in black, while inaccurate theories are shown in red. Notice that the score (based on the log probability of the (w, θ) pair from Definition 27) is a reliable indicator of the accuracy of the interpretation. This means that we can run a grid search over guesses for k_g and n_g, choose the interpretation with the highest score, and confidently expect that this interpretation will be accurate on the held-out data. The central point here is that we do not need to provide the system with information about the way the fuzzy sequence is grouped into chunks. Rather, *the system itself can induce the correct way to group the data as part of the apperception process.*

Since the induced baseline representations often were not sharply discrete, we did not compare it to the Apperception Engine on the same scoring but we evaluated it only on its capacity to correctly predict elements of the sequence. In Figure 5.21 we observe that the baseline correctly learns to predict only the *abababab...* sequence and the *abcabcabcabc...* (k = 2) sequence. The first sequence is correctly predicted, whereas the second one, though simple to learn for the model, does not present the correct choice of parameters, showing that the model, even though able to predict some sequences, cannot provide a reliable accuracy for choosing correct parameter guesses. Further looking into the Gumbel-Softmax parameters shows that, when the model learns the sequence well, it does induce a meaningful crisp distribution, but when it does not it learns a distribution which is not useful for interpretation. We also notice that for the sequences it cannot learn, the neural model exhibits severe overfitting on the single example, an expected phenomenon when trained on a low number of examples.

Chapter 6

Kant's cognitive architecture

In this chapter, we describe the particular interpretation of Kant that underlines the computer models described above. It might seem, at first, somewhat odd to present the philosophical motivation only after the computer system has already been described. But the example below (that is needed to concretise our particular interpretation of Kant) requires the formal machinery that has been developed in earlier chapters. Readers who are not interested in the interpretation of Kant should feel free to skip this chapter.

The material in this chapter is based, in part, on the following publications:

> "Kant on Constituted Mental Activity", *The American Philosophical Association*, Volume 16, 2017.
>
> "A Kantian Cognitive Architecture", *Philosophical Studies*, 2018.
>
> "Formalizing Kant's Rules", *Journal of Philosophical Logic*, 2019.

6.1 Introduction

We are familiar with the idea that social activity is *constituted* activity. The utterance of the words "I do" counts, in the right circumstances, as an acceptance of marriage vows. Pushing the wooden horse-shaped piece forward counts, in the right circumstances, as moving the knight to king's bishop three. Jones' running away counts, in the right circumstances, as desertion. These social actions are things we can only do *indirectly*, by doing something else. A social action is not something we can *just do*.

Kant's cardinal innovation, as I read him, is to see *mental* activity as constituted activity. This plurality of sensory perturbations counts, in the right circumstances, as representing a red triangle. This activity of rule application counts, under the right circumstances, as seeing an apple. This activity of rule construction counts, under the right circumstances, as forming the belief that Caius is mortal. Kant's surprising claim is that *mental activity is itself constituted*. We have to perform a certain type of activity in order to experience a world at all.

6.1.1 From counts-as to counting-as

Let us start by considering the *activity* of counting-as:

- Jones counts Smith's contortion of the lips as a delighted smile
- The sergeant counts Jones' running away as desertion
- The teacher counts the boy's squiggle as an "s"
- The vicar counts the utterance of the words "I do" as an acceptance of the marriage vows

Notice that these examples describe the *activity* of counting-as, rather than the mere *relation* of counts-as. The counts-as relation is commonly formulated as:

> x counts as y (in context c)

This sentence, ascribing a three-place relation between x, y and c, ignores the *person* who is doing the counting-as, and the *business of counting itself*, and focuses solely on the resulting judgement. If we want to acknowledge the individual performing the counting, and the activity of counting as, we would write it as:

> agent *a* counts x as y (in context c)

This sentence describes the activity of counting-as, and makes explicit the person who is *doing* the counting.

Under what circumstances would it be ok to forget about the person doing the counting? Perhaps it would be ok to suppress the agent and the activity of counting-as in cases where everyone agreed about what counted-as what, where mass agreement in counting-as was taken for granted. Throughout the *Investigations* [Wit09], Wittgenstein repeatedly asks us to stop taking this mass communal agreement for granted. He demands "what if one person reacts in one way and another in another?" (*Investigations*, §206). For example, he considers the case where:

> a person naturally reacted to the gesture of pointing with the hand by looking in the direction of the line from finger-tip to wrist, not from wrist to finger-tip (*Investigations*, §185)

The divergence here is a difference in what activity the deviant person is counting the gesture *as*. The deviant is counting the gesture as pointing in the opposite direction from what "we"[1] count the gesture as.

Whenever Wittgenstein talks about counts-as, he is careful to talk about the *activity* of counting-as, rather than an abstract *relation* of counts-as that presupposes communal agreement:

> But now imagine a game of chess translated according to certain rules into a series of actions which we do not ordinarily associate with a game - say into yells and stamping of feet. ... *Should we still be inclined to count them as* playing a game? What right would one have to say so? (*Investigations*, §200) (my emphasis)

Wittgenstein focuses on edge cases like these, cases where we are no longer sure that everybody agrees about what counts as what, in order to help us stop treating this mass agreement as *given*. In a shared culture, there is indeed mass agreement in what counts as what. But this mass agreement is an *achievement*, something painfully accomplished by constant communication and teaching, a fragile accomplishment that is always in need of renewal. For Wittgenstein, as Cavell reads him [Cav99], mass agreement in counting-as activity is not something that should be presupposed at the beginning of philosophical activity, but is instead rather *something to be explained*.

> I find my general intuition of Wittgenstein's view of language to be the reverse of the idea many philosophers seem compelled to argue against in him: it is felt that Wittgenstein's view makes language too public, that it cannot do justice to the control I have over what I say, to the innerness of my meaning. But my wonder, in the face of what I have recently been saying, is rather how he can arrive at the completed and unshakable edifice of shared language from within such apparently fragile and intimate moments - private moments - as our separate counts and out-calls of phenomena, which are after all hardly more than our interpretations of what occurs, and with no assurance of conventions to back them up. *The Claim of Reason, p.36*

Instead of an abstract "x-counts-as-y" relation that suppresses the agent performing the counting-as activity and that presupposes communal agreement, Wittgenstein wishes us to start with an *individual* agent counting something as something. It is this same counting-as activity that is needed, I claim, to understand Kant's project in the First Critique.

6.1.2 From derivative to original intentionality

Consider the humble barometer, a simple sensory device that can detect changes in atmospheric pressure. If the mercury rises, this means the atmospheric pressure is increasing; if the mercury goes

[1] For Wittgenstein, the community of "we" *just is* the set of individuals who count-as in the same way.

down, the pressure is decreasing. Now *we count* the mercury's rising *as* the machine responding to the atmospheric pressure. We count, in other words, a process that is internal to the instrument (the mercury rising) as representing changing properties of an external world (atmospheric pressure increasing). But although *we* count the internal process as representing an external process, the *barometer itself* does not. The barometer is incapable of counting the internal process as a representing because - of course - it is incapable of counting anything as anything.

The barometer does not, in other words, have original intentionality. *We* might interpret some of its activities as representations, but *it* does not.

The distinction between **original** and **derivative intentionality** comes from Haugeland [Hau90]. Intentionality is derivative if it is attributed by someone else, by another agent who is doing the counting-as:

> At least some outward symbols (for instance, a secret signal that you and I explicitly agree on) have their intentionality only derivatively - that is, by by inheriting it from something else that has the same content already (e.g. the stipulation in our agreement). And, indeed, the latter might also have its content only derivatively, from something else again; but obviously, that can't go on forever. Derivative intentionality, like an image in a photocopy, must derive eventually from something that is not similarly derivative; that is, at least some intentionality must be original (non derivative). (Intentionality All Stars p.385)

We can reformulate the derivative/original intentionality distinction in terms of the counting-as activity:

- x has derivative intentionality in representing p if an agent y (distinct from x) counts x's activity as x's representing p
- x has original intentionality in representing p if x himself counts x's activity as x's representing p

What distinguishes an agent with original intentionality from a mere sensory instrument is that the former *counts its own sensings as* representations of a determinate external world:

> There is no doubt whatever that all our cognition begins with experience; for how else should the cognitive faculty be awakened into exercise if not through objects that stimulate our senses and in part themselves produce representations, in part bring the activity of our understanding into motion to compare these, to connect or separate them, and thus to *work up the raw material of sensible impressions into a cognition of objects that is called experience*? [B1] (my emphasis)

Original intentionality, in other words, is a type of activity interpretation. Just as I can count his moving the horse-shaped wooden piece from one square to another as his moving his knight to king's bishop three, just so I can count the perturbations of my sensory instruments as my representing a determinate world[2].

6.1.3 From sensory agents to cognitive agents

A **sensory agent** is some sort of animal or device, equipped with sensors. It might have a temperature gauge, a camera with limited resolution, or a sonar that can detect distance. The sensory agent is continually performing what roboticists call the sense-act cycle: it detects changes to its sensors, and responds by bodily movements.

A thermostat, for example, is a simple sensory agent. When it notices that the temperature has got too low, it responds by increasing the temperature. The thermostat has a sense-act cycle, but it does not experience the world it is responding to. *We* count the perturbations of its gauge as representations of the temperature in the room it is in, but *it* does not. The gauge movements count as temperature representations *for us*, but not *for the thermostat*. Nothing counts as anything for the thermostat. It just responds *blindly*.

By contrast, a **cognitive agent** is a sensory agent with original intentionality, who *counts* his sensings *as* his representing an external world. He interprets his own sensory perturbations as his representation of a coherent unified world of external objects, interacting with each other. This world contains one particular distinguished object, with sensors, that the cognitive agent counts as his body, and he *interprets* his sensings *as* the stimulation of his body's sensors by interaction with the other objects.

Kant's fundamental question is:

> *What does a sensory agent have to do, in order for it to count its own sensory perturbations as experience, as a representation of an external world?*

What, in other words, must a sensory agent do to be a cognitive agent?

Note that this is a question about intentionality - not about knowledge. Kant's question is very different from the standard epistemological question:

> Given a set of beliefs, what else has to be true of him for us to count his beliefs as knowledge?

[2] (Aside). In other words, we can only represent a world because we can count some activity as mental activity. Therefore, the ability to count activity as intentional (representational) behaviour is necessary to be able to think a world at all. This has interesting consequences for scepticism about others' minds. The sceptic suggests it is possible for us to be able to make sense of a purely physical world of physical activity, and asks with what right we assume that some of this activity is mental activity. But if the above is right, the capacity to count activity as mental activity is *necessary to think anything at all* - there is no intentionality-free representation of the world, in terms of bare particulars. There is always already the ability to see activity as intentional activity before we can see anything.

Kant's question is *pre-epistemological*: he does not assume the agent is "given" a set of beliefs. Instead, we see his beliefs as an *achievement* that cannot be taken for granted, but has to be *explained*:

> Understanding belongs to all experience and its possibility, and the first thing that it does for this is not to make the representation of the objects distinct, *but rather to make the representation of an object possible at all* [A199, B244-5]

Kant asks for the conditions that must be satisfied for the agent to have any possible cognition (true *or* false) [A158, B197].

6.1.4 Kant's fundamental question

Kant's fundamental question, then, is:

> What activities must be performed if the agent is to achieve **experience**?[3]

Now this is not an empirical psychological question about the processes that *homo sapiens* happen to use, but rather a question of *a priori* psychology[4]: what must a system – any physically realised system at all[5] – do in order to achieve experience?[6]

In this chapter, I will try to distill Kant's answer to this fundamental question, and reinterpret his answer as the specification of a cognitive architecture.

6.2 Experience and synthetic unity

A central claim of the *Transcendental Deduction* is that:

> (1) In order to achieve experience, I must unify my intuitions. [A110]

Before we can assess the truth of such a claim, we first need to understand what it means. (i) What does Kant mean by an experience? (ii) What are intuitions? (iii) What does it mean to unify them? I shall consider each in turn.

[3] The subtitle of the *Transcendental Deduction* in the First Edition is: "On the *a priori* grounds for the possibility of experience." [A95]

[4] In this thesis, I side with Longuenesse[Lon98], Waxman [Wax14], and others in interpreting the first half of the *Critique* as *a priori* psychology. *Contra* Strawson [Str18], I believe that *a priori* psychology is a legitimate and important form of inquiry, and that if we try to expunge it from Kant's text, there is not much left that is intelligible.

[5] There are a number of places in the *Critique* where Kant seems to restrict his inquiry to just humans e.g., [B138-9]. But Kant uses the term "human" to refer to any agent who perceives the world in terms of space and time and has two distinct faculties of sensibility and understanding. This is a much broader characterisation than just *homo sapiens*.

[6] Because the second question is broader, it is more relevant to the project of artificial intelligence [Den78].

6.2.1 What does Kant mean by 'experience'?

Kant's notion of **experience** ('Erfahrung') is close to our usual use of the term. I shall list some features of this term as Kant uses it.

- Experience is *everyday*. It is not an unusual peak state that people only achieve occasionally, like enlightenment or ecstasy. Rather, it is a state that most of us have most of the time when we are awake.
- Experience is *unified*. At any one time, I am having *one* experience [A110]. I cannot have multiple simultaneous experiences. I may be conscious of multiple stimuli, but they are all part of one experience.
- Experience is *mine* [B134]. It belongs to me. My experience is different from your experience.
- Experience is *articulated* [Ste13]. It is not a mere 'blooming, buzzing confusion' [JBBS90]. Rather, experience is composed of distinct objects with distinct properties.
- Experience is *not (merely) conceptual*. It is not just a collection of beliefs. It is, to anticipate, a unified combination of sensible and discursive cognition.
- Experience is *not necessarily veridical*. It purports to represent the world accurately, but may fail to do so [Lon98, Ste13, Wax14].

Experience, then, is an everyday not-necessarily-veridical mental state in which I am conscious of various distinct objects and their attributes.

Experience is not something we should take for granted. Rather, experience is *an achievement*. When I open my eyes, I see various objects, with various properties that change over time. But this experience is a complex achievement that only occurs if a myriad of underlying processes work exactly as they should do. The central contribution of Kant's *a priori* psychology is to describe in detail the underlying processes needed in order for experience to be achieved.

6.2.2 What does Kant mean by 'intuition'?

An **intuition** ('Anschauung') is a representation of a particular object[7] (e.g., this particular jumper) or a representation of a particular attribute[8] of a particular object at a particular time (e.g., the particular dirtiness of this particular jumper at this particular time).

[7] [B76]

[8] A186/B229: "The determinations of a substance that are nothing other than *particular ways for it to exist* are called accidents." Note that whenever Kant talks about "existence" in the Analogies, he is really talking about a particular *way of existing*. See e.g., A160/B199: "synthesis is either mathematical or dynamical: for it pertains partly merely to the intuition, partly to the existence of an appearance in general". Here, "the existence of an appearance" means the particular way of existing of an appearance (e.g., the particular dirtiness of this particular jumper).

Intuitions are produced by the faculty of **sensibility** [A19/B33]: the receptive faculty that detects sensory input. Sensibility provides the agent with a *plurality* of intuitions [B68], which the mind needs to make sense of.

Intuitions are private to the individual. My intuitions are different from yours. It is not just that we do not share intuitions – we *cannot* share intuitions, as they are essentially private. To see this, consider four possible relations between an action and its object:

1. the object existed before and after the action (e.g., kicking the football)
2. the object existed before but not after the action (e.g., destroying the evidence)
3. the object existed after but not before the action (e.g., making a cake)
4. the object existed neither before nor after, but was only an *aspect* of the action

Let us focus on the fourth. When I draw a circle in the air, this thing – the circle – only exists for the duration of the activity because it is an *aspect* of the activity. Or consider "the contempt in his voice": this thing, this contempt, only exists for the duration of his speech-act because it is an aspect of the speech-act.

The way I read Kant, the object of intuition is a type (4) object: it only exists as part of the act because it is an aspect of the act.[9]

> But in order to cognize something in space, e.g., a line, I must **draw** it. [B137]

Now because intuiting is a private mental act (no other agent can perform the same token-identical act), and because the object of intuition is a type (4) object that only exists as an aspect of the act, it follows that the object of intuition inherits the privacy of the intuiting act of which it is an aspect. Nobody else can have my particular object of intuition because this object is an aspect of my activity of intuition, and nobody else can perform this particular activity.

Intuitions are distinct from concepts. While an intuition is a representation of a particular object, a concept is a general representation that many intuitions fall under [B377]. For Kant, intuitions

[9] Kant interpreters differ on whether intuitions are relations between conscious minds and actual existing material objects [All09, Gom13, McL16], or whether the object of an intuition is just a mental representation that in no way implies the existence of a corresponding external physical object [Lon98, Ste15, Ste17]. The interpretation in this thesis fits squarely within the latter, representational interpretation. My reason for preferring the representational interpretation is based on a general interpretive prejudice: whenever there are two ways of reading Kant, and one of those interpretations relies on fewer prior capacities, thus requiring the mind to do more work to achieve the coherent representation of an external world that we take for granted in our everyday life, then prefer that interpretation. The relational view takes for granted a certain type of cognitive achievement: the ability of the mind to be about an external object. The representational view, by contrast, sees this intentionality, this mind-directedness, as something that requires work to be achieved. Thus, simply because it is more demanding and asks harder questions, it should be preferred. Further, and not coincidentally, the representational view can be implemented in a computer program, while it is entirely unclear how we could begin to implement any relational view that takes for granted the ability for the mind's thoughts to be directed to particular external physical objects.

and concepts are distinct types of representation. While empiricists saw concepts as a special type of intuition that is used in a general way, and while rationalists saw intuitions as a special type of concept that is maximally specific, Kant understood intuitions and concepts to be entirely distinct *sui-generis* types of representation. His reasons for thinking intuitions and concepts are entirely distinct are: (i) they come from distinct faculties (sensibility and understanding respectively); (ii) while intuitions are private to an individual, concepts can be shared between individuals; (iii) while intuitions are immediately directed to an object (the particular object only exists as an aspect of the activity of intuiting, just as the circle only exists as an aspect of the activity of drawing a circle in the air), concepts are only mediately related to objects via intuitions [A68/B93, B377].

The intuition occupies a unique place in Kant's *a priori* psychology: it is the ultimate goal of all thought, the final end that all cognition is aiming at. All the other aspects of thought (e.g. concepts and judgements) are only needed in so far as they help to unify the intuitions:

> In whatever way and through whatever means a cognition may relate to objects, that through which it relates immediately to them, and *at which all thought as a means is directed as an end, is intuition*. [A19/B33, my emphasis.]

6.2.3 What does Kant mean by 'unifying' intuition?

Recall Kant's key claim that:

> (1) In order to achieve experience, I must unify my intuitions.

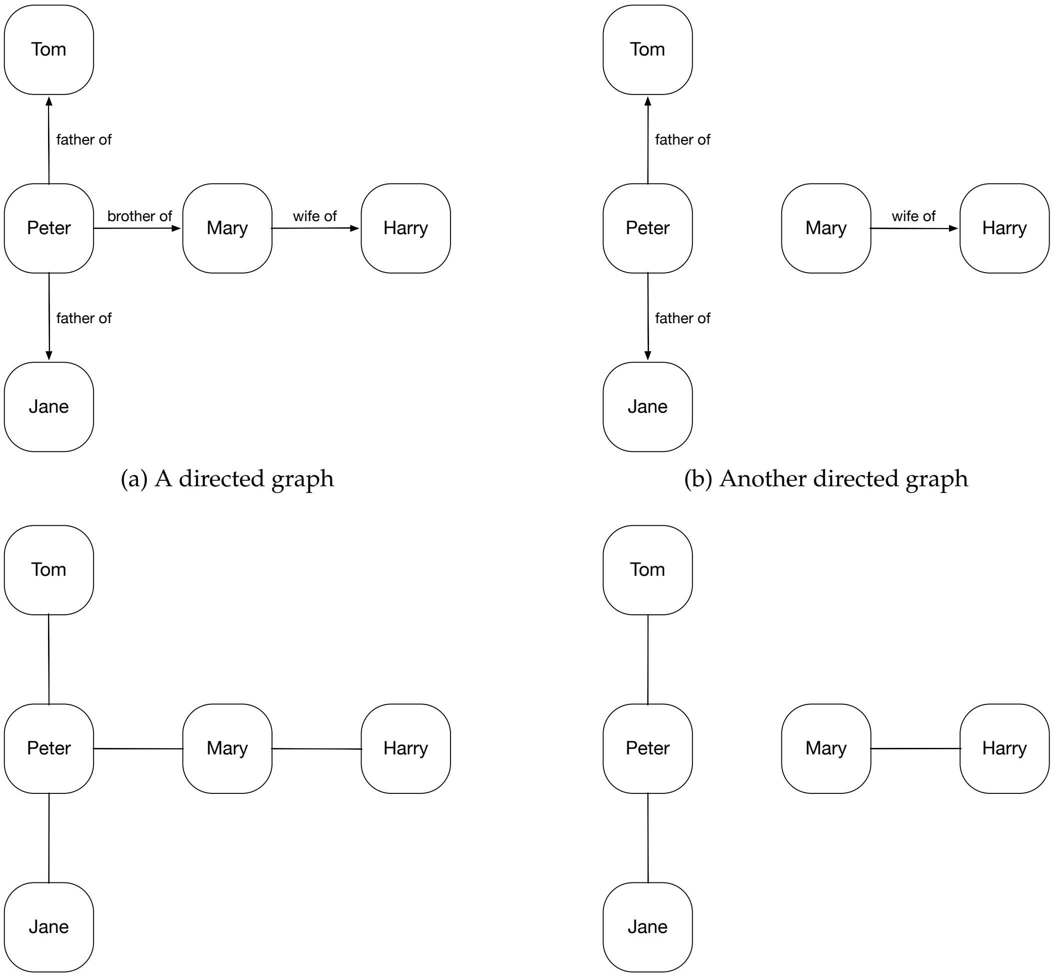
Here, the *explanandum* is a mental state (experience), while the *explanans* is a process (the process of unifying the intuitions). But what, exactly, does this process involve, and how will we know when it is finished?

The process of unifying intuitions can be unpacked as a particular type of synthesising process that satisfies a particular constraint, the constraint of unity:

> But in addition to the concept of the manifold and of its synthesis, the concept of combination also carries with it the concept of the *unity of the manifold*. [B130]

I shall first consider the synthesising process in general, and then turn to the unity constraint. The activity of synthesis may seem frustratingly metaphorical or ill-defined:

> The inadequacies of such locutions as "holding together" and "connecting" are obvious, and need little comment. Perceptions do not move past the mind like parts on a conveyor belt, waiting to be picked off and fitted into a finished product. There is no workshop where a busy ego can put together the bits and snatches of sensory experience, hooking a color to a hardness, and balancing the two atop a shape. [Wol63, p. 126]



(a) A directed graph

(b) Another directed graph

(c) Undirected version of (a).

(d) Undirected version of (b)

Figure 6.1: Binary relations as directed graphs. Only (c) is fully connected.

What exactly does it mean to unify intuitions? What is the glue that binds the intuitions together? As I read Kant, the only thing that can bind intuitions together is the *binary relation*[10]. Consider Figure 6.1. Here, in Figure 6.1(a), we have various objects related by various directed binary relations. The diagram below it, Figure 6.1(c), is the undirected variant of (a). Note that Figure 6.1(c) is **connected**: we can get from every node to every other node via some path. Figure 6.1(b) shows another set of directed binary relations. The diagram below it, Figure 6.1(d), is the undirected variant of (b). Figure 6.1(d) is not connected, since we cannot reach *Mary* from *Peter*, for example.

Synthesising intuitions means connecting the intuitions together using binary relations so that the resulting undirected graph is fully connected. The synthesising process is the job of the faculty of **productive imagination**[11] [A78/B103; A188/B230], described in Section 6.3 and formalized in Section 6.9.

But there is much – much more – to unifying intuitions than just connecting them together with binary relations. The extra requirement that must be satisfied for a connected binary graph to count as a unification of intuitions is that the graph satisfies Kant's unity conditions. While there are many ways to connect intuitions together via binary relations to form a connected graph, only a small subset of these satisfy the various conditions of unity that Kant imposes. These unity conditions are satisfied by the faculty of **understanding** [A79/B104], and are described in detail in Sections 6.5, 6.6, 6.7, and 6.8.

The second claim, then, unpacks what it means to unify intuitions:

(2) Unifying intuitions means combining them using binary relations to form a connected graph, in such a way as to satisfy various unity conditions (described in detail in Sections 6.5, 6.6, and 6.8).

6.2.4 The status of claim 1

Claim (1), then, is the claim that an agent can only achieve experience – everyday conscious experience of a single articulated world – if it can unify its intuitions by connecting them together in a relational graph that satisfies various (as yet unspecified) unity conditions.

Let us break this down into two claims:

(1a) In order to achieve experience, my intuitions must be unified.

(1b) In order for my intuitions to be unified, I must unify them.

Claim (1a) can be interpreted with at least two levels of strength. A strong interpretation treats the claim as *definitional*: experience *just is* unified intuition. A weaker interpretation sees the claim as

[10] The precise binary relations involved are listed in the *Schematism* and described in detail in Section 6.3.

[11] Kant distinguished between the productive and reproductive imagination [A100-2]. Here, we focus exclusively on the productive imagination. The reproductive imagination's job is to recall earlier determinations and reproduce them. This capacity is taken for granted in the current implementation: we assume the whole sequence of sensory input has been given as a whole, so the agent does not need to recall earlier elements.

merely a necessary condition: experience *requires* unified intuition, but it also needs more besides. In this thesis, we adopt the stronger interpretation, and there is reason to think that Kant endorsed this stronger interpretation too.[12]

The second claim (1b) is not entirely trivial. An alternative possibility is that my intuitions arrive, via the faculty of sensibility, *already unified*. But Kant clearly rules out this alternative[13]. So, then, if my intuitions do not arrive already unified, and if I cannot pay or persuade somebody else to unify them for me[14], then I must unify them myself. This is a task that only I can do.

Kant also uses various alternative formulations of Claim 1. For example:

> (1*) In order for the intuitions to be *mine*, I must unify them. [B134]

This follows from Claim 1 if experience just is (definitional equality) the intuitions that are mine.

He also uses another formulation:

> (1**) In order for me to be *conscious of the intuitions*, I must unify them. [B135]

This follows from Claim 1 if experience just is (definitional equality) the consciousness of my intuitions.

6.3 Synthesis

In this section, I describe the relations that are used by the imagination to connect the intuitions together [A78/B103].

In Figure 6.1(a), the intuitions were connected by empirical relations (e.g., father-of). These family-tree relations may relate some types of objects in some situations, but they do not relate all objects in all situations.

When Kant talks about pure synthesis [A78/B104], he means connecting intuitions by **pure** relations that apply to all intuitions in all situations[15]. Why does Kant insist that synthesis can only use pure relations to connect intuitions? Because the unity conditions (that will be described in Sections 6.5, 6.6, and 6.8) are conditions that must apply to *every possible synthesis* of intuitions. If the unity conditions are to apply to every possible synthesis, they can only reference relations that feature in every possible synthesis, and these are the pure relations.

There are three[16] operations that bind intuitions together:

[12] "[Experience] is therefore a synthesis of perceptions." [A176/B218] "There is only one experience, in which all perceptions are represented as in thoroughgoing and lawlike connection." [A110]

[13] "Yet the **combination** (*conjunctio*) of a manifold in general can never come to us through the senses, and therefore cannot already be contained in the pure form of sensible intuition." [B129]

[14] Nobody else can get anywhere near my intuitions because they are aspects of my private mental acts. See Section 6.2.2.

[15] Kant enumerates the pure relations in the *Schematism*.

[16] The containment operation is described in the *Axioms of Intuition*, the comparison operation in the *Anticipations of Perception*, and the inherence operation in the *First Analogy*.

- **containment**: $in(X, Y)$ means that object X is (currently) in object Y (e.g., the package is in the kitchen)

- **comparison**: $X < Y$ means that attribute X is (currently) less than attribute Y (e.g., the weight of the package is less than the weight of the spoon)

- **inherence**: $det(X, Y)$ means that attribute Y (currently) inheres in object X (e.g., this particular heaviness (of 2.3 kg) is an attribute of this particular parcel)

When two intuitions are bound together by one of the three operations, the result is a **determination**. Thus, $det(a, b)$, $in(a, b)$, and $a < b$ are all determinations. Determinations hold at a particular moment or moments in time; they do not persist indefinitely [A183-4,B227].

The constituents of determinations are intuitions, representations of individuals; these are either particular objects, or particular attributes of those objects. To hold $det(a, b)$ is to ascribe particular attribute b to particular object a (for example, to ascribe this particular dirtiness to this particular jumper).

It is absolutely essential, I believe, for understanding Kant's architecture that we distinguish clearly between attributes and concepts. Attributes are a type of intuition representing the particular way in which a particular object exists at a particular moment. Concepts, by contrast, are general representations. A number of different attributes typically fall under the same concept. Consider, for example, the particular dirtiness of this particular jumper, and the particular dirtiness of this particular laptop. Both attributes fall under the concept "dirty", but they are nevertheless distinct attributes: this jumper's particular dirtiness is different in myriad subtle ways from the dirtiness of my laptop.

Just as an attribute is a different kind of representation from a concept, just so a determination is a different kind of thought from a judgement. Seeing the particular dirtiness of the particular jumper at this particular moment (a determination) is very different from believing that the particular jumper is dirty (a judgement). In the former, I notice an individual property of an individual object. In the latter, I subsume a concept representing an individual object (the particular jumper) under a general concept ("dirty").

A determination is not a judgement, but a *way of perceiving*:

- I hear the baby in the cot (containment)

- I feel the package being heavier than the spoon (comparison)

- I see the dirtiness of the jumper (inherence)

In each case, the argument of the perceptual verb is a *noun-phrase*, not a that-clause [Sel78].

Since a determination is a way of perceiving, it does not have a truth-value:

> For truth and illusion are not in the object insofar as it is intuited, but in the judgment about it insofar as it is thought. Thus it is correctly said that the senses do not err; yet not because they always judge correctly, but because they do not judge at all. Hence truth, as much as error, and thus also illusion as leading to the latter, are to be found only in judgments, i.e., only in the relation of the object to our understanding... In the senses there is no judgment at all, neither a true nor a false one. [A293-4/B350] See also [*Jäsche Logic* 9:53].

As well as the three pure operations that bind intuitions together, there are three[17] pure relations that bind determinations together:

- **succession**: $succ(P_1, P_2)$ means that P_1 is succeeded (at the next time-step) by P_2
- **simultaneity**: $sim(P_1, P_2)$ means that P_1 occurs at the same moment as P_2
- **incompatibility**: $inc(P_1, P_2)$ means that P_1 and P_2 are incompatible

When two determinations are bound together by one of the three relations, the result is a **connection**[18]. Thus, $succ(in(a, b), in(a, c))$ means that a's being in b is succeeded by a's being in c, and $inc(det(a, b), det(a, c))$ means that attributing b to a is incompatible with attributing c to a.

6.3.1 The justification for this particular set of operations and relations

Why these particular pure relations? What makes this particular list special? The justification for this list is that the three pure operations and the three pure relations together constitute a *minimal set of binary operators that together are sufficient to construct the forms of space and time* [A145/B184ff].[19]

According to Kant, intuitions and determinations do not arrive with space and time coordinates attached [B129]. The job of sensibility is just to provide us with intuitions, but not to arrange them in objective space/time. It is the function of *synthesis*, the job of the imagination, to connect the intuitions together, using the pure operations and relations described above, so as to construct the objective spatio-temporal form:

> since time itself cannot be perceived, the determination of the existence of objects in time can only come about through their combination in time in general, hence only through *a priori* connecting concepts. [A176/B219]

[17] The succession and simultaneity relations are described in the second and third *Analogies*, and incompatibility is discussed in the *Postulates of Empirical Thought*.

[18] "Experience is possible only through the representation of a necessary connection of perceptions." [B218]

[19] This claim holds for a suitably qualified minimal notion of space. See Section 6.5.

To see that sensibility does not provide us with intuitions that are already positioned in space and time, consider a robot with a camera that provides a two-dimensional array of pixels for each visual snapshot. The robot receives information about the location of each pixel in subjective two-dimensional space, and it must determine the positions of objects in *three*-dimensional space. Suppose a yellow pixel is left of a red pixel. Does the yellow pixel represent an object that is in front of the object represented by the red pixel, or behind? The visual input does not provide this information – the robot must decide itself. Next, consider time. Suppose the robot receives a sequence of visual impressions as its camera surveys the various parts of a large house [B162]. Do these subjectively successive impressions count as various representations of one moment in objective time, or do they represent different moments of objective time? The sensory input arrives ordered in subjective space/time but not in objective space/time.[20] In order to place our intuitions in objective space/time, the imagination needs to connect them together using the pure relations described above.[21]

The three pure operations together with the three pure relations constitute a minimal set that is sufficient for generating the form of objective space/time. The containment operation *in* allows us to combine intuitions into a spatial field (a minimal representation of space that abstracts from the number of dimensions [Wax14]) [A162/B203ff]. The comparison operation < allows us to compare two different attributes; if we generate an intermediate attribute between two comparable attributes, we can generate an intermediate moment in time between two observed moments [A165/B208ff], thus filling time [A145/B184]. The inherence operation *det* allows us to ascribe different attributions to an object at different times. The simultaneity and succession relations allow us to order determinations in time. Finally, the incompatibility relation allows us to test when sets of determinations are compossible.

> Now one sees from all this that the schema of each category contains and makes representable: in the case of magnitude, the generation (synthesis) of time itself, in the successive apprehension of an object; in the case of the schema of quality, the synthesis of sensation (perception) with the representation of time, or the filling of time; in the case of the schema of relation, the relation of the perceptions among themselves to all time (i.e., in accordance with a rule of time-determination); finally, in the schema of modality and its categories, time itself, as the correlate of the determination of whether and how an object belongs to time. The schemata are therefore nothing but *a priori* **time-determinations** in accordance with rules, and these concern, according to the order of the categories, the **time-series**, the **content of time**, the **order of time**, and finally the **sum total of time** in regard to all possible objects. [A145/B184ff]

The third key claim, then, is:

[20] Kant makes this claim many times in the *Principles*. See [A181/B225], [A183/B226], etc.

[21] In [Wax14] Chapter 3, Wayne Waxman makes a powerful case that intuitions do not arrive from sensibility already unified. They arrive as a mere multitude, and it is the job of the imagination to unify them in space/time. In other words, what the empiricist takes as "given" (the unified field of sensory input) is not actually "given" but rather has to be *achieved* by a mental process.

(3) Synthesis involves (i) connecting intuitions together via containment, comparison, and inherence operations to form determinations; and (ii) connecting determinations together via succession, simultaneity, and incompatibility relations.

6.4 The unity conditions

There are many ways to connect intuitions together via binary relations to form a connected graph[22], but only a small fraction of these satisfy the various unity conditions that Kant imposes.

(4) There are, in total, four types of unity condition that Kant imposes: (i) the unity conditions for the synthesis of mathematical relations, (ii) the unity conditions for the synthesis of dynamical relations, (iii) the requirement that the judgements are underwritten by determinations, and (iv) the conceptual unity condition.

I shall go through each in turn.

6.5 The unity conditions for the synthesis of mathematical relations

Kant divides the pure relations into two groups: the **mathematical relations** (containment and comparison) and the **dynamical relations** (inherence, succession, simultaneity, and incompatibility). The mathematical relations control the arbitrary synthesis of homogeneous elements[23], while the dynamical relations control the necessary synthesis of heterogeneous elements[24] [B201n].

Kant says that the mathematical relations combine "what does not necessarily belong to each other" while the dynamical relations combine what "necessarily belongs to one another" [B201n]. This means that the agent has freedom to synthesise using containment and comparison in a way that is unconstrained by the conceptual realm of the understanding, but the synthesis using the dynamical categories is constrained by judgements produced by the understanding.[25]

I shall start with the unity conditions for the mathematical relations, before moving to the unity conditions on the dynamical relations. The fundamental unity condition for the mathematical relations is that the intuitions are combined in a fully connected graph. There are two further specific conditions, one for containment and one for comparison.

[22] If there are n nodes, then there are $2^{\binom{n}{2}}$ simple undirected graphs. The number of simple connected graphs for n nodes is the integer sequence $A001187$ which starts $1, 1, 1, 4, 38, 728, 26704, 1866256, ...$ See http://oeis.org/A001187

[23] Observe that *in* relates two objects of intuition, while $<$ relates two intuition attributes.

[24] Observe that *det* relates two different types of intuition, an attribute and an object.

[25] See also [B110]: "the first class (mathematical categories) has no correlates which are to be met with only in the second class". Here, the correlates are the judgements that are required to underwrite the dynamical connections, but that are not required to underwrite the mathematical compositions.

The unity condition for containment requires that there is some object, the maximal container, which contains all objects at all times [A25/B39]. Slightly more formally, the first unity condition for the synthesis of mathematical relations is:

(5)(a) There exists some intuition x such that for each object of intuition y, for each moment in time, there is a chain of *in* determinations between y and x.

Of course, objects can move about, from one container to another, but at every moment, the objects must always be contained in the maximal container.

Satisfying this unity condition means positing both pure objects (spatial regions with a mereological structure) and also impure objects (appearances) which are in the spatial regions.

Once objects have been placed in the containment hierarchy, and once we know which intuitions fall under which concepts, then we have all the information we need for *counting*. In order to count how many pens are in the box, I need to be able to tell whether each object falls under the concept "pen", and I also need to be able to tell which objects are actually in the box and which are outside. Thus, as Kant says, the pure schema of magnitude is "number, which is a representation that summarizes the successive addition of one (homogeneous) unit to another" [A142/B182]. The appearances are homogenous since they fall under the same concept, and we know which appearances to count and which to ignore by choosing a particular container in the containment hierarchy.

Now this containment hierarchy is a *necessary aspect* of any spatial representation: if we fix the positions and extensions of objects in 3D space, then the containment hierarchy is also fixed. But, of course, the converse does not hold: specifying the containment hierarchy does not determine all the spatial information. Suppose, for example, that x and y are both in container z. We know that x and y are in the same container, but we do not know if x is above y, or below it. We do not know how near x is to y, etc.

The containment hierarchy is a distinguished sub-structure of the spatial world. If we abstract from our spatial representation all the aspects that are peculiar to our human form of intuition, all that is left is the containment hierarchy. As Kant says:

> "Thus if, e.g., I make the empirical intuition of a house into perception through apprehension of its manifold, my ground is the necessary unity of space and of outer sensible intuition in general, and I as it were draw its shape in agreement with this synthetic unity of the manifold in space. This very same synthetic unity, however, *if I abstract from the form of space, has its seat in the understanding, and is the category of the synthesis of the homogeneous in an intuition* in general, i.e., the category of quantity, with which that synthesis of apprehension, i.e., the perception, must therefore be in thoroughgoing agreement. [B162]

And again:

> The pure *image* of all magnitudes (quantorum) for outer sense is space... The pure *schema* of magnitude (quantitatis), however, as a concept of the understanding, is number. [A142/B182]

Of course, a spatial representation performs many functions. It allows us, for example, to position and orient the parts of our bodies to manipulate other objects. But the function of space that is highlighted in the First Critique is space as the *medium in which appearances are unified*. Now space-qua-unifier-of-intuitions has fewer essential properties than space-qua-form-of-human-outer-sense. Qua unifier of intuitions, the key property of space is that it supports a containment hierarchy, in which we can tell which objects are in which containers. Kant makes it clear, when he first introduces space in the *Aesthetic*, that the function of space that he is focusing on is its ability to support the containment hierarchy:

> For in order for certain sensations to be related to something outside me (i.e., to something in another place in space from that in which I find myself), thus in order for me to represent them as outside one another, thus not merely as different but as in different places, the representation of space must already be their ground) [A23/B38]

Space, qua unifier, is just the medium in which appearance can be placed together, the medium that allows me to infer from "I am intuiting x" and "I am intuiting y" to "I am intuiting x and y." This abstract unifying space just is the containment hierarchy: "space is the representation of coexistence (juxtaposition)"[A374].

To summarize, although Kant's notion of space was the standard (at the time) three-dimensional space of Euclidean geometry (B41), when he was thinking of space as the medium in which appearances can be unified, he focused on a substructure in which many of the features of space have been abstracted away: the containment hierarchy.[26]

The unity condition for comparison[27] simply requires that:

> (5)(b) The comparison operator < forms a strict partial order.

Of course, we do not insist that < is a *total* order: although the dirtiness of this jumper can be compared with the dirtiness of this mug, the *weight* of this jumper need not be comparable with the dirtiness of this mug.

[26] For a related position, see Waxman [Wax14] Section 4B: "It as if the mere use of the word 'space' is enough for many to reflexively read into Kant's doctrine virtually every meaning commonly attached to the term, or at least everything one supposes to remain after factoring in the adjective 'pure'. It becomes a space with all the features attributed to it by Euclid or Newton and so a space a priori incompatible with the features that have been or will be ascribed to space by later mathematicians and physicists. But... the unity of sensibility clearly does not require that pure space be determinately flat hyperbolic or elliptical, three-dimensional or ten-dimensional or any other number of dimensions, Ricci-flat or Ricci-curved, etc."

[27] See [A143/B182-3] and [A168/B210].

We do not, also, insist that $<$ is *dense*.[28] This is because we follow Kant in wanting to allow *finite* models.[29]

6.6 The unity conditions for the synthesis of dynamical relations

The dependency of the dynamical relations on judgement is perhaps the most important, the most original, and the most difficult part of the *Transcendental Analytic*. In fact, one of the major reasons that Kant rewrote the *Transcendental Deduction* in the B edition is precisely to re-express this condition as clearly as possible. In this section, I shall first explain Kant's general strategy before going into the specific details of how he handles each of the pure dynamical relations.

Kant was dissatisfied with the presentation of the Transcendental Deduction in the A edition. In the B edition, he changed the exposition significantly by splitting the proof into two parts (concluding in §20 and §26)[30]. The first part of the Transcendental Deduction, culminating in §20, relies heavily on a new explanation of the categories that was added to §13 in the B edition:

> I will merely precede this with the **explanation of the categories**. They are concepts of an object in general, by means of which its intuition is regarded as **determined** with regard to one of the **logical functions** for judgments. Thus, the function of the **categorical** judgment was that of the relationship of the subject to the predicate, e.g., "All bodies are divisible." Yet in regard to the merely logical use of the understanding it would remain undetermined which of these two concepts will be given the function of the subject and which will be given that of the predicate. For one can also say: "Something divisible is a body." Through the category of substance, however, if I bring the concept of a body under it, it is determined that its empirical intuition in experience must always be considered as subject, never as mere predicate; and likewise with all the other categories. [B128-9]

There are many other places where Kant makes similar claims.[31] What exactly is the claim here, and how exactly does Kant justify it?

Imagine someone trying to connect his intuitions together. Suppose he has "intuition dyslexia" – he is not sure if this intuition is the object and this other intuition is the attribute, or the other way round. Or he has two determinations in a relation of succession, but he is not sure which is earlier and which is later. The intuitions are swimming before his eyes. He needs something that can pin down which

[28] A relation R is dense if Rxy implies there exists a z such that Rxz and Rzy.

[29] [Pin17] page 119.

[30] The first half aims to show that we are always permitted to apply the pure concepts to intuitions, while the second half aims to show that the pure judgements (the synthetic *a priori* claims of the *Principles*) always hold.

[31] For example, in a note added to Kant's copy of the first edition: "Categories are concepts, through which certain intuitions are determined in regard to the synthetic unity of their consciousness as contained under these functions; e.g., what must be thought as subject and not as predicate." He also makes similar claims in the *Metaphysik von Schon*, quoted in *Kant and the Capacity to Judge*, p.251, and *Prolegomena* §20.

intuitions are assigned which roles, but what could perform this function? Kant's fundamental claim is that it is only the *judgement* that can fix the positioning of the intuitions. Moreover, this is not just one role of the judgement amongst many – this is the *primary* role of the judgement:

> a judgment is nothing other than the way to bring given cognitions to the objective unity of apperception [B141]

More specifically, the relative positions of intuitions in a determination can only be fixed by *forming a judgement that necessitates this particular positioning*. This judgement contains concepts that the intuitions fall under, and the position of the intuitions in the determination are *indirectly determined* by the positions of the corresponding intuitions in the judgement. See Figures 6.2 and 6.3. Thus:

> The same function that gives unity to the different representations in a judgment *also gives unity to the mere synthesis of different representations in an intuition*. The same understanding, therefore, and indeed by means of the very same actions through which it brings the logical form of a judgment into concepts by means of the analytical unity, also brings a transcendental content into its representations by means of the synthetic unity of the manifold in intuition in general. [A79/B104-5]

There is a parallel claim one level up, at the level of complex judgements: the relative positions of determinations in a connection can only be fixed by forming a complex judgement that itself contains a pair of judgements as constituents[32] that necessitates this particular positioning. This complex judgement contains two constituents – judgements – that the two determinations fall under, and the position of the determinations in the connection are indirectly determined by the positions of the corresponding judgements in the complex judgement.

What justification does Kant provide for this claim? His argument goes something like this: the aim of the dynamical relations is to order the intuitions and determinations in *objective* space-time. Now we can only achieve objectivity by imposing *necessity* on the combination.[33] But the faculty of imagination is entirely incapable of imposing necessity. All the imagination can do is connect the intuitions using the pure relations – it cannot impose necessity on those connections.[34] In fact, the only element that can provide the desired necessity is the judgement.[35] Thus, the only way dynamical relations can be ordered in objective space-time is by indirectly positioning them, using judgements that impose the necessity that the connections require.[36]

[32] Kant is emphatic on this point: " hypothetical and disjunctive judgments do not contain a relation of concepts but of judgments themselves." [B141]

[33] "Our thought of the relation of all cognition to its object carries something of necessity with it." [A104] The concept of an object is "the concept of something in which [the appearances] are necessarily connected" [A108]

[34] "Apprehension is only a juxtaposition of the manifold of empirical intuition, but no representation of the necessity of the combined existence of the appearances that it juxtaposes in space and time is to be encountered in it." [A176/B219]

[35] "This word [the copula "is"] designates the relation of the representations to the original apperception and its *necessary*

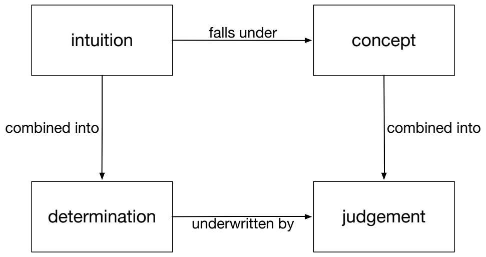


Figure 6.2: Intuitions are combined into determinations, just as concepts are combined into judgements. An intuition falls under a concept, just as a determination is underwritten by a judgement.

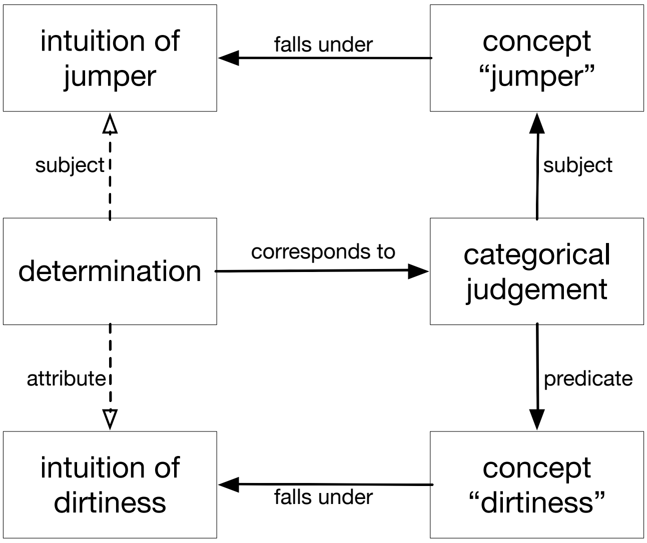


Figure 6.3: Here, the imagination wants to construct an inherence determination involving an intuition of a particular jumper and an intuition of a particular dirtiness. But it does not know whether the jumper intuition should be the subject of the determination, and the dirtiness should be the attribute, or the other way round. The imagination itself does not have the resources to resolve this indecision, but the understanding – the capacity to judge together with the power of judgement – can answer this question. The capacity to judge constructs a categorical judgement, "some jumper is dirty", that corresponds to the determination. The power of judgement decides that my intuition of this jumper falls under the concept "'jumper", and my intuition of this particular dirtiness falls under the concept "dirty". Now, given these assignments, the relative positions of the intuitions in the determination are fixed, indirectly determined by the corresponding positions of the concepts in the judgement. The dashed arrows are determined by the solid arrows.

In terms of the cognitive faculties responsible for the various processes, the **capacity to judge**[37] is responsible for constructing the judgements, and the faculty of **the power of judgement**[38] is responsible for constructing the subsumptions that decide which intuitions fall under which concepts.

This, then, is the general claim, as it applies to all the dynamical relations. Next, I shall describe the various forms of judgement that are needed to underwrite the various dynamical relations: inherence, succession, simultaneity, and incompatibility.

6.6.1 Inherence must be backed up by a categorical judgement

The first of the four conditions of dynamical unity is that the positions of intuitions in an inherence determination must be backed up by a corresponding judgement[39] :

> (6)(a) If I form an inherence determination, ascribing a particular attribute a to a particular object o, then I must be committed to a judgement "this/some/all X are P", where o falls under X, and a falls under P.

Suppose, for example, I am seeing the particular dirtiness of this particular jumper. This inherence determination is a combination of two bare particulars: this particular jumper and this particular instantiation of dirtiness. Now it is essential, in seeing the inherence correctly, that this particular dirtiness is the attribute and this particular jumper is the object in which the attribute inheres. Things would be very different indeed if the intuition of the dirtiness is the object, and the intuition of the jumper is the attribute.[40]

Kant's fundamental claim is that it is only because I form some corresponding categorical judgement that I am able to fix the positions of the two arguments of the inherence operator *det* [B128-9]. In this case, suppose I have formed the judgement "Some jumper is dirty." Now my intuition of this particular jumper falls under the concept "jumper", and my intuition of this particular dirtiness (of this particular jumper at this particular moment) falls under the concept "dirty". Thus, I am able to fix the positions of the two arguments to the inherence operator indirectly, via the judgement and the

unity, even if the judgement itself is empirical, hence contingent." [B142]

[36] Here, the agent "binds" itself in two distinct but related senses. First, it binds its intuitions together via the pure relations. But this binding at the intuitive sensible level must be underwritten by a second binding at the conceptual discursive level: it is only because the agent binds itself to a rule relating concepts that the binding of intuitions achieves the necessity required for objectivity. See [ESS19].

[37] See [A81/B106] and [Lon98].

[38] See [A132/B171] and [Kan90].

[39] In each of the unity conditions that follow, I restrict to the case of unary predicates. The extension to binary, ternary, and so on is straightforward but complicates the presentation.

[40] It is perhaps tempting to argue that it is just obvious which is the attribute and which is the object of the inherence: we can tell from the *types* of the two intuitions which one is which. Above, I said that there are two types of intuitions: intuitions of objects and intuitions of particular attributes. But this distinction only applies *after* a judgement has been constructed which allows the intuitions to be positioned; before that, these intuitions are not yet dignified with these roles as intuitions of objects or intuitions of particular attributes; they are just indeterminate intuitions. In other words, this response just begs the question, assuming that we have already access to the very positioning assignments that we are struggling to achieve.

falls-under relation. I see the positions of the intuitions in the inherence *through* the corresponding judgement.

Now of course I do not need to use that precise judgement “Some jumper is dirty” to fix the positions of the intuitions in the inherence determination. I could have used “Some jumper is revolting”, or ”This jumper is dirty”, and so on and so forth. All that is needed is *some* categorical judgement where the two intuitions fall under the two concepts.

6.6.2 Succession must be backed up by a causal judgement

The second condition of dynamical unity is that every succession of determinations must be backed up by a causal judgement:

> (6)(b) If I form a succession, in which one determination (say, particular object o having particular attribute a) is followed by another determination (say, o having incompatible attribute b), then I must have formed a conditional judgement “If $\phi(X)$ holds and X is P then X becomes Q at the next time-step”, where object o falls under concept X, attribute a falls under concept P, attribute b falls under concept Q, and $\phi(X)$ is an sentence featuring free variable X.

Suppose, for example, I see the jumper’s cleanliness followed by the jumper’s dirtiness. It is essential, when seeing this succession, that I see the order correctly. Seeing the cleanliness followed by the dirtiness is very different from seeing the dirtiness followed by the cleanliness.

Kant claims[41] that it is only because I form some corresponding causal judgement that I am able to fix the positions of the two determinations in the succession relation [A189/B232]. Suppose, for example, I have formed the causal rule that if I wallow about in the mud, then the clothing I wear will transform from clean to dirty. Now my intuition of this jumper falls under the concept “clothing”, my intuition of this particular cleanliness falls under the concept “clean”, and my intuition of this particular dirtiness falls under the concept “dirty”. Thus, I am able to fix the positions of the two determinations in the succession relation indirectly, via the causal judgement and the falls-under relation.

6.6.3 Simultaneity must be backed up by a pair of causal judgements

The third condition of dynamical unity is that every simultaneity of determinations must be backed up by a pair of causal judgements:

[41] Not all commentators agree with this way of reading Kant. Beatrice Longuenesse, for example, believes that we do not have to have already formed a causal judgement – we just need to acknowledge that we should form a causal judgement. For Longuenesse, perceiving a succession means being committed to look for a causal rule – it does not mean that I need to have already found one [Lon98].

(6)(c) If I form a simultaneity, in which one determination (say, particular object o_1 having particular attribute a) is simultaneous with another determination (say, object o_2 having attribute b), then there must be a pair of causal judgements, one of which states that an attribute of o_1 (simultaneous with a) causally depends on an attribute of o_2, and another of which states that an attribute of o_2 (simultaneous with b) causally depends on an attribute of o_1.

Suppose, for example, I have two determinations simultaneously, one involving the sun, and one involving the moon. Now since simultaneity is a symmetric relation, it does not matter which of the two determinations is placed where in the *sim* relation. But it does matter whether we ascribe simultaneity or succession to the pair of determinations. When we are presented with a subjective succession of determinations, should we ascribe them to the same moment (of objective time) or to two successive moments (of objective time)?[42]

Kant's claim here is that in order to choose simultaneity over succession, we need to form a pair of judgements describing, for both objects, how some attribute of that object causally depends on some attribute of the other [A212/B259]. I do not dwell on this principle, because it is the most controversial[43], hard to understand, and does not feature in our computer implementation.

6.6.4 Incompatibility must be backed up by a disjunctive judgement

Kant talks throughout the *Postulates* about the possibility of an *object* - not of the possibility of a sentence being true. It is easy to see this as a category error, or as elliptical: perhaps "the object is possible" is short-hand for "it is possible that the object exists"? This temptation must be resisted. Kant predicates possibility/actuality/necessity of *determinations* as well as of judgements. When we connect two determinations with the *inc* connective, we are making a modal connection between two elements, two ways of seeing, elements that *do not have a truth value*.

Kant claims[44] that every incompatibility between determinations must always be backed up by a disjunctive[45] judgement:

(6)(d) If I form an incompatibility in which one determination (say, particular object o having attribute a) is incompatible with another (say, particular object o having attribute b), then I must have formed a judgement "All X are either (exclusive disjunction) P or Q or ...", in which o falls under X, a falls under P, and b falls under Q.

[42] "The apprehension of the manifold of appearance is always successive. The representations of the parts succeed one another. Whether they also succeed in the object is a second point for reflection, which is not contained in the first." [A189/B234]

[43] See e.g., [Lon98] p.388.

[44] "The schema of possibility is the agreement of the synthesis of various representations with the conditions of time in general (e.g., since opposites cannot exist in one thing at the same time, they can only exist one after another)." [A144/B184]

[45] Recall that for Kant, disjunctions are *exclusive*: "p or q" means either p or q but not both.

Suppose, for example, I see this jumper's cleanliness as incompatible with the jumper's dirtiness. Now this is, to repeat, an incompatibility between determinations, ways of seeing, not an incompatibility between *judgements*. But Kant claims that this incompatibility between determinations must be underwritten by an exclusive-or disjunctive judgement. Suppose, for example, I have formed the judgement that every article of clothing is either clean or dirty. Now my intuition of this particular cleanliness falls under the concept "clean", my intuition of this particular dirtiness falls under the concept "dirty", and my intuition of this particular jumper falls under the concept "article of clothing." Thus, the judgement (expressing an incompatibility between concepts) justifies the relation between determinations.

6.7 Making concepts sensible

As well as the unity condition requiring that determinations are underwritten by judgements, there are also unity conditions in the other direction, requiring that judgements are supported by corresponding determinations.

> It is thus just as necessary to make the mind's concepts sensible (i.e., to add an object to them in intuition) as it is to makes its intuitions understandable (i.e., to bring them under concepts).[A51/B75]

The requirement here is that judgements cannot "float free" of the underlying intuitions. Instead, each judgement must be backed up by a corresponding determination.

More specifically (and restricting ourselves to unary predicates):

> (7) If I form a judgement, ascribing a concept P to a particular object X, then there must be a corresponding inherence determination ascribing particular attribute a to particular object o, where o falls under X and a falls under P.

It might seem that this condition is trivially satisfied given that the agent starts with intuitions and determinations, and forms judgements to make them intelligible. But this is not always so: sometimes the agent constructs new *invented objects* to make sense of the sensible given and ascribes properties to these invented objects (see Example 7). In such cases, condition (7) requires that as well as subsuming object o under concept P, there is also a corresponding particular individual attribute a that inheres in o. The experiment below in Section 6.12 shows just such an example where an invented object is postulated, and particular individual attributes of that object are posited in imagination to make the concepts sensible.

6.8 Conceptual unity

In addition to the synthetic unity described above, Kant also requires that one's concepts are unified by being connected together via judgements. I shall first consider a weak form of this constraint, before describing a stronger version.

A judgement connects various concepts together. For example, the judgement "some bodies are divisible" connects the concepts of "body" and "divisible". Let us say two concepts are *together* if there is some judgement in which they both feature. Define *together** as the transitive closure of *together*. Now the weak constraint of conceptual unity is that every pair of concepts are *together**.

Kant uses a significantly stronger constraint. His requirement is that the concepts are not just connected, but that they are connected into a *hierarchy* of genera and species[46]. In order that one's concepts form a *system* in this sense, we focus exclusively on the judgement form of exclusive disjunction [A70/B95]. Consider a judgement of the form "every X is either (exclusive) P or Q". This does not merely state that P and Q are exclusive; it also states that P and Q form a *totality*: the totality of concepts that together capture X. By bringing concepts under the *xor* judgement form, we bring them into a hierarchical community with a genera-species structure[47].

The condition of conceptual unity is the requirement that:

(8) Every concept features in some disjunctive judgement.

6.9 Achieving synthetic unity

It is time to take stock. For Kant, the fundamental mental representation is the intuition, a representation of a particular object or a particular attribute of a particular object. All the other types of representation serve only to unify the intuitions into a coherent whole.

Intuitions can be combined into determinations using the three pure operations of containment, comparison, and inherence. Further, determinations can be combined into connections using the pure relations of succession, simultaneity, and incompatibility. (See Section 6.3).

In order for the connections of determinations to achieve unity[48], multiple conditions must be sat-

[46] See [Lon98, p.105].

[47] "What the form of disjunctive judgment may do is contribute to the acts of forming categorical and hypothetical judgments the perspective of their possible systematic unity", [Lon98], p.105.

[48] In Section §16 of the B deduction, Kant distinguishes four types of unity using two cross-cutting distinctions: analytic versus synthetic unity, on the one hand, and original versus empirical unity, on the other. Analytic unity is achieved when the mind has the ability to subsume each of its intuitions and determinations under the unary predicate "I think". Synthetic unity is achieved when the intuitions and determinations are connected together via the pure relations of Section 6.3 in such a way as to satisfy the unity conditions of Sections 6.5, 6.6, 6.7, and 6.8. Synthetic unity is the more fundamental concept, as it is presupposed by analytic unity [B133]. The distinction between empirical and original unity is the difference between a particular unity achieved by a particular mind when confronted with a particular sensory sequence, and what is in common between all unities achieved by all minds no matter which sensory sequence they are provided with. In this thesis, I focus on the general conditions common to all minds when achieving synthetic unity.

isfied. The mathematical operations (of containment and comparison) must form a structure of the appropriate sort (Section 6.5), the dynamical functions (of inherence, succession, simultaneity, and incompatibility) must be underwritten by judgements of the appropriate sort (Section 6.6), the judgements must be underwritten by determinations of the appropriate sort (Section 6.7), and the concepts used in judgements must form their own unity (Section 6.8).

Why these unity conditions in particular? One of the remarkable things about Kant's philosophy is its systematicity. Instead of being content with merely enumerating the pure concepts of the understanding, Kant insists on showing how the pure concepts form a *system*, by showing that these are all and only the *a priori* concepts needed to make sense of experience.[49] The same systematicity requirement applies to the unity conditions: he must show that these are *all and only* the unity conditions needed for the synthesis of apprehension to achieve objectivity. To see that the unity conditions described above form a system, observe that there are two realms of cognition: the sensible intuitions and the discursive concepts. There are exactly four possible conditions involving these two realms: (i) a requirement that the intuitions achieve their own individual unity, (ii) a requirement that the intuitive realm respects the conceptual, (iii) a requirement that the conceptual realm respects the intuitive, and (iv) a requirement that the conceptual realm achieves its own individual unity. Here, (i) is the requirement that the synthesis of apprehension forms a fully connected graph satisfying 5(a) and 5(b) (Section 6.5). Condition (ii) is the requirement that the connections between intuitions are underwritten by corresponding judgements (Section 6.6). Condition (iii) is the requirement that the judgements respect the intuitions (Section 6.7). The final condition (iv) is the requirement that the discursive realm of judgement achieves conceptual unity (Section 6.8).

If the agent does all these things, and satisfies all these conditions, then it has achieved *experience*: it has combined the plurality of sensory inputs into a coherent representation of a single world. Achieving experience requires four faculties: sensibility (to receive intuitions), the imagination (to connect intuitions together using the pure relations as glue), the capacity to judge (to generate judgements), and the power of judgement (to decide whether an intuition falls under a concept).

How does this interpretation relate to the debate between conceptualism and non-conceptualism? According to our interpretation, intuitions are formed by sensibility, entirely independently of the understanding.[50] Further, intuitions can be connected (via the pure relations of Section 6.3) by the imagination, without the need for the understanding.[51] But intuitions can only constitute *experience* if the intuitions are brought under concepts (via the power of judgement) and the concepts are combined into judgements (via the capacity to judge): experience requires understanding working in concert with sensibility and the imagination to bring the connected intuitions into a unity. Thus,

[49] See [Lon98, p.105].

[50] "Appearances can certainly be given in intuition without functions of the understanding." [A90/B122]. "The manifold for intuition must already be given prior to the synthesis of the understanding and independently from it." [B145]

[51] "Synthesis in general is, as we shall subsequently see, the mere effect of the imagination, of a blind though indispensable function of the soul, without which we would have no cognition at all, but of which we are seldom even conscious" [A78/B103].

both sensibility and understanding need each other if they are to jointly achieve experience.[52]

Here are the core claims, brought together in one place for ease of reference:

1. In order to achieve experience, I must unify my intuitions.
2. Unifying intuitions means combining them using binary relations to form a connected graph, in such a way as to satisfy the various unity conditions.
3. Synthesis involves (i) connecting intuitions together via containment, comparison, and inherence operations to form determinations; and (ii) connecting determinations together via succession, simultaneity, and incompatibility relations.
4. There are, in total, four types of unity condition that Kant imposes: (i) the unity conditions for the synthesis of mathematical relations, (ii) the unity conditions for the synthesis of dynamical relations, (iii) the requirement that the judgements are underwritten by determinations, and (iv) the conceptual unity condition.
5. The unity conditions for the synthesis of mathematical relations are:
   (a) There exists some intuition x such that for each object of intuition y, for each moment in time, there is a chain of *in* determinations between y and x.
   (b) The comparison operator $<$ forms a strict partial order.
6. The unity conditions for the synthesis of dynamical relations are:
   (a) If I form an inherence determination, ascribing a particular attribute a to a particular object o, then I must be committed to a judgement "this/some/all X are P", where o falls under X, and a falls under P.
   (b) If I form a succession, in which one determination (say, particular object o having particular attribute a) is followed by another determination (say, o having incompatible attribute b), then I must have formed a conditional judgement "If $\phi(X)$ holds and X is P then X becomes Q at the next time-step", where object o falls under concept X, attribute a falls under concept P, attribute b falls under concept Q, and $\phi(X)$ is an sentence featuring free variable X.
   (c) If I form a simultaneity, in which one determination (say, particular object o_1 having particular attribute a) is simultaneous with another determination (say, object o_2 having attribute b), then there must be a pair of causal judgements, one of which states that an attribute of o_1 causally depends on an attribute of o_2, and another of which states that an attribute of o_2 causally depends on an attribute of o_1.

[52] "Thoughts without content are empty, intuitions without concepts are blind." [A50-51/B74-76]. But note the striking asymmetry between the types of deficiency when one activity is performed without the other: *blindness* is a deficiency of a living conscious being, while *emptiness* is a deficiency of a mere *container*. This asymmetry confirms the interpretation in Section 6.2.2 that unity of intuition is the final end of all thought, and conceptual thought is merely a means to that end.

(d) If I form an incompatibility in which one determination (say, particular object o having attribute a) is incompatible with another (say, particular object o having attribute b), then I must have formed a judgement "All X are either (exclusive disjunction) P or Q or ...", in which o falls under X, a falls under P, and b falls under Q.

7. The requirement that the conceptual realm respects the intuitive is the condition that if I form a judgement, ascribing a concept P to a particular object X, then there must be a corresponding inherence determination ascribing particular attribute a to particular object o, where o falls under X and a falls under P.
8. The unity condition for conceptual unity is the requirement that every concept must feature in some disjunctive judgement.

In this section, I shall formalise the task of achieving synthetic unity of apperception. The formalism introduced is necessary for the derivation of the categories below.

6.9.1 The pure relations

Let $\mathcal{I}$ be the set of intuitions, $\mathcal{D}$ the set of determinations, and C the set of connections. The signature of the three pure operations of containment, comparison, and inherence are:

$$\begin{aligned} in &: \mathcal{I} \times \mathcal{I} \rightarrow \mathcal{D} \\ < &: \mathcal{I} \times \mathcal{I} \rightarrow \mathcal{D} \\ det &: \mathcal{I} \times \mathcal{I} \rightarrow \mathcal{D} \end{aligned}$$

The signature of the three pure relations of succession, simultaneity, and incompatibility are:

$$\begin{aligned} succ &: \mathcal{D} \times \mathcal{D} \rightarrow C \\ sim &: \mathcal{D} \times \mathcal{D} \rightarrow C \\ inc &: \mathcal{D} \times \mathcal{D} \rightarrow C \end{aligned}$$

For example, if a, b, c are intuitions of type $\mathcal{I}$, then $det(a, b)$, $in(a, b)$, and $b < c$ are determinations of type $\mathcal{D}$; and $succ(det(a, b), det(a, c))$ and $sim(in(a, b), b < c))$ are connections of type C.

Now *succ* is stipulated to be a *functional relation*: there is at most one Y such that $succ(X, Y)$.[53] To get from this functional *succ* to the traditional successor (where a determination is succeeded by *many* determinations), we define non-functional $succ'$ as the closure of *succ* under the rule:

$$succ(X, Y) \wedge sim(X, X_2) \wedge sim(Y, Y_2) \rightarrow succ'(X_2, Y_2)$$

We insist that *sim* and $succ'$ are well-behaved in that:

[53] The reason for this will become clear when deriving the category of *Cause* (Section 6.10).

- *sim* is an equivalence relation
- *sim* is closed under the rules:

$$
\begin{aligned}
succ'(X,Y) \wedge succ'(X_2,Y) &\rightarrow sim(X,X_2) \\
succ'(X,Y) \wedge succ'(X,Y_2) &\rightarrow sim(Y,Y_2)
\end{aligned}
$$

- Let $succ^*$ be the transitive closure of $succ'$. There is no X such that $succ^*(X,X)$.

Given these constraints, a set of *succ*, *sim* connections induces a sequence $(S_1, ..., S_n)$ of **states** (i.e. maximal sets of simultaneous determinations).[54] For example, given determinations $p_1 ... p_{10}$ and connections:

$$
\begin{array}{lll}
sim(p_1,p_2) & sim(p_2,p_3) & succ(p_3,p_4) \\
sim(p_4,p_5) & succ(p_5,p_6) & succ(p_6,p_7) \\
sim(p_7,p_8) & sim(p_7,p_9) & succ(p_7,p_{10})
\end{array}
$$

we induce the sequence of states $(S_1, ..., S_4)$ where:

$$
\begin{aligned}
S_1 &= \{p_1,p_2,p_3\} \\
S_2 &= \{p_4,p_5\} \\
S_3 &= \{p_6\} \\
S_4 &= \{p_7,p_8,p_9,p_{10}\}
\end{aligned}
$$

6.9.2 Achieving synthetic unity

The input that the mind receives from sensibility is a sequence $(\pi_1, ..., \pi_t)$ of individual determinations from $\mathcal{D}$. Note that the input is not a sequence of *sets* of determinations that are already assumed to be simultaneous, but a sequence of *individual* determinations. Kant insists on this:

> The apprehension of the manifold of appearance is always successive. The representations of the parts succeed one another. Whether they also succeed in the object is a second point for reflection, which is not contained in the first... Thus, e.g., the apprehension of the manifold in the appearance of a house that stands before me is successive. Now the question is whether the manifold of this house itself is also successive, which certainly no one will concede. [A189/B234ff]

Here, Kant asks us to imagine an agent surveying a large house from close range. Its visual field cannot take in the whole house in one glance, so its focus moves from one part of the house to

[54] Kamp has a similar construction from *succ* and *overlaps* [Kam79].

another. Its sequence of visual impressions is successive, but there is a further question whether a pair of (subjectively) successive visual impressions represents the house at a single moment of objective time, or at two successive moments of objective time.[55]

Given a sequence $(\pi_1, ..., \pi_t)$ of individual determinations, the task of making sense of sensory input is is to construct a synthetic unity - a triple $(\kappa, \upsilon, \theta)$ - satisfying various conditions, where:

- $\kappa \subseteq C$ is a set of connections between determinations
- $\upsilon \subseteq \mathcal{I} \times P_1$ is the falls-under relation (also known as subsumption) between intuitions and unary predicates P_1
- θ is a collection of judgements

The connections κ are generated by the faculty of *imagination*. Note that not all the determinations in κ need come from the original sequence $(\pi_1, ..., \pi_t)$. Some of the determinations may involve new invented objects constructed by *pure intuition* (for spaces and times) or by the *imagination* (for hypothesised unperceived empirical objects). The connections must satisfy the following conditions:

- If π_i, π_{i+1} are successive determinations in $(\pi_1, ..., \pi_t)$, then either $sim(\pi_i, \pi_{i+1})$ or $succ(\pi_i, \pi_{i+1})$ must be in κ
- The determinations are fully connected: every determination in κ is connected to every other determination via some path of undirected edges.

While the falls-under relation υ is generated by *the power of judgement*, the theory θ is a collection of judgements that is generated by the *capacity to judge*. The language of judgements is formalized in Chapter 3, but in brief: judgements are either rules or constraints. Rules are either arrow rules $\alpha_1 \wedge ... \alpha_n \rightarrow \alpha_0$ (stating that if $\alpha_1, ..., \alpha_n$ all hold, then α_0 also holds at the same time-step), or causal rules $\alpha_1 \wedge ... \alpha_n \ni\!\!\!- \alpha_0$ (stating that if $\alpha_1, ..., \alpha_n$ all hold, then α_0 also holds at the *next* time-step). Constraints are either *xor* judgements $\alpha_1 \oplus ... \oplus \alpha_n$ (stating that exactly one of the α_i hold) or a uniqueness constraint $\forall X, \exists! Y, r(X, Y)$ (stating that for each X there is exactly one Y such that $r(X, Y)$).

Figure 6.4 shows two different ways of grouping the four faculties, according to two cross-cutting distinctions. According to one distinction, sensation and imagination both fall under *sensibility* because both faculties process intuitions.[56] The power of judgement and the capacity to judge both fall under the *understanding* because both faculties process concepts. According to the other distinction, sensation falls under *receptivity* because it is a purely passive capacity that merely receives what it is given. The other three faculties fall under *spontaneity*[57] because the agent is free to construct *whatsoever it pleases*, as long as the resulting construction satisfies the various unity conditions.

[55] See also [Lon98, p.359].

[56] "Now since all of our intuition is sensible, the imagination, on account of the subjective condition under which alone it can give a corresponding intuition to the concepts of understanding, belongs to **sensibility**." [B151]

[57] See [A51/B75], [B133], [B151].

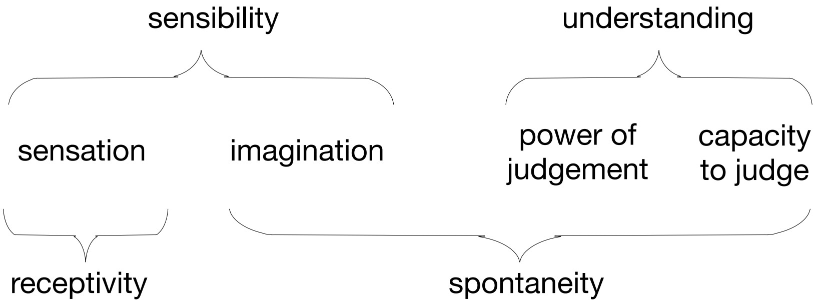


Figure 6.4: The relationship between the four faculties

We have now assembled the materials needed to define the task of synthetic unity.

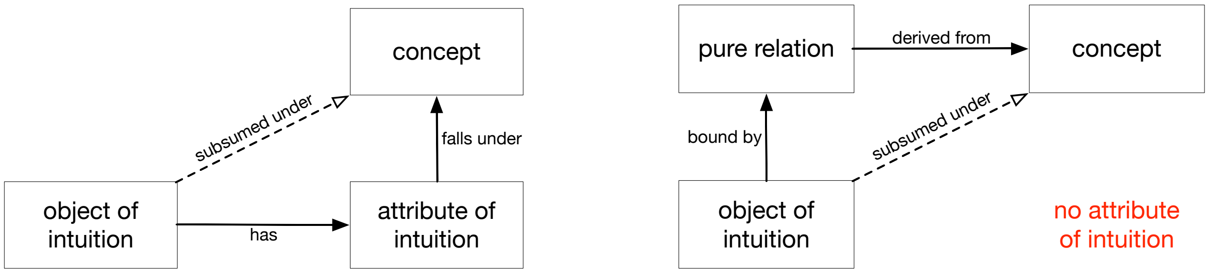
Definition 28. *Given a sequence* $(\pi_1, ..., \pi_t)$ *of determinations, the* **task of achieving synthetic unity of apperception** *is to construct a triple* $(\kappa, \upsilon, \theta)$ *as described above that satisfies the unity conditions of Sections 6.5, 6.6, 6.7, and 6.8.* △

6.10 The derivation of the categories

The problem of the pure categories is explained in the opening paragraphs of the *Schematism*:

> In all subsumptions of an object under a concept the representations of the former must be **homogeneous** with the latter, i.e., the concept must contain that which is represented in the object that is to be subsumed under it, for that is just what is meant by the expression "an object is contained under a concept." ... Now pure concepts of the understanding, however, in comparison with empirical (indeed in general sensible) intuitions, are entirely unhomogeneous, and can never be encountered in any intuition. Now how is the **subsumption** of the latter under the former, thus the **application** of the category to appearances possible, since no one would say that the category, e.g., causality, could also be intuited through the senses and is contained in the appearance? [A137/B176 ff]

For empirical concepts, an object's being subsumed under a concept can be explained in terms of a particular attribute that the object has which falls under the concept. See Figure 6.5(a). Suppose, for example, my intuition of this particular jumper is subsumed under the concept "dirty". This subsumption is explained by (i) the object of intuition having, as one of its determinations, a particular attribute of intuition (my representation of the particular dirtiness of this particular jumper at this particular moment), and (ii) the attribute of intuition falling under the concept "dirty". The problem, for the pure concepts such as *Unity*, *Reality*, *Substance*, and so on, is that there is no corresponding attribute of intuition, so the explanation of the subsumption in Figure 6.5(a) is not applicable. What, then, justifies or permits us to subsume the objects of intuition under the pure concepts?



(a) The explanation of subsumption for an empirical concept

(b) The explanation of subsumption for a pure concept

Figure 6.5: Both diagrams provide an explanation for an object being subsumed under a concept. In (a), the concept is empirical, and the explanation goes via the intermediary of an attribute of intuition. In (b), the concept is pure, there is no corresponding attribute, and the explanation goes via the another intermediary: a pure relation.

According to Kant, what justifies my subsuming an object under a pure concept is the existence of a *pure relation*[58] that the object is bound to. See Figure 6.5(b). Here, the subsumption of the object under the pure concept is explained by (i) the object of intuition being bound to the pure relation, and (ii) the pure concept being derivable from the pure relation. Note that in both Figures 6.5(a) and (b) there is an intermediary that explains the object being subsumed under a concept, but it is a different sort of intermediary in the two cases:

> Now it is clear that there must be a third thing, which must stand in homogeneity with the category on the one hand and the appearance on the other, and makes possible the application of the former to the latter. This mediating representation must be pure (without anything empirical) and yet intellectual on the one hand and sensible on the other. Such a representation is the transcendental schema. [A138/B177]

The "transcendental schema" is just another term for what I have been calling a pure relation: *in*, $<$, *det*, *succ*, *sim*, and *inc*.

This, then, is the outline of Kant's argument explaining how the pure concepts (categories) apply to objects of intuition. The next stage is to show, in detail, for each pure concept, exactly how it is derived from the corresponding pure relation.

The derivation is straightforward and Kant did not see the need to spell it out.[59] But for the sake of maximal explicitness, we shall go through each in turn.

[58] I.e. one of the six pure relations introduced in Section 6.3 and defined in Section 6.9.1.

[59] In [Bra09], Brandom describes how new unary concepts can be derived from given relations. So, for example, if we have the binary relation $P(x, y)$ representing that x admires y, then we can form the new unary predicate $Q(x)$ defined as $Q(x) = R(x, x)$. Here, $Q(x)$ is true if x is a self-admirer. In a similar manner, the unary categories are derived from the pure relations of Section 6.3.

Starting with the title of *Relation*, intuition X falls under the pure concept **substance** if there exists an intuition Y such that $det(X, Y)$ is a determination in κ [B128-9]. Likewise, X falls under the pure concept **accident** if there exists an intuition Y such that $det(Y, X)$ is a determination in κ. Determination π falls under the pure concept **cause** if there exists a determination π' such that $succ(\pi, \pi')$ is in κ [A144/B183]. Likewise, determination π falls under the pure concept **dependent** if there exists a determination π' such that $succ(\pi', \pi)$ is in κ.[60] A set Π of determinations falls under the pure concept **community** if for each π, π' in Π, $sim(\pi, \pi')$ is in κ [A144/B183-4].

Moving to the title of *Modality*, a set Π of determinations falls under the pure concept **possible** if there is some sequence of sensor readings, and some theory θ that makes sense of those readings, such that Π is contained in one of the states of the trace of θ [A144/B184]. A set Π of determinations is **actual** if it is contained in one of the states of the trace of the best theory that explains the sensor readings that have been received.[61] A set Π of determinations is **necessary** if if it is contained in every state of the trace of the best theory that explains every possible sensory sequence.

Moving next to the title of *Quality*, intuition X falls under the pure concept of **reality** if there exists an intuition Y such that $Y < X$ [A168/B209]. Likewise, intuition X falls under the pure concept of **negation** if there does not exist an intuition Y such that $Y < X$.

Moving, finally, to the title of *Quantity*, the categories of *Unity*, *Plurality*, and *Totality* are slightly more involved because they are implicitly indexed by a predicate p. A container is a **unity** of p's if it contains all the objects that fall under p. In other words, X falls under the pure concept of unity if for all Y, $(Y, p) \in v$ implies $in(Y, X)$. A container is a **plurality** of p's if all the objects within it fall under p. In other words, X falls under the pure concept of plurality if for all Y, $in(Y, X)$ implies $(Y, p) \in v$. A container is a **totality** of p's if it contains all and only the objects that fall under p.[62]

Returning to the overall argument for the derivation of the categories, Kant's deontic argument can be summarized as:

- Achieving experience requires that I connect the intuitions using the pure relations.
- If I connect the intuitions using the pure relations, then I may apply the pure concepts (the categories) to the objects of intuition.
- Therefore, achieving experience permits me to apply the pure concepts to the objects of intuition.

Thus the *quid juris* question [A84/B116] can be answered in the affirmative. Note, however, that my permission to apply the pure concepts to objects of intuition is conditioned on my *activity*, the activity

[60] This definition relies on *succ* being defined as a functional relation in Section 6.9.1.

[61] "The postulate for cognizing the **actuality** of things requires **perception**, thus sensation of which one is conscious – not immediate perception of the object itself the existence of which is to be cognized, but still its connection with some actual perception." [A225/B272]

[62] Kant says that a totality is a plurality considered as a unity [B111].

of trying to achieve experience. Hence Kant's conclusion that the categories are only permitted to apply to objects of experience.[63]

Kant insisted that the categories are not innate. The pure unary concepts are not "baked in" as primitive unary predicates in the language of thought. The only things that are baked in are the fundamental capacities (sensibility, imagination, power of judgement, and the capacity to judge) together with the pure relations of Section 6.3. The categories themselves are *acquired* – derived from the pure relations *in concreto* when making sense of a particular sensory sequence. But they are *originally* acquired [*Entdeckung*, Ak. VIII, 222-23; 136.][64] because they are *always* derivable from *any* sensory sequence. The pure concepts, then, are not innate but originally acquired [Lon98].[65]

6.11 Kant's cognitive architecture

The first half of the *Critique of Pure Reason* is a sustained exercise in *a priori* psychology: the study of the processes that must be performed if the agent is to achieve experience. For Kant, this *a priori* psychology was largely a means to an end, or ends. In fact, his psychology served two overriding goals. One of his goals was metaphysical: to enumerate once and for all the pure aspects of cognition: those features of cognition that must be in place no matter what sensory input has been received. The pure aspects of cognition include the pure forms of intuition (space and time, as described in the *Aesthetic*), the pure concepts (the categories, as described in the *Analytic of Concepts*), and the pure judgements (the synthetic *a priori* propositions, as described in the *Principles*). His other overriding goal was metaphilosophical: to delimit the bounds of sense, and finally put to rest various interminable disputes[66] (by showing that the pure concepts can only be applied to objects of possible experience).

But I believe that, apart from its role as a means to his metaphysical and metaphilosophical ends, Kant's peculiar brand of psychology has independent interest in its own right, *as a specification of a cognitive architecture*. To test this hypothesis, we need to implement this architecture in a computer program, and test it on a wide array of examples. Kant's theory is intended to be a general theory of what is involved in achieving experience, so – if it actually works – it should apply to any sensory input. To test the viability of this architecture, then, we need to evaluate it in a large and diverse set of experiments.

[63] "The category has no other use for the cognition of things than its application to objects of experience." [B145]

[64] This is quoted in [Lon98].

[65] Some cognitive scientists (e.g. Gary Marcus [Mar18b]) place Kant on the nativist side of the nativist versus empiricist debate. But the key question for Kant is not what humans are born with, but what agents *must do* in order to make sense of the sensory input. It is a normative question of *a priori* psychology, not an empirical question about ontogenetic development. From Kant's perspective, the list of innate concepts proposed by cognitive scientists [SK07] is a "mere rhapsody" [A81/B106] unless they can be unified under a *common principle*. Nativists compile their list of innate concepts by looking at what human babies can do. But the capacities that evolution has hard-wired to help us in our particular situation are not *maximally general*. For example, babies can distinguish faces from other shapes before they are born, but the concept of a face is not a pure concept in Kant's sense.

[66] He wanted to "put an end to all dispute" [A768/B796].

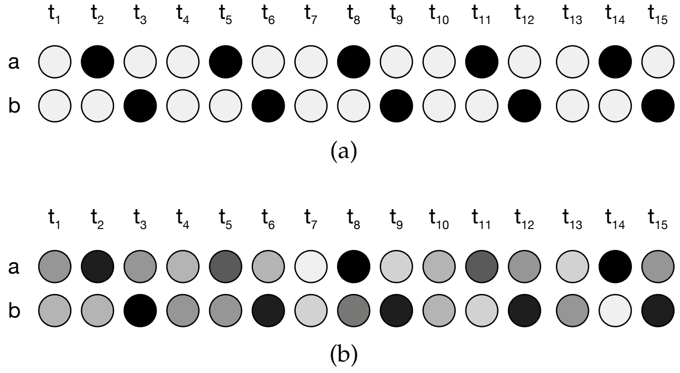

t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15
a
b
(a)
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15
a
b
(b)

Figure 6.6: A simple sequence involving two sensors. (a) shows a noise-free version, where the pattern is clearly apparent. (b) shows the fuzzy version with random noise that is used in this experiment.

Recall that Kant's cognitive architecture involves three capacities: the understanding (generating judgements), the capacity to judge (mapping intuitions to concepts), and the imagination (connecting intuitions together). In Chapter 3, we implemented the understanding, and in Chapter 4 we tested it in a wide variety of domains: cellular automata, sequence induction tasks, rhythms and simple nursery tunes, occlusion tasks, and multimodal binding tasks. In Chapter 5, we also implemented the capacity to judge, and tested it on sequence induction tasks, *Sokoban*, and fuzzy sequences. In the rest of this chapter, we describe our implementation of the imagination, and evaluate the system as a whole.

In our implementation, the three faculties are implemented in one ASP program. The understanding is implemented as an unsupervised program synthesis system, the power of judgement is implemented as a binary neural network (also implemented in ASP), and the productive imagination is implemented as a set of choice rules (also implemented in the same ASP program).

6.12 Experiment 1: flashing lights

I shall describe two experiments showing Kant's theory in action. We first describe the sensory input, and then the interpretation produced by our system.

6.12.1 The sensory input

The sensory input is a noisy version of Example 1.

In this experiment, there are two light sensors that can register various levels of intensity. If we take readings of both sensors at regular intervals, we get Figure 6.6. Here, the top row shows a human-readable discretised version of the sensor readings. The bottom row shows a noisier, fuzzier version of the same pattern. It is this second fuzzier version that is used in this experiment. But the sensory

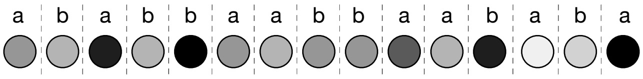


Figure 6.7: The input to the Apperception Engine is a sequence of individual readings. The engine must choose how to group the individual readings into groups of simultaneous readings.

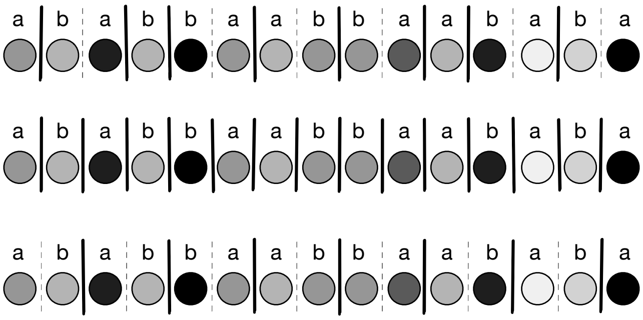


Figure 6.8: We show three ways of parsing the individual readings (in subjective time) into a succession of simultaneous readings (in objective time). The thin dashed lines divide the readings in subjective time, while the thicker lines group the individual readings into sets of simultaneous readings in objective time. The bottom row of the three represents the correct ground-truth way of grouping the readings.

input, as presented in Figure 6.6(b), shows the sensory readings after they have already been assigned to particular moments in time. In Kant's theory, this time-assignment is not something that is given to the system, but rather is a hard-won *achievement*. In Kant's theory, the sensory input is presented as a sequence of *individual* sensory readings, and the agent has to decide how the various readings should be combined together into moments of objective time. So the actual input to the Kantian agent is shown in Figure 6.7. Here, the agent is given a sequence of individual sensory readings, and must choose how to combine them together into a succession of simultaneous readings. While Figure 6.7 shows the sequence of individual readings in subjective time, Figure 6.8 shows a variety of different ways of parsing the raw sequence into moments. The bottom row of Figure 6.8 shows the correct way of parsing the sequence in Figure 6.7; this correct parse corresponds to Figure 6.6(b).

The input, then, is the sequence shown in Figure 6.7. In our implementation, the continuous sensor readings are first discretised into binary vectors of length 3. Thus, the sequence of Figure 6.7 is represented as:

$$\begin{gathered} det(a,[1,0,0]) \\ det(b,[1,0,1]) \\ det(a,[0,0,1]) \\ det(b,[1,0,1]) \\ det(b,[0,0,0]) \\ det(a,[1,0,0]) \\ det(a,[1,0,1]) \\ ... \end{gathered}$$

The total sequence $(d_1,...,d_{50})$ is a list of 50 inherence determinations. Note that the readings do not simply alternate between a and b. Sometimes there are multiple a's or b's in a row. The subjective sequence records the sequence of items the agent is attending to (he can only attend to one sensation at a time), and the agent might attend to either sensor at any moment of subjective time. Given this sequence in subjective time, we must reconstruct the moments of objective time by connecting the determinations using the relations of simultaneity and succession.

6.12.2 The model

Given the sensory sequence, the agent must construct an interpretation that makes sense of the sequence. The interpretation consists of:

1. A **synthesis of intuitions**. This contains a set of determinations (that must include the original sensory sequence, but can also include determinations involving other invented intuitions) connected together via the pure relations of *sim*, *succ*, and *inc*.

2. A **collection of subsumptions**. This is a set of mappings from intuitions of individual objects to general concepts. The mapping is implemented as a binary neural network.

3. A **set of judgements** that connect the concepts together.

In our implementation, each of these three processes are implemented as parts of one large ASP program. The productive imagination is implemented as a choice rule, the power of judgement is implemented as a binary neural network, and the understanding is implemented as an unsupervised program synthesis system.

The synthesis of intuitions

The given sequence $(d_1, ..., d_{50})$ is a sequence of individual determinations in subjective time. We need to produce a sequence of sets of determinations in objective time. For each consecutive pair d_t, d_{t+1}, they can either be simultaneous or successive.

We implement the productive imagination as a choice rule:

```
1 { sim((BV1, Obj1, ST), (BV2, Obj2, ST+1));
    succ((BV1, Obj1, ST), (BV2, Obj2, ST+1)) } 1 :-
        bv_at(BV1, Obj1, ST), bv_at(BV2, Obj2, ST+1).
```

Here, `sim` and `succ` are relations between triples containing the attribute `BV`, the object of intuition `Obj`, and the subjective time index `ST`. We need to include the subjective time index `ST` so that two determinations featuring the same object and the same attribute (but at different times) are not identified. In our example, this choice rule gives us 2^{50-1} possibilities.[67]

Once the *sim* and *succ* relations are provided, this determines the positions of the determinations in objective time:

```
position((BV, Obj, 1), 1) :- bv_at(BV, Obj, 1).

position(X, T) :- position(Y, T), sim(Y, X).

position(X, T+1) :-
    position(Y, T), succ(Y, X), max_subjective_time(MT), T+1 <= MT.

max_subjective_time(MT) :- is_st(MT), not is_st(MT+1).

is_st(ST) :- bv_at(_, _, ST).
```

Here, the first argument of `position` is a triple containing the attribute `BV`, the object of intuition `Obj`, and the subjective time index `ST`. The second argument of `position` is the time index in *objective time*.

We insist that no two distinct readings of the same sensor can be present in the same moment of objective time:

```
:- position((BV1, Obj, _), T), position((BV2, Obj, _), T), BV1 != BV2.
```

The incompossibility relation between determinations is derived from the incompossibility between subsumptions:

[67] The current implementation assumes that any pair of consecutive sensor readings are either simultaneous or successive. This precludes the possibility that there are intermediate time-steps between the two consecutive readings. In future work, I plan to expand the choice rule to allow this further possibility, so that it is possible to abduce intermediate time-steps.

```
inc((BV1, Obj, ST1), (BV2, Obj, ST2)) :-
    bv_at(BV1, Obj, ST1),
    bv_at(BV2, Obj, ST2),
    possible_pred(BV1, P1),
    possible_pred(BV2, P2),
    not is_ambiguous(BV1),
    not is_ambiguous(BV2),
    incompossible(s(P1, Obj), s(P2, Obj)).

is_ambiguous(BV) :-
    possible_pred(BV, P1),
    possible_pred(BV, P2),
    P1 != P2.
```

The set of subsumptions

A subsumption maps an intuition (a bit vector) to a concept (symbol). We implement the power of judgement using a binary neural network parameterised by Boolean weights. (See Section 5.4.1). We use `clingo` to jointly find the weights of the neural network and construct the judgements.

The neural network's input is a binary vector (of length 3 in this experiment) and the output is a binary vector of length $|P|$ (where $|P|$ is the number of unary predicates). The neural network implements a multilabel classifier mapping binary vectors to $2^{|P|}$.

The power of judgement is implemented by the binary network together with the following choice rule implementing the multilabel classifier:

```
1 { senses(s(C, Obj), T) : possible_pred(BV, C) } 1 :-
    position((BV, Obj, _), T).

possible_pred(BV, c_p) :- bnn_result(BV, 1, 1).
possible_pred(BV, c_q) :- bnn_result(BV, 2, 1).
```

This choice rule states that for each object `Obj` that is assigned intuition attribute `BV` at objective time `T`, the object `Obj` must be assigned exactly one of the predicates C_i such that the i'th output bit is 1. In other words, subsume the object `Obj` under one of the unary predicates `C` that is associated with `BV`.

The set of judgements

Kant's faculty of understanding is implemented as a program synthesis system that takes as input a stream of sensory information, and produces a theory (a set of judgements) that both explains the sensory stream and also satisfies various unity conditions.

Filling in the unperceived details

Recall Kant's unity condition that judgements should be underwritten by determinations:

> (7) If I form a judgement, ascribing a concept *P* to a particular object *X*, then there must be a corresponding inherence determination ascribing particular attribute *a* to particular object *o*, where *o* falls under *X* and *a* falls under *P*.

To implement this, we add two choice rules stating that if an object *X* satisfies *p* (respectively *q*) at *T*, then there is some particular attribute *Attr* ascribed to *X* at *T* (where *Attr* falls under *p* (respectively *q*)):

```
1 { obj_bv_at(Attr, X, ObjT) : bnn_result(Attr, 1, 1) } 1 :-
    holds(s(c_p, X), ObjT).

1 { obj_bv_at(Attr, X, ObjT) : bnn_result(Attr, 2, 1) } 1 :-
    holds(s(c_q, X), ObjT).
```

This code relies on the additional predicates connecting subjective with objective time:

```
obj_bv_at(Attr, X, ObjT) :-
    bv_at(Attr, X, SubjT),
    subj_obj_t(SubjT, ObjT).

subj_obj_t(SubjT, ObjT) :- position((_, _, SubjT), ObjT).
```

Finding the best model

When the three sub-systems (the imagination, power of judgement, and understanding) described above are implemented in one system, many different interpretations are found. In order to decide between the various interpretations, we use the following preferences:

1. We prefer shorter theories over longer theories, all other things being equal.

2. We prefer sets of subsumptions which assign fewer intuitions to the same concept.

The first weak constraint penalises theories based on length:

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

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

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

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

Note that ground atoms in the initial conditions are penalised just the same as unground atoms in the rules.

The second weak constraint penalises subsumptions for assigning many intuitions to the same concept.

```
:~ max_bnn_examples_per_predicate(M). [M@4, M]

max_bnn_examples_per_predicate(M) :- M = #max{N : count_bnn_examples_per_predicate(C, N)}.

count_bnn_examples_per_predicate(C, N) :- possible_pred(,C), N = #count{E : possible_pred(E, C)}.
```

See Section 5.3 for a justification of this weak constraint.

The raw apperception framework

In terms of the formalism of Section 5.2, the raw apperception framework $(\pi_{\mathbf{w}}, n, \Delta, \phi, C)$ is:

- $\pi_{\mathbf{w}}$ is a tiny binary neural network of size $3 \times 2 \times 2$. The input layer receives the three bits of the sensory input, the middle layer has just two units, and the output layer has two nodes representing whether the input satisfies the unary predicates p and q. The network has just 10 weights.

- $n = 2$ since there are only two classes: p and q.

- The "disjunctifier" Δ is implemented by the clauses in Section 6.12.2.

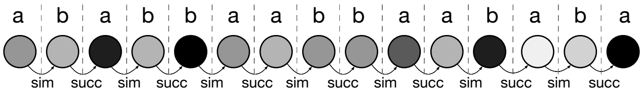


Figure 6.9: How the objective temporal sequence is constructed from the subjective temporal sequence via the pure relations of *sim* and *succ*.

The type signature $\phi = (T, O, P, V)$, where:

$$\begin{aligned} T &= \{sensor\} \\ O &= \{o_1, o_2, o_3\} \\ P &= \{p(sensor), q(sensor), r(sensor, sensor)\} \\ V &= \{X{:}sensor, Y{:}sensor\} \end{aligned}$$

The constraints $C = \{\forall X{:}sensor, p(X) \oplus q(X)\}$.

6.12.3 Results

The interpretation found by the Apperception Engine consists of a triple $(\kappa, \upsilon, \theta)$ consisting of a synthesis of intuitions, a collection of subsumptions, and a set of judgements. We shall consider each in turn.

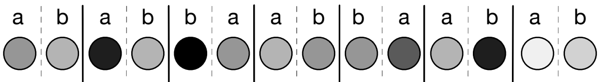
The synthesis of intuitions κ. When confronted with the sensory sequence of Figure 6.7, the engine produces a set κ of connections using the pure relations of *sim*, *succ*, and *inc*. Here is an excerpt:

sim(([1, 0, 0], a, 1), ([1, 0, 1], b, 2))	succ(([1, 0, 1], b, 2), ([0, 0, 1], a, 3))	inc(([1, 0, 0], a, 1), [0, 0, 1], a, 3))
sim(([0, 0, 1], a, 3), ([1, 0, 1], b, 4))	succ(([1, 0, 1], b, 4), ([0, 0, 0], b, 5))	inc(([1, 0, 1], b, 2), ([0, 0, 0], b, 5))
sim(([0, 0, 0], b, 5), ([1, 0, 0], a, 6))	succ(([1, 0, 0], a, 6), ([1, 0, 1], a, 7))	inc(([1, 0, 0], a, 6), ([0, 0, 1], a, 10))
sim(([1, 0, 1], a, 7), ([1, 0, 0], b, 8))	succ(([1, 0, 0], b, 8), ([1, 0, 0], b, 9))	inc(([1, 0, 1], a, 7), ([0, 0, 0], a, 15))

Here, the determinations are triples containing an attribute, an object, and an index in *subjective* time. This index is needed so that two determinations sharing the same object and attribute at different moments of time are nevertheless treated as distinct.

Figure 6.9 shows how the *succ* and *sim* relations produce objective time from subjective time.

The falls-under relation υ. The Apperception Engine constructs two unary predicates, p and q, and subsumes the binary vectors under them. The binary neural network implements a multilabel classifier, mapping binary vectors to subsets of $\{p, q\}$. The subsumptions υ produced by the engine are:



$p(a)$	$q(a)$	$p(a)$	$p(a)$	$q(a)$	$p(a)$	$p(a)$
$p(b)$	$p(b)$	$q(b)$	$p(b)$	$p(b)$	$q(b)$	$p(b)$
$q(c)$	$p(c)$	$p(c)$	$q(c)$	$p(c)$	$p(c)$	$q(c)$

Figure 6.10: The subsumptions generated by the engine. The dashed lines divide subjective time, while the solid lines divide moments of objective time. The atoms generated at each moment are displayed below.

$$\begin{array}{ll} [0,0,0] \mapsto \{q\} & [0,0,1] \mapsto \{q\} \\ [0,1,0] \mapsto \{q\} & [0,1,1] \mapsto \{p,q\} \\ [1,0,0] \mapsto \{p\} & [1,0,1] \mapsto \{p\} \\ [1,1,0] \mapsto \{p\} & [1,1,1] \mapsto \{p\} \end{array}$$

Note that $[0,1,1]$ is considered ambiguous.

Figure 6.13 shows the subsumptions generated by the engine. Note the introduction of an invented object, c, that was not part of the sensory input.

The set of judgements θ. Along with the synthesis of intuitions and the collection of subsumptions, the Apperception Engine also generates a theory θ, a set of judgements that explain the dynamics of the system. The theory constructed for the problem of Figure 6.7 is $\theta = (\phi, I, R, C)$. The type signature ϕ consists of types T, objects O, and predicates P where:

$$\begin{array}{rcl} T & = & \{sensor, space\} \\ O & = & \{a{:}sensor, b{:}sensor, c{:}sensor, s_1{:}space, s_2{:}space, s_3{:}space, s_4{:}space\} \\ P & = & \{p(sensor), q(sensor), in(sensor, space), in_2(space, space), r(space, space)\} \end{array}$$

The initial conditions I, rules R and constraints C are:

$$\begin{array}{rcl} I & = & \left\{ \begin{array}{llll} p(a) & p(b) & q(c) & \\ in(a, s_1) & in(b, s_2) & in(c, s_3) & \\ in_2(s_1, s_w) & in_2(s_2, s_w) & in_2(s_3, s_w) & in_2(s_w, s_w) \\ r(s_1, s_2) & r(s_2, s_3) & r(s_3, s_1) & \end{array} \right\} \\ R & = & \left\{ \begin{array}{l} q(X) \ni\!\!- p(X) \\ in(X, S_1) \wedge in(Y, S_2) \wedge r(S_1, S_2) \wedge q(X) \ni\!\!- q(Y) \end{array} \right\} \\ C & = & \left\{ \begin{array}{l} \forall X{:}sensor,\ p(X) \oplus q(X) \\ \forall X{:}sensor, \exists! Y{:}space,\ in(X, Y) \\ \forall X{:}space, \exists! Y{:}space,\ in_2(X, Y) \\ \forall X{:}sensor, \exists! Y{:}sensor,\ r(X, Y) \end{array} \right\} \end{array}$$

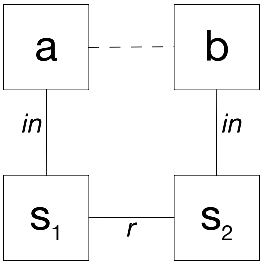


Figure 6.11: Sensors a and b are indirectly connected via the *in* and r relations. The dashed line represents the indirect connection that is derived from the direct connections.

Here, the sensors a and b are given as part of the sensory input, but sensor c is an invented object, constructed by the imagination. The invented objects s_1, s_2, and s_3 are three parts of space, constructed by pure intuition. The three spaces are all parts of the spatial whole s_w.

The unary predicates p and q are used to distinguish between a sensor's being on and off. The *in* relation places sensors in space, and the in_2 relation places spaces inside the spatial whole. The r relation is used to define a one-dimensional space with wraparound.[68] Note that our "spatial unity" requirement is rather minimal: we just insist that there is *some* containment structure connecting the intuitions together. It is not essential that the space constructed has the particular three-dimensional structure that we are accustomed to. Any spatial structure will do as long as the intuitions are unified [Wax14, Chapter 3]. In terms of Kant's distinction between the *form of intuition* and the *formal intuition* [B160n], the relation r describes the form of intuition (relations between objects) while the particular spaces (s_1, s_2, s_3, and s_w) represent the formal intuitions.

Note that the given objects of sensation (the sensors a and b) are not directly related to each other. Rather, they are *indirectly related* via the spatial objects and the *in* and r relations. See Figure 6.11.

The rules describe how the unary properties p and q change over time. The first rule states that objects that satisfy q at one time-step will satisfy p at the next time-step. The second rule describes how the q property moves from one sensor to its right neighbour.

The constraints are constructed to satisfy conceptual unity (Section 6.8). The first insists that every sensor is either p or q but not both. The second requires that every sensor is contained within exactly one spatial region.

Filling in the unperceived details. In order to make concepts sensible (Section 6.7), the engine must ensure there is a determination corresponding to every judgement. In particular, the judgements involving invented unperceived object c must be underwritten by corresponding determinations. This means that for each time step at which $p(c)$ (respectively $q(c)$) is true, there must be an inherence determination $det(c, a)$ ascribing particular attribute a to c, where c falls under p (respectively q).

[68] Note that, in this example, the spatial structure is static. But see e.g. Sections 4.2.5 and 5.5.2 for examples where objects move around.

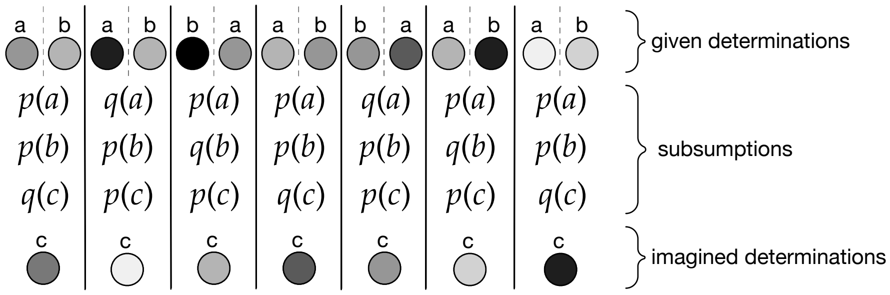

a
b
a
b
b
a
a
b
b
a
a
b
a
b
given determinations
p(a) q(a) p(a) p(a) q(a) p(a) p(a)
p(b) p(b) q(b) p(b) p(b) q(b) p(b)
q(c) p(c) p(c) q(c) p(c) p(c) q(c)
subsumptions
c
c
c
c
c
c
c
imagined determinations

Figure 6.12: The determinations imagined by the engine. Here we show the given determinations (top row), the subsumptions (middle row), and the imagined determinations (bottom row) that are generated to satisfy condition (7): the requirement that every judgement needs to be underwritten by a determination. Thus, for example, the atom $q(c)$ in time step 1 needs to be underwritten by an inherence determination attributing a particular shade of q-ness to object c.

Satisfying this condition means *imagining* particular attributes assigned to c for each moment of objective time. One set of determinations satisfying this condition is shown in Figure 6.12.

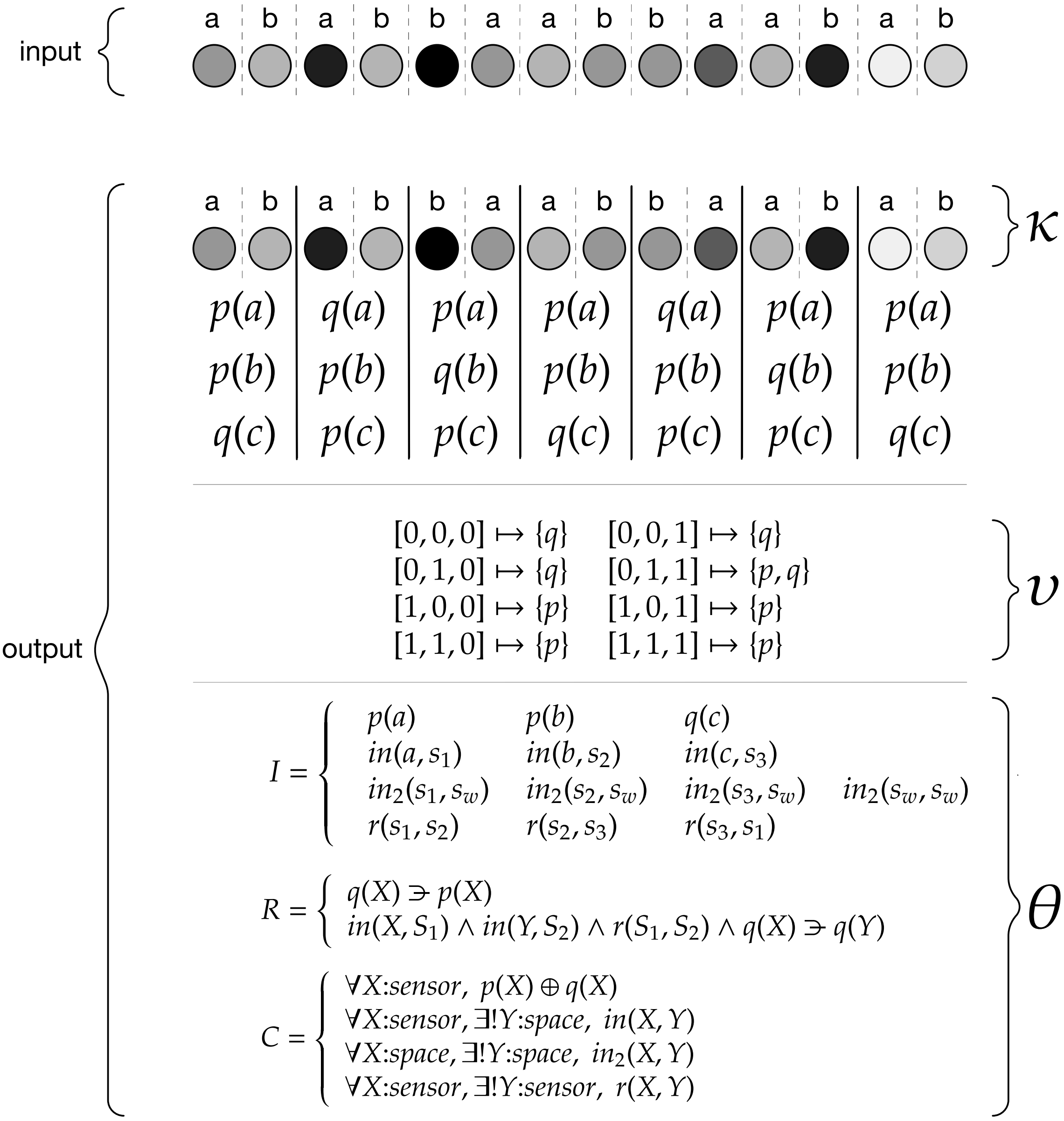
Thus, the unperceived object c is not merely subsumed under a predicate, but is also involved in a determination. *Even though c is an external object with which the agent has no sensory contact, it is cognised as satisfying particular perceptual determinations.* This is, I believe, the truth behind the Kant-inspired claim that "perception is a kind of controlled hallucination" [Cla13].

Note that requirement (7) of Section 6.7 insists that object c must be involved in *some* determination, but does not – of course – insist on any *particular* determination. The productive imagination is free to construct any determination it pleases.

Discussion. Figure 6.13 shows the whole experiment, from the original input to the complete output consisting of a synthesis of intuitions, a collection of subsumptions, and a set of judgements. The Apperception Engine has discerned a discrete intelligible structure behind the continuous noisy input. It started with a fuzzy sensory input, and perceived, amongst all the noise, an underlying system involving two discrete unary predicates, p and q, and devised a simple theory explaining how p and q change over time.

Let us pause to check that the interpretation of Figure 6.13 satisfies the various conditions (Section 6.9) required to achieve synthetic unity:

- The determinations are connected together via the relations of *succ*, *sim*, and *inc* to form a fully connected graph, as required in Section 6.3.
- The containment condition 5(a) of Section 6.5 is satisfied by the initial conditions I of Figure 6.13. Here, s_w is the spatial whole in which all other objects are contained, directly or indirectly.
- The $<$ relation is not needed in this particular example. The empty relation trivially satisfies the condition 5(b) that $<$ is a strict partial order.


input
a b a b b a a b b a a b a b
output
a b a b b a a b b a a b a b
κ
p(a) q(a) p(a) p(a) q(a) p(a) p(a)
p(b) p(b) q(b) p(b) p(b) q(b) p(b)
q(c) p(c) p(c) q(c) p(c) p(c) q(c)
υ
θ

$$\begin{array}{ll} [0,0,0] \mapsto \{q\} & [0,0,1] \mapsto \{q\} \\ [0,1,0] \mapsto \{q\} & [0,1,1] \mapsto \{p,q\} \\ [1,0,0] \mapsto \{p\} & [1,0,1] \mapsto \{p\} \\ [1,1,0] \mapsto \{p\} & [1,1,1] \mapsto \{p\} \end{array}$$

$$I = \left\{ \begin{array}{llll} p(a) & p(b) & q(c) & \\ in(a, s_1) & in(b, s_2) & in(c, s_3) & \\ in_2(s_1, s_w) & in_2(s_2, s_w) & in_2(s_3, s_w) & in_2(s_w, s_w) \\ r(s_1, s_2) & r(s_2, s_3) & r(s_3, s_1) & \end{array} \right.$$

$$R = \left\{ \begin{array}{l} q(X) \ni\!\!\!- p(X) \\ in(X, S_1) \wedge in(Y, S_2) \wedge r(S_1, S_2) \wedge q(X) \ni\!\!\!- q(Y) \end{array} \right.$$

$$C = \left\{ \begin{array}{l} \forall X{:}sensor,\ p(X) \oplus q(X) \\ \forall X{:}sensor, \exists! Y{:}space,\ in(X, Y) \\ \forall X{:}space, \exists! Y{:}space,\ in_2(X, Y) \\ \forall X{:}sensor, \exists! Y{:}sensor,\ r(X, Y) \end{array} \right.$$

Figure 6.13: The result of applying the Apperception Engine to the input of Figure 6.7. The dashed lines divide moments of subjective time, while the solid lines divide moments of objective time. We show the synthesis of intuitions κ, the subsumptions υ, and the theory θ. We also show the ground atoms at each step of objective time, generated by applying the subsumptions υ to the raw input.

- The requirement 6(a) of Section 6.6.1, that every inherence determination is underwritten by a judgement, is satisfied by the theory θ together with the subsumptions v. Consider, for example, the first determination in the given sequence: $det(a, [1, 0, 0])$. Note that $[1, 0, 0] \mapsto p$ according to v, a is an object of type *sensor* and the determination is underwritten by the judgement $\exists X{:}sensor, p(X)$.

- The requirement 6(b) of Section 6.6.2, that every succession is underwritten by a causal judgement, is satisfied by the theory θ together with the subsumptions v. Consider, for example, the succession:

  $$succ(([0, 0, 1], b, 4), ([1, 1, 0], b, 5))$$

  This represents the succession of $det(b, [0, 0, 1])$ by $det(b, [1, 1, 0])$. Note that $[0, 0, 1] \mapsto q$ and $[1, 1, 0] \mapsto p$ according to the subsumptions v, and rules R contain the causal judgement $q(X) \ni\!\!\!- p(X)$.

- The requirement 6(c) of Section 6.6.3 is not used in our implementation of the Apperception Engine.

- The requirement 6(d) of Section 6.6.4, that every incompatibiity is underwritten by a constraint, is satisfied by the constraints C in θ together with the subsumptions v. Consider, for example, the incompatibility:

  $$inc(([1, 0, 0], a, 1), [0, 0, 1], a, 3))$$

  This incompatibility between determinations is underwritten by the constraint $\forall X{:}sensor,\ p(X) \oplus q(X)$, together with the mappings $[1, 0, 0] \mapsto p$ and $[0, 0, 1] \mapsto q$.

- The requirement 7 of Section 6.7 is satisfied by the inherence determinations featuring invented object c as shown in Figure 6.12.

- The requirement 8 of Section 6.8, that every predicate features in some *xor* or uniqueness constraint, is satisfied by the theory θ of Figure 6.13. Here, predicates p and q feature in the constraint $\forall X{:}sensor,\ p(X) \oplus q(X)$, *in* features in the constraint $\forall X{:}sensor, \exists! Y{:}space,\ in(X, Y)$, and so on for the other binary relations.

This, then, is Kant's cognitive architecture in action. It is pleasant to see the Apperception Engine extract a coherent interpretable theory from the indeterminate sensory input it is given.

6.12.4 Perceptual discernment and conceptual discrimination

Compare the interpretation of Figure 6.13 with the alternative degenerate interpretation of Figure 6.14. Both interpretations satisfy the unity conditions of Sections 6.5, 6.6, and 6.8, but they do so in very different ways. While Figure 6.13 discerns a difference between the inputs – dividing them into two classes, p and q – and constructs a theory that explains how p and q properties interact over time,

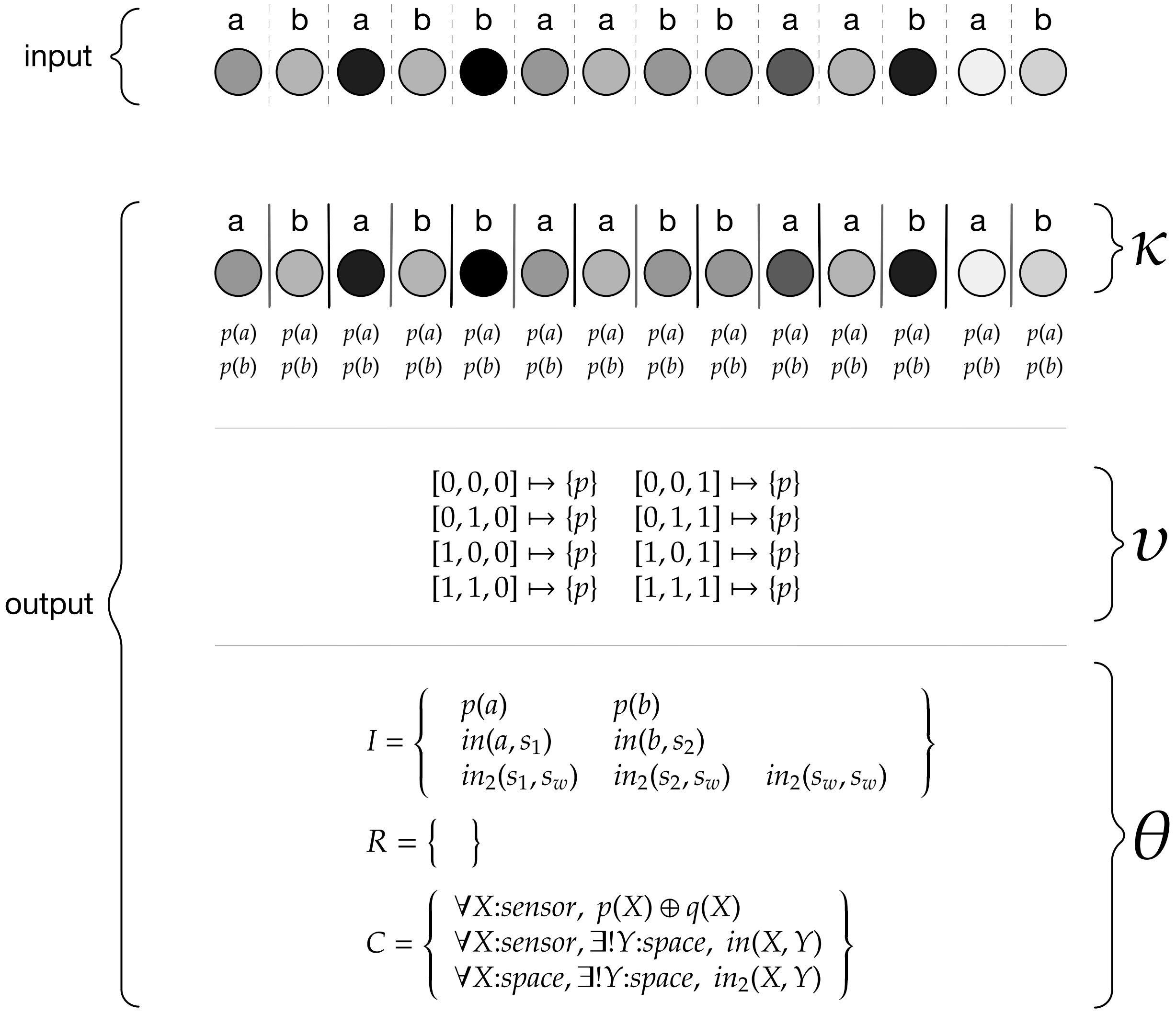

input
a b a b b a a b b a a b a b
output
a b a b b a a b b a a b a b
κ
$p(a)$ $p(a)$ $p(a)$ $p(a)$ $p(a)$ $p(a)$ $p(a)$ $p(a)$ $p(a)$ $p(a)$ $p(a)$ $p(a)$ $p(a)$ $p(a)$
$p(b)$ $p(b)$ $p(b)$ $p(b)$ $p(b)$ $p(b)$ $p(b)$ $p(b)$ $p(b)$ $p(b)$ $p(b)$ $p(b)$ $p(b)$ $p(b)$
$[0,0,0] \mapsto \{p\}$ $[0,0,1] \mapsto \{p\}$
$[0,1,0] \mapsto \{p\}$ $[0,1,1] \mapsto \{p\}$
$[1,0,0] \mapsto \{p\}$ $[1,0,1] \mapsto \{p\}$
$[1,1,0] \mapsto \{p\}$ $[1,1,1] \mapsto \{p\}$
υ
$I = \left\{ \begin{array}{lll} p(a) & p(b) & \\ in(a, s_1) & in(b, s_2) & \\ in_2(s_1, s_w) & in_2(s_2, s_w) & in_2(s_w, s_w) \end{array} \right\}$
$R = \{ \}$
$C = \left\{ \begin{array}{l} \forall X{:}sensor,\ p(X) \oplus q(X) \\ \forall X{:}sensor, \exists! Y{:}space,\ in(X, Y) \\ \forall X{:}space, \exists! Y{:}space,\ in_2(X, Y) \end{array} \right\}$
θ

Figure 6.14: An alternative degenerate interpretation of the input of Figure 6.7. Here, all sensory input is mapped, indiscriminately, to p. Because no discriminations are made, and nothing changes, the induced theory is particularly simple.

Figure 6.14, by contrast, fails to discern any difference between the input vectors. Because Figure 6.14 is coarser and less discriminating, mapping all input vectors to p and none to q, it can make do with a much simpler theory: if everything is always p and never q, we do not need a complex theory to explain how objects transition between p and q.[69]

In Kant's theory of synthetic unity, as we interpret it and implement it, this phenomenon holds across the board. In order to discern a fine-grained discrimination between sensory input, we must provide a theory that underwrites that distinction, a theory that explains how the various properties that we have discriminated actually interact. Fine-grained perceptual discrimination requires an articulated theory (a collection of concepts and judgements) that underpins the distinctions made at the sensible level. Intuitions without concepts are blind.

There is a recurrent myth that humans have fallen from a state of pre-conceptual grace [Jay00]. At some mythic earlier time, humans were not saddled with the conceptual apparatus we now take for granted, and – precisely because they were unburdened by concepts and judgements – were able to perceive the world in all its glory, with a fine-grained vividness we moderns can only dream of. It is as if there is only a finite amount of consciousness to go round; because we modern concept users waste some of that consciousness on the conceptual side of our experience, there is less consciousness remaining to spend on the sensible side. The mythic earlier man, by contrast, is able to spend all his consciousness on the sensible level. Thus for him, in his state of pre-conceptual grace, the colours are brighter.

If Kant is right, this myth gets things exactly the wrong way round. Consciousness is not a zero-sum game between sensibility and understanding, in which one side's gains must be the other side's losses. Rather, perceptual discrimination at the sensible level requires conceptual discrimination from the understanding. *The more intricate the theories we are able to construct, the more vividly we are able to see.*

6.13 Experiment 2: the house

In the *Second Analogy*, Kant describes the following example:

> The apprehension of the manifold of appearance is always successive. The representations of the parts succeed one another. Whether they also succeed in the object is a second point for reflection, which is not contained in the first... Thus, e.g., the apprehension of the manifold in the appearance of a house that stands before me is successive. Now the question is whether the manifold of this house itself is also successive, which certainly no one will concede. [A189/B234ff]

[69] The Apperception Engine considers and evaluates many different theories when presented with the sensory input of Figure 6.14. It prefers the interpretation of Figure 6.13 over the degenerate interpretation of Figure 6.14 precisely because the former discriminates finer. In Chapter 5, I explain how one interpretation is preferred to another, and justify the ordering using simple Bayesian considerations.

Here, Kant asks us to imagine an agent surveying a large house from close range. Its visual field cannot take in the whole house in one glance, so its focus moves from one part of the house to another. Its sequence of visual impressions is successive, but there is a further question whether a pair of (subjectively) successive visual impressions represents the house at a single moment of objective time, or at two successive moments of objective time.

In the B edition of the *Transcendental Deduction*, Kant contrasts the example of the house with another example: an agent watching water slowly freeze:

> Thus if, e.g., I make the empirical intuition of a house into perception through apprehension of its manifold, my ground is the necessary unity of space and of outer sensible intuition in general, and I as it were draw its shape in agreement with this synthetic unity of the manifold in space... If (in another example) I perceive the freezing of water, I apprehend two states (of fluidity and solidity) as ones standing in a relation of time to each other. [B162]

Kant asks us to compare and contrast the two cases of subjective succession. In the first case, the subjective succession represents an *objective simultaneity*: the perceived state of the top of the house is simultaneous with the perceived state of the bottom of the house, even if the subjective impressions are successive. In the second case, by contrast, the subjective succession represents an *objective succession*: the water's transition from liquid to solid is a fact about the world, not just about my mental states. Kant characterises the difference between the two cases in terms of modality: in the case of the house, I could have received the impressions in a different order: I could have seen the bottom of the house before the top. But in the case of the water freezing, I could not have seen the solid state before the liquid state.[70]

We gave the Apperception Engine a simplified version of Kant's example (see Figure 6.15). In our experiment, there is a pixel image that is too large for the agent to survey in one glance. The agent's sensors are only able to take in a small window of the image at any moment, and the agent must move the sensory window around to survey the whole image. Just as in Kant's case, where we cannot take in the whole of the house in one glance, here the agent cannot take in the whole of the pixel image at one glance, and must reconstruct it from the succession of fragments.

Note that in this experiment, the actions are exogenous and do not need to be explained by the Apperception Engine. What does need to be explained is the changing sensor information.

When we give the sequence of Figure 6.15 to the Apperception Engine, it is able to reconstruct the complete picture from the sequence of incomplete partial perceptions. The best theory it finds is

[70] See [Lon98, p.358].

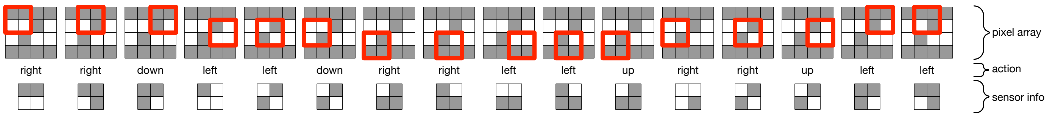


Figure 6.15: For each of the sixteen time-steps, we show the 4×4 pixel grid and the positions of the sensors (as a red square), the action performed, and the values of the four sensors.

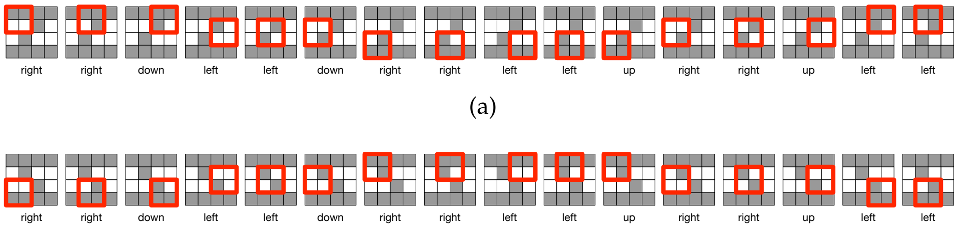
$\theta = (\phi, I, R, C)$ where:

$$
\begin{aligned}
I &= \left\{ \begin{array}{llll}
\mathit{off}(w_{1,2}) & \mathit{off}(w_{1,3}) & \mathit{off}(w_{2,3}) & \mathit{off}(w_{3,2}) \\
\mathit{off}(w_{4,2}) & \mathit{off}(w_{4,3}) & \mathit{on}(w_{1,1}) & \mathit{on}(w_{1,4}) \\
\mathit{on}(w_{2,1}) & \mathit{on}(w_{2,2}) & \mathit{on}(w_{2,4}) & \mathit{on}(w_{3,1}) \\
\mathit{on}(w_{3,3}) & \mathit{on}(w_{3,4}) & \mathit{on}(w_{4,1}) & \mathit{on}(w_{4,4}) \\
\mathit{in}(v_{1,1}, w_{1,4}) & \mathit{in}(v_{1,2}, w_{1,3}) & \mathit{in}(v_{2,1}, w_{2,4}) & \mathit{in}(v_{2,2}, w_{2,3})
\end{array} \right\} \\
R &= \left\{ \begin{array}{l}
\mathit{off}(W) \wedge \mathit{in}(V, W) \wedge \mathit{zero}(N) \rightarrow \mathit{intensity}(V, N) \\
\mathit{on}(W) \wedge \mathit{in}(V, W) \wedge \mathit{one}(N) \rightarrow \mathit{intensity}(V, N) \\
\mathit{right}(M) \wedge r(W, W_2) \wedge \mathit{in}(V, W) \ni \mathit{in}(V, W_2) \\
\mathit{left}(M) \wedge r(W_2, W) \wedge \mathit{in}(V, W) \ni \mathit{in}(V, W_2) \\
\mathit{up}(M) \wedge b(W, W_2) \wedge \mathit{in}(V, W) \ni \mathit{in}(V, W_2) \\
\mathit{down}(M) \wedge b(W_2, W) \wedge \mathit{in}(V, W) \ni \mathit{in}(V, W_2)
\end{array} \right\} \\
C &= \left\{ \begin{array}{l}
\forall C{:}\mathit{cell}, \mathit{on}(X) \oplus \mathit{off}(X) \\
\forall V{:}\mathit{sensor}, \exists! C{:}\mathit{cell}, \mathit{in}(V, C) \\
\forall V{:}\mathit{sensor}, \exists! N{:}\mathit{number}, \mathit{intensity}(C, N)
\end{array} \right\}
\end{aligned}
$$

Here, the initial conditions I specify the on/off values of the sixteen pixels $\{w_{i,j} \mid i \in \{1..4\}, j \in \{1..4\}\}$, and the initial placements of the four sensors $\{v_{i,j} \mid i \in \{1,2\}, j \in \{1,2\}\}$. Note that the on/off values of the pixels do not change over time. The 2D spatial relations between the cells $w_{i,j}$ are represented by the r and b relations. E.g. $r(w_{1,1}, w_{2,1})$, $b(w_{1,1}, w_{1,2})$. The spatial relations using r and b are provided as background knowledge.

The first two rules state that if a sensor is attached to a cell, and that cell is on (respectively off), then the intensity of the sensory is 1 (respectively 0). The other rules describe how the sensors move as the actions are performed. Note that the image reconstructed by the Apperception Engine is a mirror image of the original image used to generate the sensory data. Figure 6.16(b) shows the sequence as reconstructed by the Apperception Engine. In this interpretation, the original image has been flipped vertically, and the actions "up" and "down" have been interpreted as "down" and "up" respectively.

Thus, the Apperception Engine is able to make sense of Kant's famous "house" example [B162]: the engine posits a two-dimensional array of pixels and interprets the sensors as moving across the array. Although the sensory readings are successive, the engine posits an *objective simultaneity* to explain the subjective succession of sensor readings.



(a)

(b)

Figure 6.16: The top row (a) shows the ground truth used to generate the sensory data, while the bottom row (b) shows the reconstruction made by the Apperception Engine. Note that the reconstruction is a mirror image of the ground truth, flipped vertically, in which "up" is interpreted as down, and "down" is interpreted as up.

6.14 Rigidity and spontaneity

There is a popular image of Kant as a rigid rule-bound automaton whose daily routine was so tightly scheduled you could use it to calibrate your clock. According to this popular image, Kant's philosophy (both practical and theoretical) is as rigid and rule-bound as his unusually unremarkable personal life. What is most unfair about this gross mischaracterisation is that it omits the critical fact that, for Kant, the rules I am bound to are rules that *I myself create*.

Spontaneity and self-legislation are at the heart of Kant's philosophy, both practical and theoretical. In his practical philosophy, I am free to construct *any maxims whatsoever* – as long as they satisfy the universalisability conditions of the categorical imperative. In his theoretical philosophy, I am free to construct *any rules whatsoever* – as long as they satisfy the unity conditions. When confronted with a stream of raw sensory input, the Kantian agent constructs a synthesis of apprehension, a set of subsumptions mapping intuitions to concepts, and a set of judgements connecting concepts together. The agent is completely free to construct *any* synthesis of apprehension, *any* set of subsumptions, and *any* set of judgements – so long as the package jointly satisfies the unity conditions (Sections 6.5, 6.6, and 6.8). These conditions of unity are not unnecessary extraneous requirements that Kant insists on for some personal Puritan preference – they are the absolutely minimal conditions necessary for it to be *you* who is doing the constructing. According to Kant, the conditions that need to be satisfied to interpret the sensory input as a coherent representation of a single world are *exactly the same conditions* that need to be satisfied for there to be a *self* who is perceiving that world.[71]

Unlike the popular image, Kant's vision of the mind is one of remarkable freedom. I am continually constructing the program that I then execute. The only constraint on this spontaneous construction is the requirement that there is a single person looking out. In our computer implementation, this spontaneity is manifest in a particular way: when given a sensory sequence, the Apperception Engine constructs an unending sequence of increasingly complex interpretations, each of which

[71] "The *a priori* conditions of a possible experience in general are at the same time conditions of the possibility of the objects of experience." [A111]

satisfies Kant's unity conditions. (See Chapter 3). The engine must decide, somehow, which of these interpretations to choose.[72]

6.15 Rigidity and diachrony

Wittgenstein is sometimes interpreted as denying the possibility of any rule-based account of cognition. Throughout the *Investigations* [Wit09], Wittgenstein draws our attention, again and again, to cases where our rules *give out*:

> I say "There is a chair" What if I go up to it, meaning to fetch it, and it suddenly disappears from sight? - - "So it wasn't a chair, but some kind of illusion". - - But in a few moments we see it again and are able to touch it and so on. - - "So the chair was there after all and its disappearance was some kind of illusion". - - But suppose that after a time it disappears again - or seems to disappear. What are we to say now? *Have you rules ready for such cases* - rules saying whether one may use the word "chair" to include this kind of thing? But do we miss them when we use the word "chair"; and are we to say that we do not really attach any meaning to this word, because *we are not equipped with rules for every possible application of it*? (*Investigations*, §80)

Our rules for the identification of chairs cannot anticipate every eventuality, including their continual appearance and disappearance - but this does not mean we cannot recognise chairs. Or, to take another famous example, we have rules for determining the time in different places on Earth. But now suppose someone says:

> "It was just 5 o'clock in the afternoon on the sun" (*Investigations*, §351)

Again, our rules for determining the time do not cover all applications, and sometimes just *give out*. They do not cover cases where we apply time of day on the sun. Since any set of rules is inevitably limited and partial, we must continually improvise and update.

This point is important and true, but is fully compatible with Kant's vision of the cognitive agent. Such an agent is *continually constructing* a new set of rules that makes best sense of its sensory perturbations. It is not that it constructs a set of rules, once and for all, and then applies them rigidly and unthinkingly forever after. Rather the process of rule construction is a continual effort.

Kant describes an *ongoing process* of constructing and applying rules to make sense of the barrage of sensory stimuli:

[72] Our way of deciding between the various interpretations is described in Section 5.3. This is one place where we attempt to go beyond Kant's explicit pronouncements, since he does not give us guidance here.

> There is no unity of self-consciousness or "transcendental unity of apperception" apart from this effort, or conatus towards judgement, *ceaselessly affirmed and ceaselessly threatened with dissolution* in the "welter of appearances" [Lon98, p.394]

Kant's apperceptual agent is continually constructing rules so as to best make sense of the barrage of sensory stimuli. If he were to cease constructing these rules, he would cease to be a cognitive agent, and would be merely a *machine*.

In *What is Enlightenment?* [Kan84], Kant is emphatic that the cognitive agent must never be satisfied with a statically defined set of rules - but must always be modifying existing rules and constructing new rules. He stresses that adhering to any statically-defined set of rules is a form of *self-enslavement*:

> Precepts and formulas, those mechanical instruments of a rational use, or rather misuse, of his natural endowments, are the ball and chain of an everlasting minority.

Later, he uses the term "machine" to describe a cognitive agent who is no longer open to modifications of his rule-set. He defines **enlightenment** as the *continual willingness to be open to new and improved sets of rules*. He imagines what would happen if we decided to fix on a particular set of rules, and forbid any future modifications or additions to that rule-set. He argues that this would be disastrous for society and also for the self.

In *The Metaphysics of Morals* [Kan97], he stresses that the business of constructing moral rules is an ongoing never-ending task:

> Virtue is always in progress and yet always starts from the beginning. - It is always in progress because, considered objectively, it is an ideal and unattainable, while yet constant approximation to it is a duty. That it always starts from the beginning has a subjective basis in human nature, which is affected by inclinations because of which *virtue can never settle down in peace and quiet with its maxims adopted once and for all* but, if it is not rising, is unavoidable sinking. [MM 6:409, my emphasis]

Just as for moral rules, just so for cognitive rules: Kant's cognitive agent is always constructing new rules to make sense of the pattern.[73]

Some of Wittgenstein's remarks are often interpreted as denying the possibility of *any* sort of rule-based account of cognition:

> We can easily imagine people amusing themselves in a field by playing with a ball so as to start various existing games, but playing many without finishing them and in between throwing the ball aimlessly into the air, chasing one another with the ball and bombarding one another for a joke and so on. And now someone says: The whole time they are playing a ball-game and following definite rules at every throw. (*Investigations* §83).

[73] In this respect, the Apperception Engine is only a partial implementation of Kant's vision, as our system does not support incremental theory revision. See Section 8.6.

Now there is a crucial scope ambiguity here. Is Wittgenstein merely denying that there is a set of rules that captures the ball-play at every moment? Or is he making a stronger claim, claiming that there is *some* moment during the ball-play that cannot be captured by *any set of rules at all*? I believe the weaker claim is more plausible: we make sense of the world by applying rules, but we need to continually modify our rules as we progress through time. Wittgenstein's passage in fact continues:

> And is there not also the case where we play and make up the rules as we go along? And there is even one where we alter them, as we go along.

Here, he does not consider the possibility of there being activity that cannot be explained by rules - rather, he is keen to stress the *diachronic* nature of the rule-construction process: one set of rules at one moment in time, a modified set of rules at a subsequent moment. Thus Wittgenstein's remarks on rules should not be seen as precluding any type of rule-based account of cognition, but rather as emphasising the importance of always being open to revising one's rules in the light of new information. As T. S. Eliot once observed[74]:

> For the pattern is new in every moment
> And every moment is a new and shocking
> Valuation of all we have been

[74] Four Quartets, *East Coker*.

6.16 The table

In *Kant's Theory of Mental Activity*, Robert Wolff makes the following striking confession:

> But when I tried to restate Kant's teaching in my own words, I discovered that I simply could not do so. Indeed, the very wealth of detail which I had gleaned from the secondary literature proved an embarrassment, for out of it emerged no single coherent doctrine. The Analytic, and in particular the Deduction, appeared to me a great tangle of insights and half-completed proofs. Each time I began to unravel it, I found myself enmeshed in still further loops and snarls. Where it began and where it ended I could not tell, nor was I sure that it would unwind into a single connected strand of argument. In puzzling over this failure, it occurred to me that the problem might lie less in the complexity of the text than in the obscurity of certain of its key terms. While expending immense energy comparing proof texts and decomposing compound passages, the commentators had neglected to explain the meanings of the pivotal concepts on which Kant's analysis turned. In particular, I realized that I hadn't any clear idea of what Kant meant by "synthesis". [Wol63, p. vii - viii]

In an effort to provide clarity, we present a table mapping some of Kant's terms to our implementation:

Cognitions	
Intuition	A vector e.g. $[1, 0, 0]$.
Concept	A predicate e.g. p.

Representations	
Determination	A ground term connecting two intuitions together e.g. $det(b, [1, 0, 1])$, $in(a, b)$, or $a < b$.
Subsumption	A ground term subsuming an intuition under a predicate e.g. $s(p, b)$.
Judgement	A rule or constraint e.g. $p(X) \mathrel{\ni\!\!-} q(X)$ or $\forall X\ p(X) \oplus q(X)$.

Faculties	
Sensation	The capacity to receive raw sensory input as a sequence of inherence determinations.
Imagination	A collection of ASP choice rules for (i) connecting determinations via *succ* or *sim*, (ii) connecting intuitions via *det*, and (iii) relating intuitions via *in*.
Power of judgement	A binary neural network (implemented in ASP).
Capacity to judge	Unsupervised rule synthesis (implemented in ASP).

Processes	
The synthesis of apprehension	The construction of a set κ of connected determinations. See Section 6.9.
The synthesis of recognition	The construction of subsumptions v and theory θ satisfying the unity conditions. See Section 6.9.

Pure aspects of intuition	
Form of intuition	Relations between pure intuitions e.g. the r relation of Section 6.12.3.
Formal intuition	Constructed spatial objects e.g. s_1, s_2 of Section 6.12.3.

Pure aspects of concepts	
Schemata	Pure relations connecting intuitions and determinations e.g. *in*, <, *det*, *succ*, *sim*, and *inc*. See Section 6.9.1.
Categories	Unary predicates derived from the pure relations of the schemata. See Section 6.10.

Chapter 7

Related work

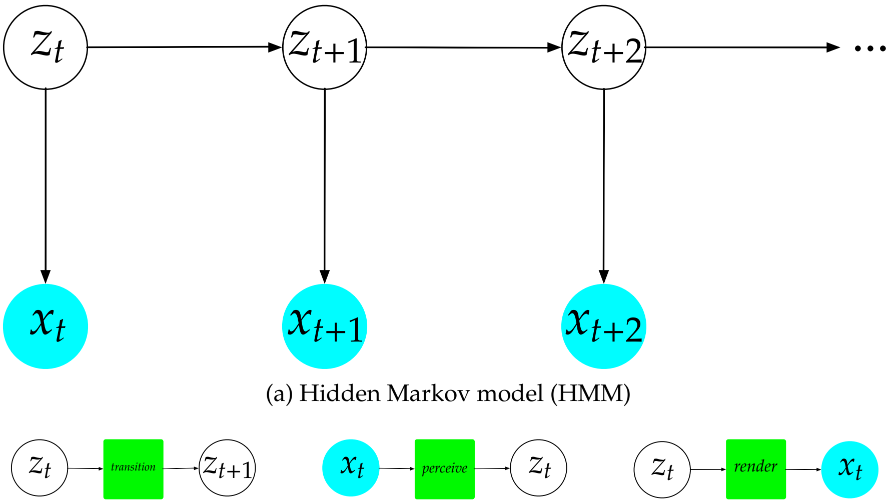
A human being who has built a mental model of the world can use that model for counterfactual reasoning, anticipation, and planning [Heg04, GS14, JL12, Har00, GT17, GAS16]. Similarly, computer agents endowed with mental models are able to achieve impressive performance in a variety of domains. For instance, Lukasz Kaiser et al. [KBM19] 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. [SHS$^+$18] 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 [Ham19]): 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)[1] [BP66, Gha01] as a general framework for describing sequential processes. Diagram 7.1a shows a HMM. 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}.

[1] 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.



(a) Hidden Markov model (HMM)

(b) Transition function

(c) Perceive function

(d) Render function

Figure 7.1: **(a)** a graphical model of a hidden Markov model. Here, each z_t is a state, a complete description of the world at a particular point t in time. The x_t is the observation at time t. While x_t is observed by the agent, z_t is not directly observed and must be inferred. Arrows indicate dependencies between variables. The Markov assumption is that the next state z_{t+1} depends only on state z_t and not on earlier states such as z_{t-1}. **(b)** represents the transition function for a HMM. **(c)** represents the perception function that takes an observation x_t and produces a state z_t. **(d)** represents the inverse rendering function that takes a latent state z_t and produces an observation x_t.

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 [FWS$^+$18, NKFL18, BPL$^+$16, CUTT16, MZW$^+$18, SGHS$^+$18] 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 [LGF16, FL17, BMSF18] 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. But some approaches take latent information seriously [OGL$^+$15, CRWM17, HS18, BWR$^+$18, JLF$^+$18]. 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).

But our use of latent information is rather different from its use in [OGL$^+$15, CRWM17, HS18, BWR$^+$18,

See Jessica Hamrick's paper for an excellent overview [Ham19] of model-based methods in deep learning that is framed in terms of POMDPs. Here, we consider HMMs. Adding actions to our model is not particularly difficult (see Section 5.5.2).

JLF$^+$18]. In their work, the latent information is merely a lower-dimensional representation of the surface information: since a neural network represents a function mapping the given sensor information to a latent representation, the latent representation is nothing more than a summary, a distillation, of the sensory given. But we use latent information rather differently. Our latent information goes *beyond* the given sensory information to include invented objects and properties that are not observed but *constructed* in order to make sense of what is observed.[2] Following John McCarthy [McC06], we assume that making sense of the surface sensory perturbations requires hypothesizing an underlying reality, distinct from the surface features of our sensors, that makes the surface phenomena intelligible.

The second dimension of variation concerns whether the learned model is explicit, symbolic and human-readable, or implicit and inscrutable. In some approaches [OGL$^+$15, CRWM17, HS18, BWR$^+$18], the latent states are represented by vectors and the dynamics of the model by weight tensors. In these cases, it is hard to understand what the system has learned. In other approaches [ZLS$^+$18, XLS$^+$19, AF18, Asa19], 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 [Ray09, IRS14, KAP15, MSPA16, KAP16, MAP18], both the latent state and the state transition function are represented symbolically. Here, the latent state is a set of ground atoms 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. [FL17]), 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 [XLS$^+$19]. Other systems additionally incorporate prior knowledge about the type of rules that are used to define the state transition function. For example, some [MSPA16, KAP16, MAP18] use prior knowledge of the event calculus [KS86]. 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 Kant-inspired unity conditions (Section 3.3).

[2] This is why we use the distinctive "covers" relation between the trace and the given sequence: the covers relation tests that each state of the given sequence is a *subset* of the corresponding state of the trace. This contrasts with other systems (e.g. LFIT [IRS14]) which test if the given state is *identical* to the corresponding state of the trace.

Our approach

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 relations 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 Kant-inspired 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 [MSPA16, KAP16] are supervised, and SPLICE [MAP18] is semi-supervised.

In the rest of this section, we describe particular systems that are related to our approach.

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

Tomer Ullman et al [GUT11, UGT12] 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[3], 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 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.

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

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 [MLT15, CM16, Cro17, LRB14, LRB18a].

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, our system takes as input a *sequence* $(S_1, ..., S_T)$ while their system considers only a single state S. Because they do not model facts changing over time, their system only needs to represent static rules and does not need to also represent causal rules.

Second, 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 third 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*: object connectedness, conceptual unity, static unity, and temporal unity. There is no analog of these Kant-inspired unity conditions in Ullman et al's system.

Fourth, 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.

Fifth, 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.

At the algorithmic level, the systems are very different. While we use a form of meta-interpretive learning (see Section 3.7.2), 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.

Why use MCMC rather than a global method that is guaranteed to find a global minimum? One reason for using MCMC is if we want to construct a distribution over candidate theories, in order to generate predictions from a mixture model. If we have a way of predicting an element x from a theory θ, then we can predict x from data D by:

$$p(x \mid D) = \sum_{\theta} p(\theta \mid D) p(x \mid \theta)$$

But this is not what Ullman et al actually do. Rather, they use MCMC to find a single point estimate, the maximum a posteriori (MAP) theory that best explains the data. Computing a distribution over theories is expensive in time and space. As they acknowledge [UGT12], humans typically only consider a tiny handful of rival theories, and often only just one.

One concern with MCMC approaches to program synthesis is that, typically, making one small change in a program requires many other changes to also be made, in order for that first change to be coherent. Suppose n changes are needed *together* to make an improvement to a candidate rule set R. Since MCMC makes these n changes *individually*, and each of the first $n-1$ changes are on their own insufficient to gain an improvement over the initial R, the chances of MCMC making all n changes and finding the improvement to R is k^{n-1} where k is the mean change of making a suboptimal switch. Since k^{n-1} quickly tends to 0 as n increases, the chances of MCMC finding large programs becomes increasingly small. This is, we believe, the reason why the rule sets found by this approach are small (typically 2 to 4 clauses with at most 2 atoms in the body) in comparison with the programs found by the Apperception Engine (which can contain over 20 clauses with up to 5 atoms in the body).

7.2 "Learning from interpretation transitions"

Inoue, Ribeiro, and Sakama [IRS14] 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. Because it does not represent incompossibliity, there is no frame axiom in LFIT.

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 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.

LFIT was tested on Boolean networks and on Elementary Cellular Automata. It is instructive to compare our system with LFIT on the ECA tasks. There are two key points of difference. First, their system does not generate the smallest set of maximally general rules. The program that LFIT learns for rule 110 contains a redundant rule but LFIT is unable to recognize its redundancy. Second, and more importantly, LFIT 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 unsupervisedly as part of the initial conditions. (See Section 4.2.1). 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.

7.3 "Unsupervised learning by program synthesis"

Kevin Ellis et al [ESLT15] use program synthesis to solve an unsupervised learning problem. Given an unlabeled 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 combined log lengths of the program, the initial conditions, and the data-reconstruction error:

$$-logP_f(f) + \sum_{i=1}^{N} \left(-logP_{x|z}(x_i \mid f(I_i)) - logP_I(I_i) \right)$$

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[4]. At a high level, their algorithm is also similar: they generate a Sketch program [SLTB$^+$06] 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. Another point of similarity between this system and ours is that both systems struggle with large datasets. Because of the way problems are encoded and then compiled into SAT problems, the tasks get prohibitively large as the dataset increases in size.

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$^{\ni}$, 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.

One 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

[4] 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.

is fully determinate, and ground the incompossibility relation that underlies the frame axiom.

7.4 “Beyond imitation: zero-shot task transfer on robots by learning concepts as cognitive programs”

Miguel Lázaro-Gredilla et al [LGLGG18] describe a system for learning procedural programs. Their system is given a set of input/output pairs (visual representations of the start state and target state), and learns a procedure that, when executed, transforms the input scene into the output scene.

At a high level, this system is similar to ours in that it learns an interpretable program from a small number of examples. Like our system, they have a program length prior that prefers small general programs to large special-case programs, so their learned programs tend to generalize well to new situations. Their system shares the same underlying assumption as ours: concepts are programs in a language of thought, learned from sensory input.

However, there are a number of key differences between our approaches.

First, our system operates on time series with a built-in domain-independent understanding of persistence (see the frame axiom in Definition 9). While they operate in a static world (the “tabletop world”) where the only agent that initiates change is the self, our system attempts to make sense of a dynamic world where any object can initiate changes.

Second, they concentrate on supervised program synthesis using input/output pairs, while we focus on *unsupervised* program synthesis.

Third, we use a declarative language for describing concepts, while they use an imperative one. For them, a concept is a procedure to change the world in a certain way (e.g. “stack green objects on the right”). For us, a concept is a predicate that can appear in a declarative sentence.

Fourth, our system includes constraints as well as update rules. These xor and uniqueness constraints are the key distinctive aspect of our architecture: they allow concepts to be unified into groups, they provide determinacy in the states of the trace, and they ground the incompossibility relation that underlies the frame axiom.

Finally, their system has a very distinctive architecture that combines visual attention, imagination, a vision hierarchy, and a dynamics model. We respect their unapologetic use of architectural bias, but note that their architecture is very different from ours.

7.5 “Learning symbolic models of stochastic domains”

Hanna Pasula et al [PZK07] 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 tick*. 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 generalises 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. One limitation of their system is that they assume only one rule can fire in any state. In our system, many rules fire (both static rules and causal rules). In theirs, there is exactly one rule. Because of this assumption, they cannnot 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.

More generally, they assume that a subset of events has been distinguished as exogenous actions and that only actions can create state changes. In our more general system, events bring about other events. In the Apperception Engine, we are not given complete (s, a, s') triples for supervised learning, but sequences of partial information $(S_1, S_2, ...)$.

7.6 "Nonmonotonic abductive inductive learning"

Oliver Ray [Ray09] 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 [KS86] 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 other Kant-inspired unity conditions, such as object connectedness or 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[5]. 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[6].

7.7 The Game Description Language and inductive general game playing

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) [GL05]. 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 `next` for specifying state transitions (equivalent to our causal rules). The *inductive general game playing* (IGGP) task [GB13, CEL19] involves learning the rules of a game from observing traces of play.

[5] It shares the same implementation strategy as ASPAL [CRL12] and ILASP [LRB14]. See Section 3.7.6 for discussion of the grounding problem associated with this family of approaches. The discussion is specifically focused on ILASP, but the same issue affects ASPAL and XHAIL *mutatis mutandem*. This issues does not affect TAL [CRL10a], however, which is closer to our implementation.

[6] See Appendix C of [EG18] for a discussion of the use of mode declarations as a language bias in ILP systems.

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.

7.8 The predictive processing paradigm

The predictive processing (PP) paradigm [Fri05, Fri12, Cla13, Swa16] is an increasingly popular model in computational and cognitive neuroscience. Inspired by Helmholtz (who was in turn inspired by Kant [Swa16]), 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. It is probabilistic in that the predicted sensory readings are represented as probability density functions. It is Bayesian in that the likelihood (represented by the information size of the misclassified predictions) is combined with prior expectations [Fri12]. It is hierarchical in that each layer provides predictions of sensory input for the layer below; there are typically many layers.

In terms of Marr's three levels of analysis, PP combines a computational level of description (what the model is doing) with the algorithmic level of description (how the model is doing what it does) in that PP models typically include a commitment to a neural net implementation of the predictive architecture.

One key difference between our approach and PP is that the Apperception Engine generates a unified 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.

Another key difference is how the two approaches address Hume's problem of induction: the problem, roughly, of justifying causal statements on the basis of scanty and particular instances of associations.

While Hume attempted to solve the problem by asserting that humans do, as a matter of contingent empirical fact, have a tendency to posit causal relations, Kant claimed that agents *must* posit causal relations in order to perceive temporal succession. In Kant's solution, necessity is required at two levels: first, we must (deontic necessity) posit causal laws in order to perceive succession; second, the law itself states what must (alethic necessity) happen when its antecedent is satisfied.

PP uses probabilistic Bayesian inference to conclude that certain sensory percepts are highly probable given other sensory percepts. For Kant, by contrast, causal rules do not apply to *most* instances with a certain probability; rather, they apply to *all* of the instances with the "dignity of necessity". We claim that the Apperception Engine, which posits universally quantified necessary causal rules in order to generate temporal succession, is closer to Kant's understanding of causation than PP.

A third key difference between our approach and PP is that our approach insists on conceptual unity, which can only be achieved by positing xor constraints which combine predicates into clusters. There is no equivalent of conceptual unity via constraints in PP approaches so far.

7.9 Other related work

We briefly outline three other related topics. One related area of research is learning action theories [Moy02, Ote05, IBN05, CSIR11, RGRS11]. 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 related area is relational reinforcement learning [DDRD01, DR08, BBDS^{+}08]. 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.

Another related area is learning game rules from player traces [Goo96, Mor96, CW03, Kai12, LRB14, GSBS15]. 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.

Chapter 8

Discussion

In conclusion, we outline the key strengths and limitations of our system.

8.1 Appealing features of the Apperception Engine

As a system for unsupervised induction of interpretable general laws from raw unprocessed data, the Apperception Engine has the following appealing features: it is (i) interpretable, (ii) accurate, and (iii) data-efficient. We shall consider each in turn.

8.1.1 Interpretability

The Apperception Engine produces a theory, an explicit program in Datalog$^{\ni}$, to make sense of its given input. This theory is interpretable at three levels.

First, we can understand the *general ontology* that the system is using: we know what persistent objects the system has posited, the types of those persistent objects, and the sorts of predicates that can apply to those objects. In *Sokoban*, for example, in Section 5.5.2, we understand there are three objects: o_1 of type t_1, and o_2 and o_3 of type t_2. The synthesised constraints act like type judgements to restrict the set of models. For example, the constraint $\forall Y{:}t_2, \exists! C{:}cell, in_2(Y, C)$ states that every block is placed in exactly one cell.

Second, we can understand how the system interprets *particular moments in time*. In *Sokoban*, in Figure 5.9, at time step t_1, for example, we understand that the system thinks o_1 is below o_2, that o_3 is in the top right corner, and that o_2 is being pushed up by o_1. As well as being able to interpret the fluent properties and relations that the system thinks hold at a particular time, we can also interpret how the system connects the raw perceptual input to the persistent objects at each moment. In Figure 5.9, for example, we can see how subregions of the 20×20 pixel array correspond to particular persistent objects.

Third, we can understand the *general dynamics* that the system believes hold universally (for all objects, and for all times). The engine is designed to satisfy the Kant-inspired constraint that whenever something changes, there must be a general universal law that explains that change: there is no change that is not intelligible. When we inspect the synthesised laws, we understand how the system believes properties change over time.

For example, the fifth rule learned for *Sokoban* is:

$$in_1(X, C_1) \wedge in_2(Y, C_2) \wedge below(C_1, C_2) \wedge action(south) \rightarrow p_3(Y)$$

Now p_3 is an invented predicate; its meaning is not apparent just from this single rule. But if we look at the other rule in which p_3 figures:

$$p_3(Y) \wedge in_2(Y, C_1) \wedge below(C_1, C_2) \ni in_2(Y, C_2)$$

we can see that p_3 is being used to represent that a block is being pushed south. Now we can understand the rule whose head is p_3 as: when the *south* action is performed, and the man X is above a block Y, then Y is pushed downwards.

It is important to note that the interpretability of individual clauses as general laws is a consequence of our using a declarative logic-based language as the target of our program synthesis, rather than, say, a procedural language. A procedural program is composed of procedures, each of which contains a recipe telling us how to *do* something. A logic program is a set of clauses, each of which tell us how *the world works*. Each clause, in other words, can be interpreted as a *judgement*. A procedure tells the computer how to do something, while a declarative clause has a truth condition: it makes a claim (that may be true *or false*) about how the world is. Systems that generate logic programs have a special *sui generis* kind of interpretability that is not shared by systems that generate procedures. Even though a procedural program may be just as human-readable as a logic program (or more so, for people who are much more familiar with imperative programming than declarative programming), it is not decomposable into constituents that can be interpreted as judgements, stating how things are.

We have attempted to show, in Sections 5.5.1, 5.5.2, and 5.5.3, how the Apperception Engine understands the sensory input it is given. It must be acknowledged, however, that the theories produced by the Apperception Engine are only readable by a small subset of humans—those comfortable reading logic programs. Whenever we say that a system is "interpretable", we mean interpretable for a particular audience in a particular context. So, although we have provided evidence that the system is interpretable for some people, there is much work to do to provide interpretations accessible to a wider audience.

8.1.2 Accuracy

The Apperception Engine attempts to discern the causal structure that underlies the raw sensory input. In our experiments, we found the induced theory to be very accurate as a predictive model, no matter how many time steps into the future we predict. For example, in *Seek Whence* (Section 5.5.1), the theory induced in Figure 5.3a allows us to predict all future time steps of the series, and the accuracy of the predictions does not decay with time.

In *Sokoban* (Section 5.5.2), the learned dynamics are not just 100% correct on all test trajectories, but they are *provably* 100% correct. These laws apply to all *Sokoban* worlds, no matter how large, and no matter how many objects. Our system is, to the best of our knowledge, the first that is able to go from raw video of non-trivial games to an explicit first-order causal model that is provably correct.

In the noisy sequences experiments (Section 5.5.3), the induced theory is an accurate predictive model. In Figure 5.17, for example, the induced theory allows us to predict all future time steps of the series, and does not degenerate as we go further into the future.

8.1.3 Data efficiency

Neural nets are able to solve some sequence induction IQ tasks from raw input when trained on a sufficiently large number of training examples [BHS+18]. Neural nets are also able to learn the dynamics of *Sokoban* from raw input, when trained on a sufficiently large number of episodes [RWR+17].

But these models are notoriously data-hungry. In comparison with humans, who are often capable of learning concepts from a handful of data [LST15], artificial neural networks need thousands or millions of examples to reach human performance.

The Apperception Engine, by contrast, is much more data-efficient. While a neural network needs millions of trajectories to achieve reasonable accuracy on *Sokoban* [BWR+18], our system is able to learn a perfectly accurate model from a *single* trajectory. While a neural network needs hundreds of thousands of examples [BHS+18] to achieve human-level performance on Raven's Progressive Matrices [CJS90], our system is able to discern a pattern from a *single* sequence. The reason for our system's unusual data efficiency is the strong (but domain-independent) inductive bias that we inject via the $\text{Datalog}^{\ni}$ language (Definition 2) and the unity constraints (Definition 11).

A system that can learn an accurate dynamics model from a handful of examples could be useful for model-based reinforcement learning. Standard model-free algorithms require millions of episodes before they can reach human performance on a range of tasks [BWR+18]. Algorithms that learn an implicit model are able to solve the same tasks in thousands of episodes [KBM19]. But a system that learns an accurate dynamics model from a handful of examples should be able to apply that model to *plan*, anticipating problems in imagination rather than experiencing them in reality [HS18], thus opening the door to extremely sample efficient model-based reinforcement learning. We anticipate a system that can learn the dynamics of an Atari game from a handful of trajectories[1], and then apply that model to plan, thus playing at reasonable human level *on its very first attempt*.

8.1.4 Summary

We can see, then, that a number of problems that have dogged neural networks since their very conception are solved or finessed when we move to a hybrid neuro-symbolic architecture that combines low-level perception with high-level apperception.

The Apperception Engine inherits the traditional advantages of Inductive Logic Programming methods, in being data-efficient, generalising well, and supporting continual learning. But our system has two key features which distinguish it from standard ILP. First, it does not require human-labelled training data, but works with *unsupervised* sequences of sensory input. Second, it does not expect its input in pre-processed symbolic form; rather, it is able to work with raw unprocessed sensory input (e.g. noisy pixels).

8.2 What makes it work

What is it about the architecture that enables the Apperception Engine to satisfy the desiderata listed above? We identify three features which are critical to its success: (i) the declarative logic programming language that is used as the target language for program synthesis, (ii) the strong inductive bias injected into the system, and (iii) the hybrid architecture that combines binary neural networks with symbolic program synthesis.

8.2.1 The declarative logic programming language

When designing a program synthesis system, a critical decision is: what form should the target language take? Our target language, Datalog$^{\ni}$, has three features which we regard as critical to the system's success.

First, the language is very *concise*. A single Datalog clause is a powerful computational construct: each quantified variable in the clause represents a single for-loop in a procedural language. In an evaluation of program-verification tasks, a Datalog program was found to be up to two orders of magnitude shorter than its Java counterpart [WACL05]. Concision is very important in program synthesis: the search space of programs considered is b^n where b is the mean branching factor and n is the program length. Thus, a concise language (in which n is shorter) is much more tractable for search [Cro19]. The conciseness of Datalog$^{\ni}$ is a key feature allowing us to synthesise theories for

[1] Atari games have become a standard benchmark for reinforcement learning agents. The Arcade Learning Environment [BNVB13] is a framework for evaluating reinforcement learning agents. State of the art reinforcement learning agents achieve human level performance, but require millions of episodes of training [MKS+13].

non-trivial domains (see the experiments in Sections 5.5.1, 5.5.2, and 5.5.3). If we had used a less concise target language, we would not have been able to solve these problems.

The second critical feature of Datalog$^{\ni}$ is that the language is *declarative*. The constituents of Datalog$^{\ni}$ programs are individual clauses. Each clause can be interpreted separately as a *judgement* that makes a distinctive claim about the world. Of course, the meaning of one clause depends on the set of clauses in which it is embedded, but (given its embedding context) a single clause still has a unique meaning as a particular claim about the world.

Contrast this *declarative decomposability* of Datalog with the procedural case: in an imperative program, the constituents are procedures, not clauses, and a procedure cannot be interpreted as a judgement with a truth-condition—a procedure is just a recipe for getting something done. The declarative decomposability of Datalog$^{\ni}$ was critical to the interpretations of Sections 5.5.1, 5.5.2, and 5.5.3.

Michalski [Mic83] was well aware of the importance of declarative decomposability:

> The results of computer induction should be symbolic descriptions of given entities, semantically and structurally similar to those a human expert might produce observing the same entities. *Components of these descriptions should be comprehensible as single 'chunks' of information*, directly interpretable in natural language, and should relate quantitative and qualitative concepts in an integrated fashion

The third critical feature of Datalog$^{\ni}$ is its built-in treatment of change and persistence. Instead of creating rules specifying all the facts that are true at a time-step, the rules only need to specify the facts that change from the previous time-step. This allows the theory to be much shorter and simpler, thus giving the program synthesis system a much better chance of finding it in a reasonable time.

8.2.2 Our inductive bias

In each of our experiments, the Apperception Engine is shown to be significantly more data-efficient than the neural network baselines. This data efficiency is only possible because of the significant inductive bias that has been injected into the system. This inductive bias involves three main aspects.

First, there is inductive bias in the *form of clauses* that are allowed in the Datalog$^{\ni}$ language. The only rules that the system is allowed to produce are general rules that quantify over all objects and all times. The system is simply incapable of formulating a rule that applies only to a particular individual, or only to a particular time. In other words, the system is *doomed to generalise*. This inductive bias comes from Kant. He argued that all judgements are universal (apply to all objects).[2] In Kant's cognitive architecture, there is no such thing as a specific judgement. Our system respects this Kantian restriction.[3] Although our system can only construct universally quantified rules, it is

[2] This follows directly from two central claims, that judgements are rules [*Prolegomena* 4:305], and that rules, in turn, are "the representation of a universal condition" [A113].

[3] LFIT has a similar inductive bias [IRS14].

capable of constructing complex theories that treat different cases differently. But the simplicity prior of Definition 17 means we prefer theories with a shorter description length, all other things being equal.

The second form of inductive bias is the introduction of *persistent objects*.[4] The system is forced to reinterpret the ephemeral play of transitory sense data as a re-presentation of a set of persistent objects, with properties that change over time. Again, this inductive bias is inspired by Kant.[5]

The third form of inductive bias is the *unity conditions on an acceptable theory* (Definition 11). These include spatial unity, conceptual unity, static unity, and temporal unity. These constraints, again, are inspired by Kant's discussion in the *Critique of Pure Reason*.[6]

The standard objection to inductive bias, of course, is that although it helps the system learn efficiently in certain domains, the same bias also prevents the system learning effectively in others. According to this objection, inductive bias must be *domain-specific* bias that can only help performance in some domains while hindering performance in others.[7]

We do not accept this argument. The inductive bias we inject is intended to be maximally general bias that applies to all domains that we can understand. The general assumptions we make—that the world is composed of persistent objects, that changes to objects must be covered by general explanatory rules, and so on—are not domain-specific insights but rather general insights about any situation that we are capable of making sense of.

8.2.3 Our hybrid neuro-symbolic architecture

Our hybrid neuro-symbolic architecture allows both neural networks and symbolic program synthesis methods to play to their respective strengths. It has often been noted that artificial neural networks and inductive logic programming have complementary strengths and weaknesses [EG18]: neural networks are robust to noisy and ambiguous data, but are data inefficient and inscrutable. Inductive logic programming approaches to machine learning, by contrast, are data-efficient and provide interpretable models, but do not easily handle noisy data[8], let alone ambiguous data. Our hybrid architecture attempts to combine the best of both worlds, using a neural network to map noisy ambiguous raw input to discrete concepts, and using program synthesis to generate interpretable models from handfuls of data.

The overall architecture, because it represents both the binary neural network and the unsupervised program synthesis system as a single ASP program, allows information to flow both ways: both

[4] The introduction of persistent objects is inevitable in domains like *Sokoban*. But it is notable that, even in domains like *Seek Whence* that do not feature persistent objects at the surface, it is the ability to posit latent persistent objects, with properties that change over time, that is needed to make sense of the sequences.

[5] See [A182/B224] ff.

[6] See Sections 3.3, 6.5, 6.6, and 6.8.

[7] For thoughtful discussions of the "no free lunch theorem" see [LH13] and [ELH14].

[8] There have been some notable recent attempts to address this [MDS$^+$18].

bottom-up and top-down. Information flows bottom-up because the ground atoms generated by the neural network are used by the program synthesis system. Information flows top-down because the high level unity conditions are the only constraints on the whole system. The system is free to choose any neural network weights whatsoever, as long as the whole system (of which the neural net is but a small part) satisfies the unity conditions. In other words, contingent information flows bottom-up while necessary constraints flow top-down. As Kant says: "through it [the constraint of unity] the understanding determines the sensibility [B160-1n]".

8.3 Concepts

What does it mean to understand a concept? When we claim that a particular agent understands a particular concept, what exactly are we attributing to it?

In Robert Brandom's monumental *Making It Explicit* [Bra94], he provides an **inferentialist**[9] interpretation of concept understanding, in which an agent understands a concept when both the following conditions are satisfied:

1. it knows when to apply the concept; in other words, it knows the **circumstances of application**
2. it knows the inferential commitments of applying the concept; in other words, it knows the **consequences of application**

For example, an agent understands the concept "red" if:

1. it is able, when confronted with objects that are red, to apply the concept "red" to them
2. it understands the inferential consequences of saying that something is red: it knows that no (monochromatic) red object is also blue, that red objects are coloured, that crimson objects are red, and so on.

Both of these capacities are required. Neither on its own is sufficient for concept understanding.

Consider, for example, a parrot that has been trained to utter "red" when it sees something that looks red. The parrot knows when to apply the concept, thus satisfying the first of the two conditions for concept understanding. But it does not know the consequences of applying the concept: it does not know that "red" and "blue" are incompatible, that red things are coloured, and so on.

Or consider, for example, Frank Jackson's famous thought experiment:

[9] Inferentialism comes to us from Wilfrid Sellars who in turn was attempting to rearticulate Kant's vision of concept understanding.

> Mary is a brilliant scientist who is, for whatever reason, forced to investigate the world from a black and white room via a black and white television monitor. She specializes in the neurophysiology of vision and acquires, let us suppose, all the physical information there is to obtain about what goes on when we see ripe tomatoes, or the sky, and use terms like ″red″, ″blue″, and so on.

Now Mary knows the inferential consequences of "red". In fact, as a leading neurophysiologist, she understands the inferential consequences of colour concepts better than anyone. But, as she has spent all her life in a black and white room, she does not yet know when to apply the concept "red". If she opens the door and is confronted with a red colour patch, she will not immediately know what colour it is.

We can use this two-aspect inferentialist interpretation of concept understanding to diagnose the limitations of both connectionism and symbolic AI. The trouble with connectionism, according to the inferentialist, is that it focuses only on the circumstances of application, while ignoring the equally important consequences of application. A neural network can be trained to emit "dog" when presented with an image of a dog, but it does not know that all dogs are mammals, that no dog is also a cat, or that corgis are a type of dog. The trouble with traditional symbolic AI, according to the inferentialist, is that it focuses only on the consequences of application—the inferential relations between concepts—while ignoring the equally crucial circumstances of application. This criticism applies to good old-fashioned AI (GOFAI) as well as more modern forms of symbolic AI such as inductive logic programming. In traditional GOFAI, a human hand-engineers the logical rules describing the inferential connections between concepts, while in inductive logic programming, the system constructs the rules itself. But in both cases, the symbolic system does not have a way of mapping raw perceptual input onto concepts. If we want to build a concept understanding system, then, we will need the system to understand *both* the circumstances of application and the consequences of application. The Apperception Engine, when connected to a neural network in the manner described above, is an attempt to realise both aspects of the inferentialist's interpretation of concept understanding: the binary neural network knows when to apply a concept (by mapping raw perceptual input to predicates), while the Apperception Engine generates the inferential connections (the constraints and inference rules) that determine the consequences of application.[10]

8.4 Limitations

I shall describe two groups of limitations: expressive limitations of the sort of theory that the Apperception Engine can synthesise, and scaling limitations of the size and complexity of theories that can be found.

[10] One key difference between our approach and Brandom's is that he believes the inferential connections between concepts are realised by implicit proprieties of practice, rather than by explicit rules. He calls the reliance on explicit rules *regulism*, and appeals to Sellars and Wittgenstein in criticising it.

8.4.1 Expressive limitations

At the moment, all rules are strict and exceptionless. There is no room in the current representation for *defeasible* causal rules (where normally, all other things being equal, *a* causes *b*). Usually, defeasible rules are expressed using negation as failure.The reason why defeasible rules cannot be expressed is that there is currently no support in Datalog$^{\ni}$ for negation as failure.

There is also no room in the current representation for *non-deterministic* causal rules (where *a* causes either *b* or *c*). The reason why non-deterministic rules cannot be expressed is that there is currently no support in Datalog$^{\ni}$ for disjunctions in the heads of strict or causal rules [LMR92].

A fundamental limitation of Datalog$^{\ni}$ is that it requires that the underlying dynamics can be expressed as rules that operate on *discrete* concepts. While the system is capable of handling raw, noisy, continuous sensory input, it assumes that the *underlying dynamics* of the system can be represented by rules that operate on discrete concepts. There are many domains where the underlying dynamics are discrete while the surface output is noisy and continuous: Raven's progressive matrices, puzzle games, and Atari video games, for example. But our system will struggle in domains where the underlying dynamics are best modelled using continuous values, such as models of fluid dynamics. Here, the best our system could do is find a discrete model that crudely approximates the true continuous dynamics. Extending Datalog$^{\ni}$ to represent continuous change would be a substantial and ambitious project.[11] A related limitation is that time is represented in our system only as a sequence of discrete time steps. There is no room in our formalism for continuous time.[12]

8.4.2 Scaling limitations

In our approach, making sense of sensory input means finding a theory that explains that input. Finding a theory means searching through the space of logic programs. This is a huge and daunting task.

The enormous size of the search space (see Section 3.7.4) means that our system is currently restricted to small problems. Because of the complexity of the search space, we must be careful to choose domains that do not require too many objects, predicates, or variables. For example, the Apperception Engine takes 5 GB of RAM and 48 hours on a standard 4-core Unix desktop to make sense of a single *Sokoban* trajectory consisting of 17 pixel arrays of size 20×20. This is, undeniably, a computationally expensive process. Although the Apperception Engine is able to synthesize significantly larger programs than other program induction systems[13], we would like to be able to solve much larger problems than we currently can. For example, we would like to scale our approach up so that we can learn the dynamics

[11] For work in this more ambitious direction, see [VLH08, AVL11, AvL^{+}17].

[12] For a much richer analysis of time that is closer to Kant's texts, see [Pin17, PVL18].

[13] See Section 3.7.6 for a comparison with ILASP. Our system significantly outperforms ILASP on apperception tasks, as shown in Table 3.4 and Figure 3.4. Recently, a successor of ILASP called FastLAS has been proposed [LRB^{+}20]. We plan, in future work, to evaluate FastLAS on the various apperception tasks.

of Atari games from raw pixels. But this will prove to be challenging, as games such as Pacman are substantially harder than our *Sokoban* test-case in every dimension: it requires us to increase the number of pixels, the number of time-steps, the number of trajectories, the number of objects, and the complexity of the dynamics.

To tame the search space, we provide type signatures for many of the examples described above. Although the Apperception Engine is capable in principle of working without any provided type signature, by enumerating signatures of increasing complexity (see Section 3.7.1), in practice for many of the harder examples, we provide a type signature that has been designed to be sufficiently expressive for the task at hand.

The dominant reason for our system's scaling difficulties is that it uses a maximising SAT solver to search through the space of logic programs. Finding an optimal solution to an ASP program with weak constraints is in Σ_2^P; but this complexity is a function of the number of ground atoms, and the number of ground atoms of our ASP program is exponential in the length of the Datalog$^{\ni}$ programs we are synthesising (see Section 3.7.4).

8.5 Basic assumptions

The Apperception Engine in its current form, and its limitations as described above, are a result of some fundamental decisions that were made early on in the project, answers to some basic questions about how to interpret and implement Kant:

1. When Kant says that every succession of determinations must be underwritten by a causal rule, does he mean that (i) there must be a causal rule that the agent believes? Or, much weaker, (ii) the agent must merely believe there *is* a causal rule?

2. When Kant says that judgements are rules, does he mean (i) explicit rules formed from discrete symbols? Or could he mean that some judgements are just (ii) implicit rules?

3. How expressive are Kant's judgements in the *Table of Judgements*? Does he just allow (i) simple definite clauses? Or does he also allow (ii) geometric rules (with disjunctions or existentials in the head)?

4. Given that the understanding involves two separable capacities – the capacity to subsume intuitions under concepts and the capacity to combine concepts into rules – how should these two capacities be implemented? Should there be (i) one system that performs both, or (ii) two separate systems, with one passing its output to the other?

5. Assuming in (4) a single system that jointly combines intuitions and forms judgements, should that single system be (i) symbolic (e.g. SAT-based) or (ii) sub-symbolic (e.g. neural)?

The design of the Apperception Engine was based on choosing option (i) at each of the five decision points. I shall attempt to justify each decision in turn.

8.5.1 Succession and causal rules

In the *Second Analogy*, Kant writes:

> If, therefore, we experience that something happens, then we always presuppose that something else precedes it, which it follows in accordance with a rule. [A195/B240]

Now this claim has a crucial scope ambiguity: does it mean that (i) whenever there is a succession there is a rule which the agent believes that underwrites the succession? Or does it mean that (ii) whenever there is a succession the agent believes that there is some rule that underwrites the successsion, even if the agent does not know what the particular rule is?

Some commentators have assumed the second, weaker interpretation. For example, Longuenesse believes that I do not have to have already formed a causal judgement to perceive a succession – I just need to acknowledge that I should form a causal judgement. For Longuenesse, perceiving a succession means *being committed to look for a causal rule* – it does not mean that I need to have *already found one*:

> The statement that ”everything that happens presupposes something else upon which it follows according to a rule” does not mean that we cognize this rule, but that we are so constituted as to search for it, for its presupposition alone allows us to recognize a permanent to which we attribute changing properties. [Lon98, p.366]

Others, including Michael Friedman [Fri92] take the first, stronger interpretation.

I do not have the space or time to enter into the exegetical fray, but would like to make one observation. If we take the first, stronger interpretation, then any implementation of Kant's theory will be a system that can be used to predict future states, retrodict past states, and impute missing data (see e.g., Example 6). This ability to fill in the blanks in the sensory stream is only available because the agent *actually constructs rules* to explain the succession of appearances. If we had implemented the second, weaker interpretation, then the agent would merely believe that there was *some* rule – it would not have been forced to find the rule, it would have been content to know that the rule existed somewhere. Such an agent would not be able to anticipate the future or reconstruct the past.

8.5.2 Explicit or implicit rules

When Kant says that judgements are rules, does he mean that judgements are (i) explicit rules formed from discrete symbols? Or could he mean that some judgements are just (ii) implicit rules (e.g., a procedure that is implicit in the weights of a neural network)?

The first interpretation, assuming judgements are explicit rules using discrete symbols in the language of thought[14], is a form of what Brandom calls *regulism* [Bra94, p.18]. The second interpretation allows for rules that are universal (they apply to all objects of a certain type), necessary (they apply in all situations), but *implicit*: the rule may not be expressible in a concise sentence in a natural or formal language. For a concrete example of the second interpretation, consider the *Neural Logic Machine* [DML$^+$19]. This is a neural network that simulates forward chaining of definite clauses but without representing the clauses explicitly. The "rules" of the Neural Logic Machine are implicit in the weights (a large tensor of floating point values) of the neural network and cannot be transformed into concise human-readable rules. Nevertheless, the rules are universal and necessary, applying to all objects in all situations.

Most commentators believe that Kant's rules are explicit rules composed of discrete symbols.[15] I do not want to contribute to the exegetical debate, but rather want to provide a practical reason for preferring the first interpretation in terms of explicit rules. Part of the attraction of the Apperception Engine as described above is that the theories found by the engine can be read, understood, and verified. In Section 5.5, for example, the theory learned from the Sokoban trace is not just correct, but *provably* correct. If we need to understand what the machine is thinking, or need to verify that what it is thinking is correct, then we must prefer explicit rules.

Another, perhaps more fundamental, reason for preferring explicit rules is that they enable us to test whether Kant's unity conditions (see Section 6.9) have been satisfied. In order to test whether every succession is underwritten by a causal judgement (Section 6.6.2), for example, we need to be able to inspect the rules produced. It is unclear how a system that operates with merely implicit rules can detect whether or not Kant's unity conditions have actually been satisfied.

8.5.3 The expressive power of Kant's logic

Commentators disagree about the expressive power of Kant's judgements. Some think Kant's logic is restricted to Aristotelian syllogisms over judgements containing only unary predicates. If this were so, Kant's logic would indeed be "terrifyingly narrowminded and mathematically trivial"[16]. Similarly, many commentators (for example, MacFarlane [42], p.26; also [55]) assume or claim that Kant's logic is highly restrictive in that it does not support nested quantifiers. Others[17] argue that Kant must have a more expressive logic in mind, a logic that includes at least nested quantifiers of the form $\forall\exists$.

[14] In this thesis, I follow Jerry Fodor in assuming that our beliefs are expressed in a language of thought [Fod75] which is symbolic and compositional. Moreover, I assume that the language of thought is something like Datalog$^{\ni}$, but somewhat more expressive [Pia11].

[15] But there is a note, inserted in Kant's copy of the first edition of the first Critique [A74/B99], which suggests that judgements need not be explicit: "Judgments and propositions are different. That the latter are *verbis expressa* [explicit words], since they are assertoric".

[16] [Haz99], quoted in [AVL11].

[17] See in particular [AVL11, AvL$^+$17], and also [ESS19].

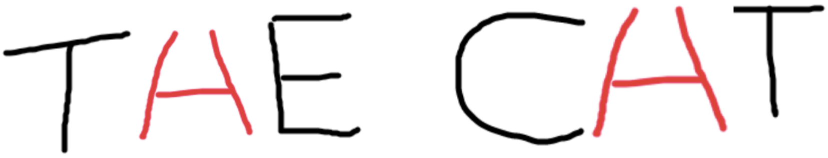


Figure 8.1: Top-down influence from the symbolic to the sub-symbolic. Here the ambiguous image (in red) is disambiguated at the sub-symbolic level using knowledge (of typical English spellings) at the symbolic level.

There is, of course, a tradeoff between the expressiveness of the logic and the tractability of learning theories in that logic: the more complex the judgement forms allowed, the harder it is to learn. Geometric logic, for example, is highly expressive[18] but it is also undecidable [Bez05]. Datalog, by contrast, is decidable, and has polynomial time data complexity [DEGV01].

Because of this tradeoff, in this work we opted for a simpler logic (i.e. Datalog$^{\ni}$ rather than geometric logic) in order to make it tractable to synthesise theories in that logic. One of the central pillars of our interpretation is that Kant's fundamental notion of spontaneity is best understood as *unsupervised program synthesis*. To test out this claim, it was necessary to build a system that is capable of generating theories to explain a diverse range of examples. Thus, in this thesis, we used an extension of Datalog to define a simple range of judgements. We do not claim that logic adequately represents the range of judgements expressible in Kant's *Table of Judgements*: after all Datalog$^{\ni}$ contains no negation symbol, no existential quantifier, and no modal operators. In future work we plan to extend this language with stratified negation as failure, disjunction in the head, and existential quantifiers, to increase its expressive power.

8.5.4 One system or two?

The understanding involves two distinguishable capacities: the capacity to subsume intuitions under concepts (the power of judgement), and the capacity to combine concepts into rules (the capacity to judge). These two capacities take different sorts of input: the power of judgement takes raw intuitions and maps them to discrete concepts, while the capacity to judge operates on discrete concepts. This difference could suggest that we need a hybrid approach involving two distinct systems for the two capacities: one system (perhaps a neural network) for mapping intuitions to concepts and another (perhaps a symbolic program synthesis system) for combining concepts into rules. According to this suggestion, the output of the first system is fed as input to the second system.

A concern with this hybrid approach is that it is very unclear how to support top-down information flow from the conceptual to the pre-conceptual. There is much evidence that expectations from

[18] More generally, [DN15] shows that, for each set Σ of first-order sentences, there is a set of sentences of geometric logic that is a conservative extension of Σ.

the conceptual symbolic realm can inform decisions at the pre-conceptual sub-symbolic realm. See, for example, Figure 8.1.[19] Here, part of the image is highly ambiguous: the 'H' of "THE" and the 'A' of "CAT" use the same ambiguous image, but we are able to effortlessly disambiguate (at the sub-symbolic level) by using our knowledge of typical English spelling at the symbolic level.

Thus, it is essential that the high-level constraints – the conditions of unity (see Section 3.3) – are allowed to inform the low-level sub-symbolic processing. This consideration precludes a two-tier architecture where a neural network transforms intuitions into concepts, and a symbolic system searches for unified interpretations. In such an architecture, it is not possible for the low-level neural network to receive the information it needs from the high-level system. The only information that the neural network will receive in a two-tier approach is a *single bit*: whether or not the high-level symbolic system was able to find a unified interpretation. It will not know *why* it was unable, or *which constraints* it was unable to satisfy. This is insufficient information.

Because of this concern, we opted for a different architecture, in which a *single system* jointly performed both tasks: both mapping intuitions to concepts and combining concepts into rules.[20]

8.5.5 SAT or gradient descent?

Assuming we have opted for a single system rather than a hybrid, the next question is whether that system should search using gradient descent[21] or using a symbolic method. The single system has to jointly perform two tasks: mapping intuitions to concepts and combining concepts into rules. Of the two tasks, I believe that finding a set of rules is much more challenging. While deep neural networks are remarkably successful at mapping raw input to classes, neural networks have struggled to solve program synthesis tasks [GBS$^+$16, BRNR17, RR16, EG18]. In fact, Alex Gaunt and others [GBS$^+$16] have argued that under certain reasonable assumptions, gradient descent methods are incapable of solving hard program synthesis tasks, as the number of suboptimal local minima grows exponentially with the size of the program. In that paper, they compared various program synthesis methods and found that SAT-based methods significantly outperformed all competitors. Thus, I believe that – as of now – the most effective way to synthesise large sets of rules is to use SAT. It is, I believe, significantly easier to use SAT to find suitable weights for a binary neural network than it is to use gradient descent to find a large set of rules.

[19] This example is adapted from [CFH92].

[20] Of course, our single system itself contains both a neural network mapping intuitions to concepts and a program sythesis component that constructs sets of rules. But this counts as a single architecture rather than a hybrid architecture because our binary neural network is implemented in ASP and the weights are found using SAT, rather than gradient descent.

[21] Gradient descent is a standard optimization method for neural networks that works by repeatedly changing the network's weights by a small amount in the direction of the negative derivative of the loss function. This approach will find a local minimum but is not guaranteed to find a global minimum.

8.5.6 Alternative options

The particular design decisions taken in the Apperception Engine represent one way of answering the five questions above. But there are many other possible architectures. One option, for example, would be to represent the rules implicitly [DML+19], and to use a single neural network to jointly learn to map intuitions to concepts and to learn the weights of the implicit rules. Another option would be to use a hybrid architecture in which a neural network, trained on gradient descent, maps intuitions to concepts, while another symbolic system combines concepts into rules. These alternative options have issues of their own, as I hope the discussion above makes clear, but the point remains that the Apperception Engine is certainly not the only way to implement Kant's cognitive architecture.

8.6 Further work

I discuss various ways in which this work could be developed.

8.6.1 Implementing a probabilistic model of raw input

The approach to apperceiving raw input described in Chapter 5 suffers from three main limitations. First, it does not handle information probabilistically. The multiclass binary neural network generates a disjunction of ground atoms, representing the various properties that the object may have – but there is no way to represent that one property is more probable than another. Second, the treatment of noise in Section 3.7.6 and the treatment of raw input in Chapter 5 are separate. It would be better to have a single system that handles noisy (mislabelled) and raw (ambiguous) input jointly. Third, the apperception framework of Section 5.2 makes the restrictive assumption that the raw input can be divided into subregions in which there is at most one object in each subregion. It would be better to design a more general framework that avoids this restrictive assumption.

In future work, I plan to implement a new system that overcomes these three limitations. This new system will work as follows.

Recall that G_ϕ is the set of all ground atoms formed from type signature ϕ. Recall that a raw input sequence of length T is a sequence $(\mathbf{r}_1, ..., \mathbf{r}_T)$ in $\mathbf{R}^T$, where $\mathbf{R}$ is the set of all possible raw inputs for a single time step, e.g. the set of all 20×20 binary pixel arrays.

In the new approach, the neural network $\pi_\mathbf{w} : \mathbf{R} \rightarrow [0, 1]^{|G_\phi|}$, parameterised by weights $\mathbf{w}$, maps a raw input $\mathbf{r}_i$ to a probability distribution over ground atoms, so that $\pi_\mathbf{w}(\mathbf{r}_i)[j]$ represents the probability of the j'th atom in G_ϕ according to $\pi_\mathbf{w}(\mathbf{r}_i)$.

The neural network $\pi_\mathbf{w}$ is designed to respect the xor constraints. For each ground constraint

$\alpha_1 \oplus ... \oplus \alpha_n$, we insist that:

$$\sum_{j=1}^{n} \pi_{\mathbf{w}}(\mathbf{r}_i)[\alpha_j] = 1 \tag{8.1}$$

We guarantee that this requirement is satisfied by placing, in the final layer of the network, a softmax function between the atoms of each ground constraint. (Recall that each ground atom features in exactly one ground constraint).

Now we want to find the best $\theta, \mathbf{w}$ pair that makes sense of the raw sensory input. In other words, we want to find:

$$\arg\max_{\theta,\, \mathbf{w}} \; p(\theta) \cdot \prod_{i=1}^{T} p(\mathbf{r}_i \mid \tau(\theta)[i], \mathbf{w}) \tag{8.2}$$

Let $A = \tau(\theta)[i]$, the set of ground atoms in the trace of theory θ at time-step i.

Applying Bayes theorem, we have:

$$\begin{aligned}
p(\mathbf{r}_i \mid A_i, \mathbf{w}) \;&=\; \frac{p(A_i \mid \mathbf{r}_i) \cdot p(\mathbf{r}_i)}{p(A_i)} && (8.3)\\
&=\; \frac{\prod_{\alpha \in A_i} \pi_{\mathbf{w}}(\mathbf{r}_i)[\alpha] \cdot p(\mathbf{r}_i)}{p(A_i)} && (8.4)\\
&=\; \frac{\prod_{\alpha \in A_i} \pi_{\mathbf{w}}(\mathbf{r}_i)[\alpha] \cdot p(\mathbf{r}_i)}{\sum_{\mathbf{r}} p(A_i \mid \mathbf{r}) \cdot p(\mathbf{r})} && (8.5)\\
&=\; \frac{\prod_{\alpha \in A_i} \pi_{\mathbf{w}}(\mathbf{r}_i)[\alpha] \cdot p(\mathbf{r}_i)}{\sum_{\mathbf{r}'} \prod_{\alpha \in A_i} \pi_{\mathbf{w}}(\mathbf{r}')[\alpha] \cdot p(\mathbf{r}')} && (8.6)
\end{aligned}$$

Substituting in, we want to find:

$$\arg\max_{\theta,\, \mathbf{w}} \; p(\theta) \cdot \prod_{i=1}^{T} \frac{\prod_{\alpha \in \tau(\theta)[i]} \pi_{\mathbf{w}}(\mathbf{r}_i)[\alpha] \cdot p(\mathbf{r}_i)}{\sum_{\mathbf{r}'} \prod_{\alpha \in \tau(\theta)[i]} \pi_{\mathbf{w}}(\mathbf{r}')[\alpha] \cdot p(\mathbf{r}')} \tag{8.7}$$

Taking logs:

$$\arg\max_{\theta,\, \mathbf{w}} \; \log p(\theta) + \sum_{i=1}^{T} \sum_{\alpha \in \tau(\theta)[i]} \log \pi_{\mathbf{w}}(\mathbf{r}_i)[\alpha] - \log \sum_{\mathbf{r}'} \prod_{\alpha \in \tau(\theta)[i]} \pi_{\mathbf{w}}(\mathbf{r}')[\alpha] \cdot p(\mathbf{r}') \tag{8.8}$$

The score is thus a tradeoff between the size of the theory, how well the trace of the theory matches the atoms perceived by the neural network, and how discriminating the neural network is in mapping many raw inputs to the same set of ground atoms.

This new approach overcomes the three limitations described above:

- The system handles information probabilistically, since the neural network maps raw input to a probability distribution over ground atoms that respects the ground xor constraints.

- The system jointly handles noisy (mislabelled) and ambiguous (raw) input. Noisy input is handled robustly by trading off the size of the theory (in the prior $p(\theta)$) with how well the atoms in the theory's trace match the distribution over atoms produced by the neural network: $\prod_{\alpha \in A_i} \pi_{\mathbf{w}}(\mathbf{r}_i)[\alpha]$.
- Third, we have removed the restrictive assumption of Section 5.2 that the raw input is divided into subregions with at most one object per subregion. Instead, we use a general formulation in which the neural network can perform any mapping from the raw input to a probability distribution over ground atoms that respects the ground constraints.

8.6.2 Adding stratified negation as failure

In order to extend Datalog$^{\ni}$ so that it can handle defeasible causal rules (Section 8.4.1), we need to add negation as failure to the language. But if we allow *unrestricted* negation as failure in static rules, then we would need to use a more complex semantics (such as stable model semantics [GL88], with multiple models of each program, or well-founded models, with three truth-values), and a more complex semantics would incur a greater performance overhead. Instead, we plan to support defeasible causal rules (and non-monotonic reasoning in general) by adding *stratified* negation as failure [ABW88], as this does not affect the complexity of the semantics: a program with stratified negation has a unique minimal model. We plan to add stratified negation to static rules, and also add unrestricted negation to causal rules, since causal rules are automatically *locally stratified*, as the head atom is true at the subsequent time-step from the body atoms.

8.6.3 Allowing non-determinism

We plan to extend the Datalog$^{\ni}$ framework to support non-deterministic environments (Section 8.4.1).

One way to handle non-determinism is to extend Datalog$^{\ni}$ to include normal logic programs under the stable model semantics. In this approach, if we want to represent that p entails a non-deterministic choice between q and r, we use two clauses:

$$q :\text{-}\ p, not\ r$$
$$r :\text{-}\ p, not\ q$$

This is the approach that ILASP [LRB14, LRB15, LRB16, LRB18a] uses to model non-deterministic environments.

But there is an alternative approach to handling non-determinism that avoids the complexity of normal logic programs and the stable model semantics. We shall 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 *every* 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.

8.6.4 Supporting incremental theory revision

One important area for future work is extending the system to accommodate incremental theory revision. So far, we have worked on the assumption that the agent is given the sequence of sensory inputs as one block: the agent receives a sequence $(S_1, ..., S_t)$, and constructs a theory to make sense of that sequence; if, subsequently, the agent receives further information S_{t+1}, it constructs an entirely new theory to make sense of $(S_1, ..., S_t, S_{t+1})$. There is no support, in the current implementation, for *incremental* theory building where we reuse parts of earlier theories to make sense of new information.

At the moment, the Apperception Engine keeps track of a single model, the one with the maximum *a posteriori* probability (Section 5.3). But the simplest theory that makes sense of $(S_1, ..., S_t)$ may not be the best candidate for building a theory that makes sense of $(S_1, ..., S_t, S_{t+1})$. Thus, in order to support incremental theory revision, we shall move away from keeping track of a single model, and instead keep track of a *distribution* over different theories.

8.6.5 Integrating with practical reasoning

Another area for future work is extending the system to incorporate practical reasoning. So far, we have focused on Kant's theoretical philosophy (in the first *Critique*) and have avoided discussing his practical philosophy (in the second *Critique*). The Apperception Engine, in its initial incarnation, is a *mere bystander*: it receives sensory input and tries to make sense of that input – but it does not act on that understanding. It has no desires, intentions, goals, or world-directed activity. It just sits there, thinking, like a sentient tree.

Kant believed that practical and theoretical reason share a "common principle" [Groundwork 4:391]. This common principle is unity via *self-legislation*: just as (in the theoretical sphere) I construct rules to unify my intuitions together into one coherent experience, just so (in the practical sphere) I construct maxims to unify my actions together into the activities of one coherent person [Kor09]. Just as the judgements I construct must satisfy unity conditions in order to achieve a coherent experience, just so the maxims I construct must satisfy the categorical imperative in order that I may achieve the status of being a coherent person. In future work, I want to build a new incarnation of the Apperception Engine that synthesises maxims, tests them for universalisability, and then acts.

8.6.6 Moving closer to a faithful implementation of Kant's *a priori* psychology

This project is an attempt to repurpose Kant's *a priori* psychology as the architectural blueprint for a machine learning system, and as such has the real potential to irritate two distinct groups of people. AI practitioners may be irritated by the appeal to a notoriously difficult eighteenth-century text, while Kant scholars may be irritated by the indelicate attempt to shoe-horn Kant's ambitious system into a simple computational formalism. The concern is that Kant's ideas have been distorted to the point where they are no longer recognisable.

In what ways, then, does the Apperception Engine represent a faithful implementation of Kant's vision, and in what ways does it fall short?

I shall focus, first, on the respects in which the computer architecture is a faithful implementation of Kant's psychological theory. Kant proposed various faculties that interoperate to turn raw data into experience: the imagination (to connect intuitions together using the pure relations as glue), the power of judgement (to decide whether an intuition falls under a concept), and the capacity to judge (to generate judgements from concepts). Throughout, Kant emphasized the spontaneity of the mind: the faculties are free to perform *whatever activity they like*, as long as the resulting system satisfies satisfies the various unity conditions described in the *Principles*.

The Apperception Engine provides a unified implementation of the various faculties Kant describes: the imagination is implemented as a set of non-deterministic choice rules, the power of judgement is implemented as a neural network, and the capacity to judge is implemented as an unsupervised program synthesis system. These sub-systems are highly non-deterministic: the imagination is free to synthesise the intuitions *in any way whatsoever*, the power of judgement is free to map intuitions to concepts *in any way it pleases*, and the capacity to judge is free to construct *any rules at all* – so long as the combined product of the three faculties satisfies the various unity conditions (implemented as *constraints*[22]).

Thus, while contingent information flows bottom-up (from sensibility to the understanding), necessary information flows top-down, as the unity conditions of the understanding are the only constraints on the operations of the system. As Kant says: "through it [the constraint of unity] the understanding determines the sensibility [B160-1n]". This is, I believe, a faithful implementation of Kant's cognitive architecture at a high level.

Next I shall turn to the various respects in which the computer architecture described above falls short of Kant's ambitious vision of how the mind must work. I shall focus on six aspects of Kant's cognitive architecture that are not adequately represented in the current implementation.

§1. The way in which raw data is given to the Apperception Engine is different from how Kant describes it. Kant describes a cognitive agent receiving a *continuous* stream of information, making sense of each segment before receiving the next. The Apperception Engine, by contrast, is given the

[22] See Sections 3.3 and 6.4)

entire stream as a single unit. If the Apperception Engine is to operate with a continuous stream, it will have to synthesise a new theory *from scratch* each time it receives a new piece of information (Section 8.6.4).

In the *A Deduction*, Kant describes three aspects of synthesis: the synthesis of apprehension in the intuition, the synthesis of reproduction in the imagination, and the synthesis of recognition in a concept. The synthesis of reproduction in the imagination involves the ability to recall past experiences that are no longer present in sensation. The Apperception Engine does not attempt to model the synthesis of reproduction. Rather, it assumes that the entire sequence is given.

The form of the raw data is also different from how Kant describes it. In Section 6.12.1, the raw data is provided as a sequence of *determinations*: assignments of raw attributes to persistent objects (sensors). Here, we assume that the agent is provided with the sensor, as a persistent object. But in Kant's architecture, the construction of determinations featuring persistent objects is a hard-won achievement - not something that is given. What *is* given, in Kant's picture, is the activity of sensing and the ability to tell when a particular sensing performed at one moment is the same sensing activity performed at another (the "unity of the action"). Thus, in Kant's picture, the agent is provided with a more minimal initial input than that given to our system, and so his agent has more work to do to achieve experience.

§2. The way space is represented in the Apperception Engine is different from how Kant describes it. For Kant, space is a single *a priori* intuition. He starts with space as a totality, and creates sub-spaces by division ("limitation" [A25/B39]). In the Apperception Engine, by contrast, we start with objects representing spatial regions, and compose them together using the containment structure (Section 6.5).

Similarly, with time, Kant starts with the original representation of the whole of time, and constructs sub-times by division [A32/B48]. In the Apperception Engine, by contrast, the sequence of time-steps are determined by the given input, and it is not possible for the system in its current form to construct new moments of time that are intermediate between the given moments. Relatedly, it is not possible to represent continuous causality (e.g. water slowly filling a container) in our formalism. In future work, we plan to enrich Datalog$^{\ni}$ so that it can represent continuous change.

§3. The Apperception Engine unifies objects by placing them in a containment structure: each object is in some spatial region which is itself part of some larger spatial region, until we reach the whole of space. In Section 6.5, I argued that this containment structure is a central component of any notion of space. But there is much more to spatial relations than the containment structure: just knowing that x and y are in z does not tell us anything about the relative positions of x and y.

Kant had a much more full-blooded conception of space than just a containment structure: he assumed three-dimensional Euclidean space [B41]. In future work, I plan to provide the Apperception Engine with three-dimensional space[23], thus providing a stronger inductive bias, which should help the

[23] Perhaps by providing an axiomatisation of Euclidean space using Tarski's formalisation [Tar67], or somesuch (but note

system to learn more data-efficiently.

§4. In the *Transcendental Deduction*, Kant argued that the relative positions of intuitions in a determination can only be fixed by forming a judgement that necessitates this particular positioning [B128]. The Apperception Engine attempts to respect this fundamental requirement by insisting that the various connections between intuitions (described in Section 6.9.1) are backed up by judgements of various forms (Section 6.6). However, the forms of judgement supported in Datalog$^{\ni}$ are a mere subset of the forms enumerated in the *Table of Judgements* [A70/B95]. Datalog$^{\ni}$ supports universally quantified conditionals, causal conditionals, and xor constraints (corresponding to Kant's disjunctive judgement). But it does not support negative judgements, infinite judgements, particular judgements, singular judgements, or modal judgements. In future work, we plan to extend the expressive power of Datalog$^{\ni}$ to capture the full range of propositions expressible in the *Table of Judgements*.[24]

§5. The *Third Analogy* states that whenever two objects' determinations are perceived as simultaneous, there must be a two way interaction between the two objects. This does not mean, of course, that there must be a direct causal influence between them, but just that there must be a chain of indirect causal influences between them.

This requirement has not been implemented in the Apperception Engine. This is because it would make it very hard for the system to find any unified interpretation at all if every time it posited a simultaneity between determinations it also had to construct some rules whereby one determination of one object indirectly caused some determination of the other object. Longueness [Lon98] has a different understanding of the second and third *Analogies*, and does not believe that we need to *have actually formed a causal rule* in order to perceive succession or simultaneity. In her interpretation, we merely need to *believe that there is a causal rule to find* (see Section 6.6.2 for a discussion). However, in our interpretation, in which the rule must actually be found before a temporal relation can be assigned, the *Third Analogy* does seem restrictively strong. In future work, we hope to address this issue and find a way to respect the simultaneity constraint.

§6.

The first *Critique* contains various discussions of various aspects of self-consciousness. But no aspect of self-consciousness is implemented in the Apperception Engine. In the *B Deduction*, Kant distinguishes the synthetic unity of apperception (the connecting together of one's intuitions via the pure relations of Section 6.9.1 in such a way as to achieve unity) from the analytic unity of apperception (the ability to subsume any of my cognitions under the predicate "I think"). He claims that synthetic unity of apperception is a necessary condition for achieving analytic unity [B133-4]. Although the Apperception Engine aims to implement the synthetic unity of apperception, no attempt has been made to implement the analytic unity of apperception.

that axiomatising Euclidean geometry requires ternary predicates, which are not currently handled in the Apperception Engine). But Tarski assumes points as primitive, where a point is defined as a vector of real numbers. It would be closer to Kant's program, I believe, to axiomatise space starting from the notion of *limitation*, without assuming real numbers as given.

[24] By contrast, the geometric logic used in [AVL11, AvL$^+$17] is much more expressive.

Kant is clear to distinguish between inner sense and explicit self-consciousness [B154]. Inner sense is the aspect of sensibility in which the mind perceives its own mental activity: it notices the formation of a belief, for example, or the application of a rule. Inner sense provides us with intuitions that must be ordered in time. Explicit self-consciousness, by contrast, is the construction of a theory that makes sense of the sequence of perturbations produced by inner sense. In inner sense I become aware of some of the cognitions I am having, and in explicit self-consciousness, I posit a theory that explains the dynamics of my own mental activity – although this hypothesized theory may or may not reflect accurately the actual mental processes I am undergoing [B156]. In future work, I plan to extend the Apperception Engine so that (some of) its own activity is perceptible via inner sense, so that the system is forced to construct a theory to make sense of its perceptions of its own mental activity.

There are, then, various aspects of Kant's theory of mental activity that are not captured in the current incarnation of the Apperception Engine. There is, I think it is fair to say, more work still to do.

8.7 Conclusion

The guiding assumption behind this project is that AI has something to learn from Kant's *a priori* psychology. In the *Critique of Pure Reason*, Kant asks: what activities must be performed by an agent – *any* finite resource-bounded agent – if it is to make sense of its sensory input. This is not an empirical question about the particular activities that are performed by *homo sapiens*, but an *a priori* question about the activities that any agent must perform. Kant's answer, if correct, is important because it provides a blueprint for the space of *all possible minds* – not just our particular human minds with their particular human foibles.[25]

If Kant's cognitive architecture is along the right lines, this will have significant impact on how we should design intelligent machines. Consider, to take one important recent example, the data efficiency of contemporary reinforcement learning systems. Recently, deep reinforcement learning agents have achieved super-human ability in a variety of games, including Atari [MKS$^+$13] and Go [SSS$^+$17]. These systems are very impressive, but also very data-inefficient, requiring an enormous quantity of training data. DQN [MKS$^+$13] requires 200 million frames of experience before it can reach human performance on Atari games. This is equivalent to playing non-stop for 40 days. AlphaZero [SSS$^+$17] played 44 million games to reach its performance level.

Pointing out the sample complexity of these programs is not intended to criticise these accomplishments in any way. They are very impressive achievements. But it does point to a fundamental difference between the way these machines learn to play the game, and the way that humans do. A human can look at a new Atari game for a few minutes, and then start playing well. He or she does

[25] "In the history of human inquiry, philosophy has the place of the central sun, seminal and tumultuous: from time to time it throws off some portion of itself to take station as a science, a planet, cool and well regulated, progressing steadily towards a distant final state." – Austin, *Ifs and Cans* [Aus56]

not need to play non-stop for 40 days. A human's data efficiency at an Atari game is a consequence of our inductive bias: we start with prior knowledge that informs and guides our search.

It is a commonplace that the stronger the inductive bias, the more data-efficiently a system can learn. But the danger, of course, with injecting inductive bias into a machine, is that it biases the system, enabling it to learn some tasks quicker, but preventing it from learning other tasks effectively. What we really want, if only we can get it, is inductive bias that is maximally general. But what are these maximally general concepts that we should inject into the machine, and how do we do so?

Neural net practitioniers, for all their official espousal of pure empiricist anti-innatism, do (in practice) acknowledge the need for certain minimal forms of inductive bias. A convolutional net [LB+95] is a particular neural architecture that is designed to enforce the constraint that the same invariants hold no matter where the objects appear in the retinal field. A long short-term memory [HS97] is a particular neural architecture that is designed to enforce the constraint that invariants that are valid at one point in time are also valid at other points in time. But these are isolated examples. *What, then, are the maximally general concepts that we should inject into the machine, to enable data efficient learning?*

The answer to this question has been lurking in plain sight for over two hundred years. In the first *Critique*, Kant identified the maximally general concepts, showed how these concepts structure perception itself, and identified the conditions specifying how the pure concepts interoperate. Kant's principles provide the maximally general inductive bias we need to make our machines data-efficient.

I want to conclude by arguing for a stronger claim. A rule-synthesising system satisfying Kant's unity conditions is not merely sufficient for data efficient learning – it is necessary:

1. In order to achieve data efficiency, we need strong priors
2. If the strong priors are domain-specific, the agent will not able to operate in a wide range of environments
3. Therefore, we need domain-agnostic strong priors
4. The only domain-agnostic strong priors are the Kantian unity conditions
5. The Kantian unity conditions are constraints on the construction of rules
6. Therefore, any data-efficient sense-making agent that can operate in a wide range of environments must construct rules that satisfy the Kantian unity conditions

Bibliography

[ABW88] Krzysztof R Apt, Howard A Blair, and Adrian Walker. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Elsevier, 1988.

[AF18] Masataro Asai and Alex Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *AAAI Conference on Artificial Intelligence*, 2018.

[All09] Lucy Allais. Kant, non-conceptual content and the representation of space. *Journal of the History of Philosophy*, 47(3):383–413, 2009.

[Asa19] Masataro Asai. Unsupervised grounding of plannable first-order logic representation from images. *arXiv preprint arXiv:1902.08093*, 2019.

[Aus56] John Langshaw Austin. Ifs and cans. *Proceedings of the British Academy*, 1956.

[AVL11] Theodora Achourioti and Michiel Van Lambalgen. A formalization of Kant's transcendental logic. *The Review of Symbolic Logic*, 4(2):254–289, 2011.

[AvL$^+$17] Theodora Achourioti, Michiel van Lambalgen, et al. Kant's logic revisited. *IfCoLog Journal of Logics and Their Applications*, 4:845–865, 2017.

[BBDS$^+$08] Lucian Bu, Robert Babu, Bart De Schutter, et al. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[BD15] Tarek R Besold and OSNABRUECK DE. The artificial jack of all trades: The importance of generality in approaches to human-level artificial intelligence. In *Proceedings of the Third Annual Conference on Advances in Cognitive Systems (ACS)*, page 18, 2015.

[BED94] Rachel Ben-Eliyahu and Rina Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial intelligence*, 12(1-2):53–87, 1994.

[Bez05] Marc Bezem. On the undecidability of coherent logic. In *Processes, Terms and Cycles: Steps on the Road to Infinity*, pages 6–13. Springer, 2005.

[BHS^{+}18] David GT Barrett, Felix Hill, Adam Santoro, Ari S Morcos, and Timothy Lillicrap. Measuring abstract reasoning in neural networks. *arXiv preprint arXiv:1807.04225*, 2018.

[BMSF18] Apratim Bhattacharyya, Mateusz Malinowski, Bernt Schiele, and Mario Fritz. Long-term image boundary prediction. In *AAAI Conference on Artificial Intelligence*, 2018.

[BNT03] Gerhard Brewka, Ilkka Niemelä, and Miroslaw Truszczynski. Answer set optimization. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, volume 3, pages 867–872, 2003.

[BNVB13] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research (JAIR)*, 47:253–279, 2013.

[BP66] Leonard E Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state Markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 1966.

[BPL^{+}16] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems (NEURIPS)*, pages 4502–4510, 2016.

[Bra94] Robert Brandom. *Making It Explicit*. Harvard University Press, 1994.

[Bra08] Robert B Brandom. *Between Saying and Doing*. Oxford University Press, 2008.

[Bra09] Robert Brandom. How analytic philosophy has failed cognitive science. *Towards an Analytic Pragmatism (TAP)*, pages 121–133, 2009.

[BRNR17] Matko Bošnjak, Tim Rocktäschel, Jason Naradowsky, and Sebastian Riedel. Programming with a differentiable Forth interpreter. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 547–556. JMLR. org, 2017.

[BWR^{+}18] Lars Buesing, Theophane Weber, Sebastien Racaniere, SM Eslami, Danilo Rezende, David P Reichert, Fabio Viola, Frederic Besse, Karol Gregor, Demis Hassabis, et al. Learning and querying fast generative models for reinforcement learning. *arXiv preprint arXiv:1802.03006*, 2018.

[Cav99] Stanley Cavell. *The Claim of Reason*. Oxford University Press, 1999.

[CEL19] Andrew Cropper, Richard Evans, and Mark Law. Inductive general game playing. *Machine Learning*, pages 1–42, 2019.

[CFG^{+}12] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub. Asp-core-2: Input language format. *ASP Standardization Working Group*, 2012.

[CFH92] David J Chalmers, Robert M French, and Douglas R Hofstadter. High-level perception, representation, and analogy: A critique of artificial intelligence methodology. *Journal of Experimental & Theoretical Artificial Intelligence*, 4(3):185–211, 1992.

[CJS90] Patricia A Carpenter, Marcel A Just, and Peter Shell. What one intelligence test measures: a theoretical account of the processing in the Raven Progressive Matrices test. *Psychological review*, 97(3):404, 1990.

[Cla78] Keith L Clark. Negation as failure. In *Logic and data bases*, pages 293–322. Springer, 1978.

[Cla13] Andy Clark. Whatever next? Predictive brains, situated agents, and the future of cognitive science. *Behavioral and Brain Sciences*, 36(3):181–204, 2013.

[CM15] Andrew Cropper and Stephen H Muggleton. Logical minimisation of meta-rules within meta-interpretive learning. In *Inductive Logic Programming*, pages 62–75. Springer, 2015.

[CM16] Andrew Cropper and Stephen H. Muggleton. Learning higher-order logic programs through abstraction and invention. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1418–1424. IJCAI/AAAI Press, 2016.

[CM18] Andrew Cropper and Stephen H. Muggleton. Learning efficient logic programs. *Machine Learning*, pages 1–21, 2018.

[CNHR18] Chih-Hong Cheng, Georg Nührenberg, Chung-Hao Huang, and Harald Ruess. Verification of binarized neural networks via inter-neuron factoring. In *Working Conference on Verified Software: Theories, Tools, and Experiments*, pages 279–290. Springer, 2018.

[Coh60] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

[Coo04] Matthew Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1):1–40, 2004.

[Cor12] Domenico Corapi. Nonmonotonic Inductive Logic Programming as Abductive Search. PhD thesis, Imperial College London, 2012.

[CRL10a] Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive logic programming as abductive search. In *Technical Communications of the 26th International Conference on Logic Programming*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

[CRL10b] Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive logic programming as abductive search. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 7, pages 34–41. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

[CRL11a] Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive logic programming in answer set programming. In *International Conference on Inductive Logic Programming*, pages 91–97. Springer, 2011.

[CRL11b] Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive logic programming in answer set programming. In *International Conference on Inductive Logic Programming*, pages 91–97. Springer, 2011.

[CRL12] Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive logic programming in answer set programming. In *Inductive Logic Programming*, pages 91–97. Springer, 2012.

[Cro17] Andrew Cropper. Efficiently Learning Efficient Programs. PhD thesis, Imperial College London, UK, 2017.

[Cro19] Andrew Cropper. Playgol: learning programs through play. *arXiv preprint arXiv:1904.08993*, 2019.

[CRWM17] Silvia Chiappa, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. *arXiv preprint arXiv:1704.02254*, 2017.

[CSIR11] Domenico Corapi, Daniel Sykes, Katsumi Inoue, and Alessandra Russo. Probabilistic rule learning in nonmonotonic domains. In *International Workshop on Computational Logic in Multi-Agent Systems*, pages 243–258. Springer, 2011.

[CT18] Andrew Cropper and Sophie Tourret. Derivation reduction of metarules in meta-interpretive learning. In *International Conference on Inductive Logic Programming*, pages 1–21. Springer, 2018.

[CUTT16] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

[CW03] Lourdes Peña Castillo and Stefan Wrobel. Learning minesweeper with multirelational learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 533–540. Morgan Kaufmann, 2003.

[DDG+08] Thomas G Dietterich, Pedro Domingos, Lise Getoor, Stephen Muggleton, and Prasad Tadepalli. Structured machine learning: the next ten years. *Machine Learning*, 73(1):3, 2008.

[DDRD01] Sašo Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine Learning*, 43(1-2):7–52, 2001.

[DEGV01] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys (CSUR)*, 33(3):374–425, 2001.

[Den78] Daniel C Dennett. Artificial intelligence as philosophy and as psychology. *Brainstorms*, pages 109–26, 1978.

[DML$^+$19] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural logic machines. *arXiv preprint arXiv:1904.11694*, 2019.

[DN15] Roy Dyckhoff and Sara Negri. Geometrisation of first-order logic. *Bulletin of Symbolic Logic*, 21(2):123–163, 2015.

[DR08] Luc De Raedt. *Logical and Relational Learning*. Springer Science & Business Media, 2008.

[DR12] Luc De Raedt. Declarative modeling for machine learning and data mining. In *International Conference on Formal Concept Analysis*, pages 2–2. Springer, 2012.

[EG18] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research (JAIR)*, 61:1–64, 2018.

[ELH14] Tom Everitt, Tor Lattimore, and Marcus Hutter. Free lunch for optimisation under the universal distribution. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 167–174. IEEE, 2014.

[ESLT15] Kevin Ellis, Armando Solar-Lezama, and Josh Tenenbaum. Unsupervised learning by program synthesis. In *Advances in Neural Information Processing Systems (NEURIPS)*, pages 973–981, 2015.

[ESS19] Richard Evans, M Sergot, and A Stephenson. Formalizing Kant's rules. *Journal of Philosophical Logic*, pages 1–68, 2019.

[Eva17] Richard Evans. Kant on constituted mental activity. *APA on Philosophy and Computers*, 2017.

[Eva19] Richard Evans. A Kantian cognitive architecture. In *On the Cognitive, Ethical, and Scientific Dimensions of Artificial Intelligence*, pages 233–262. Springer, 2019.

[FL17] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE, 2017.

[Fod75] Jerry A Fodor. *The Language of Thought*. Harvard University Press, 1975.

[FP88] Jerry A Fodor and Zenon W Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71, 1988.

[Fri92] Michael Friedman. *Kant and the Exact Sciences*. Harvard University Press, 1992.

[Fri05] Karl Friston. A theory of cortical responses. *Philosophical Transactions of the Royal Society B: Biological sciences*, 360(1456):815–836, 2005.

[Fri12] Karl Friston. The history of the future of the Bayesian brain. *NeuroImage*, 62(2):1230–1233, 2012.

[FWS+18] V Feinberg, A Wan, I Stoica, MI Jordan, JE Gonzalez, and S Levine. Model-based value expansion for efficient model-free reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[GAS16] Marta Garnelo, Kai Arulkumaran, and Murray Shanahan. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518*, 2016.

[GB13] Michael Genesereth and Yngvi Björnsson. The international general game playing competition. *AI Magazine*, 34(2):107–107, 2013.

[GBS+16] Alexander L Gaunt, Marc Brockschmidt, Rishabh Singh, Nate Kushman, Pushmeet Kohli, Jonathan Taylor, and Daniel Tarlow. Terpret: A probabilistic programming language for program induction. *arXiv preprint arXiv:1608.04428*, 2016.

[Gha01] Zoubin Ghahramani. An introduction to hidden Markov models and Bayesian networks. In *Hidden Markov models: Applications in Computer Vision*, pages 9–41. World Scientific, 2001.

[GK14] Michael Gelfond and Yulia Kahl. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents*. Cambridge University Press, 2014.

[GKKS12] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Answer set solving in practice. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(3):1–238, 2012.

[GKKS14] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo= ASP+ control. *arXiv preprint arXiv:1405.3694*, 2014.

[GKS11] Martin Gebser, Roland Kaminski, and Torsten Schaub. Complex optimization in answer set programming. *Theory and Practice of Logic Programming*, 11(4-5):821–839, 2011.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Logic Programming: Proc. Fifth International Conference on Logic Programming*, volume 88, pages 1070–1080. MIT Press, 1988.

[GL05] Michael Genesereth and Nathaniel Love. General game playing: Game description language specification. *Computer Science Department, Stanford University, Stanford, CA, USA, Tech. Rep*, 2005.

[Gom13] Anil Gomes. Kant on perception: Naive realism, non-conceptualism, and the B-deduction. *The Philosophical Quarterly*, 64(254):1–19, 2013.

[Goo96] John Goodacre. Inductive Learning of Chess Rules using Progol. PhD thesis, University of Oxford, 1996.

[GPS+17] Sumit Gulwani, Oleksandr Polozov, Rishabh Singh, et al. Program synthesis. *Foundations and Trends in Programming Languages*, 4(1-2):1–119, 2017.

[GS14] Dedre Gentner and Albert L Stevens. *Mental Models*. Psychology Press, 2014.

[GSBS15] Peter Gregory, Henrique Coli Schumann, Yngvi Björnsson, and Stephan Schiffel. The GRL system: learning board game rules with piece-move interactions. In *Computer Games*, pages 130–148. Springer, 2015.

[GT17] Tobias Gerstenberg and Joshua B Tenenbaum. Intuitive theories. *Oxford Handbook of Causal Reasoning*, pages 515–548, 2017.

[GUT11] Noah D Goodman, Tomer D Ullman, and Joshua B Tenenbaum. Learning a theory of causality. *Psychological Review*, 118(1):110, 2011.

[Ham19] Jessica B Hamrick. Analogues of mental simulation and imagination in deep learning. *Current Opinion in Behavioral Sciences*, 29:8–16, 2019.

[Har00] Paul L Harris. *The Work of the Imagination*. Blackwell Publishers, Oxford, 2000.

[Hau90] John Haugeland. The intentionality all-stars. *Philosophical Perspectives*, pages 383–427, 1990.

[Haz99] Allen Patterson Hazen. Logic and analyticity. In *The Nature of Logic*, pages 79–110. CSLI, 1999.

[HCS+16] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems (NEURIPS)*, pages 4107–4115, 2016.

[Heg04] Mary Hegarty. Mechanical reasoning by mental simulation. *Trends in Cognitive Sciences*, 8(6):280–285, 2004.

[HMP+17] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. Beta-VAE: Learning basic visual concepts with a constrained variational framework. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2(5):6, 2017.

[HO00] Jose Hernandez-Orallo. Beyond the Turing test. *Journal of Logic, Language and Information*, 9(4):447–466, 2000.

[HO17] José Hernández-Orallo. *The Measure of All Minds*. Cambridge University Press, 2017.

[Hof95] Douglas R Hofstadter. *Fluid Concepts and Creative Analogies*. Basic Books, 1995.

[Hol09] AO Holcombe. The binding problem. *The Sage Encyclopedia of Perception*, 2009.

[HOMC98] José Hernandez-Orallo and Neus Minaya-Collado. A formal definition of intelligence. In *Proceedings of International Symposium of Engineering of Intelligent Systems (EIS 98)*, pages 146–163, 1998.

[HOMPS$^+$16] José Hernández-Orallo, Fernando Martínez-Plumed, Ute Schmid, Michael Siebers, and David L Dowe. Computer models solving intelligence test problems: Progress and implications. *Artificial Intelligence*, 230:74–107, 2016.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[HS18] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems (NEURIPS)*, pages 2455–2467, 2018.

[IBN05] Katsumi Inoue, Hideyuki Bando, and Hidetomo Nabeshima. Inducing causal laws by regular inference. In *International Conference on Inductive Logic Programming*, pages 154–171. Springer, 2005.

[IRS14] Katsumi Inoue, Tony Ribeiro, and Chiaki Sakama. Learning from interpretation transition. *Machine Learning*, 94(1):51–79, 2014.

[Jay00] Julian Jaynes. *The Origin of Consciousness in the Breakdown of the Bicameral Mind*. Houghton Mifflin Harcourt, 2000.

[JBBS90] William James, Frederick Burkhardt, Fredson Bowers, and Ignas K Skrupskelis. *The Principles of Psychology*, volume 1. Macmillan London, 1890.

[JGP16] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[JL12] Philip N Johnson-Laird. Inference with mental models. *The Oxford Handbook of Thinking and Reasoning*, pages 134–145, 2012.

[JLF$^+$18] Michael Janner, Sergey Levine, William T Freeman, Joshua B Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-oriented prediction and planning. *arXiv preprint arXiv:1812.10972*, 2018.

[Kai12] Lukasz Kaiser. Learning games from videos guided by descriptive complexity. In *AAAI Conference on Artificial Intelligence*, 2012.

[Kam79] Hans Kamp. Events, instants and temporal reference. In *Semantics from Different Points of View*, pages 376–418. Springer, 1979.

[Kan84] Immanuel Kant. What is enlightenment? In *Practical Philosophy*, pages 11–22. Cambridge University Press, 1784.

[Kan90] Immanuel Kant. *Critique of the Power of Judgment*. Cambridge University Press, 1790.

[Kan97] Immanuel Kant. The metaphysics of morals. In *Practical Philosophy*, pages 353–604. Cambridge University Press, 1797.

[KAP15] Nikos Katzouris, Alexander Artikis, and Georgios Paliouras. Incremental learning of event definitions with inductive logic programming. *Machine Learning*, 100(2-3):555–585, 2015.

[KAP16] Nikos Katzouris, Alexander Artikis, and Georgios Paliouras. Online learning of event definitions. *Theory and Practice of Logic Programming*, 16(5-6):817–833, 2016.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KBM19] Lukasz Kaiser, Mohammad Babaeizadeh, and Piotr Milos. Model based reinforcement learning for Atari. *arXiv preprint arXiv:1903.00374v2*, 2019.

[Kol63] Andrei N Kolmogorov. On tables of random numbers. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 369–376, 1963.

[Kor09] Christine M Korsgaard. *Self-Constitution*. Oxford University Press, 2009.

[KS86] Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–96, 1986.

[KS16] Minje Kim and Paris Smaragdis. Bitwise neural networks. *arXiv preprint arXiv:1601.06071*, 2016.

[LB$^+$95] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks*, 3361(10):1995, 1995.

[Lev73] Leonid Anatolevich Levin. Universal sequential search problems. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973.

[LGF16] Adam Lerer, Sam Gross, and Rob Fergus. Learning physical intuition of block towers by example. *arXiv preprint arXiv:1603.01312*, 2016.

[LGLGG18] Miguel Lázaro-Gredilla, Dianhuan Lin, J Swaroop Guntupalli, and Dileep George. Beyond imitation: Zero-shot task transfer on robots by learning concepts as cognitive programs. *arXiv preprint arXiv:1812.02788*, 2018.

[LH13] Tor Lattimore and Marcus Hutter. No free lunch versus Occam's razor in supervised learning. In *Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence*, pages 223–235. Springer, 2013.

[LMR92] Jorge Lobo, Jack Minker, and Arcot Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT press, 1992.

[Lon98] Béatrice Longuenesse. *Kant and the Capacity to Judge*. Princeton University Press, 1998.

[LRB14] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs. In *European Conference on Logics in Artificial Intelligence - (JELIA)*, pages 311–325, 2014.

[LRB15] Mark Law, Alessandra Russo, and Krysia Broda. Learning weak constraints in answer set programming. *Theory and Practice of Logic Programming*, 15(4-5):511–525, 2015.

[LRB16] Mark Law, Alessandra Russo, and Krysia Broda. Iterative learning of answer set programs from context dependent examples. *Theory and Practice of Logic Programming*, 16(5-6):834–848, 2016.

[LRB18a] Mark Law, Alessandra Russo, and Krysia Broda. The complexity and generality of learning answer set programs. *Artificial Intelligence*, 259:110–146, 2018.

[LRB18b] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs from noisy examples. *arXiv preprint arXiv:1808.08441*, 2018.

[LRB$^+$20] Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda, and Jorge Lobo. FastLAS: Scalable inductive logic programming incorporating domain-specific optimisation criteria. In *AAAI*, pages 2877–2885, 2020.

[LST15] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[LUTG17] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2017.

[LV08] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*, volume 3. Springer, 2008.

[MAP18] Evangelos Michelioudakis, Alexander Artikis, and Georgios Paliouras. Semi-supervised online structure learning for composite event recognition. *arXiv preprint arXiv:1803.00546*, 2018.

[Mar18a] Gary Marcus. *The Algebraic Mind*. MIT press, 2018.

[Mar18b] Gary Marcus. Innateness, AlphaZero, and artificial intelligence. *arXiv preprint arXiv:1801.05667*, 2018.

[McC06] John McCarthy. Challenges to machine learning: Relations between reality and appearance. In *International Conference on Inductive Logic Programming*, pages 2–9. Springer, 2006.

[McL16] Colin McLear. Kant on perceptual content. *Mind*, 125(497):95–144, 2016.

[McN47] Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

[MCO19] Rolf Morel, Andrew Cropper, and Luke Ong. Typed meta-interpretive learning of logic programs. In *European Conference on Logics in Artificial Intelligence - (JELIA)*, pages 973–981, 2019.

[MDS$^+$18] Stephen Muggleton, Wang-Zhou Dai, Claude Sammut, Alireza Tamaddoni-Nezhad, Jing Wen, and Zhi-Hua Zhou. Meta-interpretive learning from noisy images. *Machine Learning*, 107(7):1097–1118, 2018.

[Mer86] Marsha J Ekstrom Meredith. Seek-whence: A model of pattern perception. Technical report, Indiana University (USA), 1986.

[Mic83] Ryszard S Michalski. A theory and methodology of inductive learning. In *Machine learning*, pages 83–134. Springer, 1983.

[Mit93] Melanie Mitchell. *Analogy-Making as Perception*. MIT Press, 1993.

[MKS$^+$13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[MLPTN14] Stephen H Muggleton, Dianhuan Lin, Niels Pahlavi, and Alireza Tamaddoni-Nezhad. Meta-interpretive learning: application to grammatical inference. *Machine Learning*, 94(1):25–49, 2014.

[MLT15] Stephen H. Muggleton, Dianhuan Lin, and Alireza Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. *Machine Learning*, 100(1):49–73, 2015.

[MMT16] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[Mor96] Eduardo M. Morales. Learning playing strategies in chess. *Computational Intelligence*, 12:65–87, 1996.

[Moy02] Steve Moyle. Using theory completion to learn a robot navigation control program. In *International Conference on Inductive Logic Programming*, pages 182–197. Springer, 2002.

[MSPA16] Evangelos Michelioudakis, Anastasios Skarlatidis, Georgios Paliouras, and Alexander Artikis. Online structure learning using background knowledge axiomatization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 232–247. Springer, 2016.

[MSZ$^+$18] Stephen H Muggleton, Ute Schmid, Christina Zeller, Alireza Tamaddoni-Nezhad, and Tarek Besold. Ultra-strong machine learning: comprehensibility of programs learned with ILP. *Machine Learning*, 107(7):1119–1140, 2018.

[Mue14] Erik T Mueller. *Commonsense Reasoning*. Morgan Kaufmann, 2014.

[Mur12] Kevin P Murphy. *Machine Learning: a Probabilistic Perspective*. MIT press, 2012.

[MZW$^+$18] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li F Fei-Fei, Josh Tenenbaum, and Daniel L Yamins. Flexible neural representation for physics prediction. In *Advances in Neural Information Processing Systems (NEURIPS)*, pages 8813–8824, 2018.

[NKFL18] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey 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)*, pages 7559–7566. IEEE, 2018.

[NKR$^+$18] Nina Narodytska, Shiva Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. In *AAAI Conference on Artificial Intelligence*, pages 6615–6624, 2018.

[OGL$^+$15] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in Atari games. In *Advances in Neural Information Processing Systems (NEURIPS)*, pages 2863–2871, 2015.

[Ote05] Ramon P Otero. Induction of the indirect effects of actions by monotonic methods. In *International Conference on Inductive Logic Programming*, pages 279–294. Springer, 2005.

[Pia11] Steven Piantadosi. *Learning and the Language of Thought*. PhD thesis, Massachusetts Institute of Technology, 2011.

[Pin17] Riccardo Pinosio. The Logic of Kant's Temporal Continuum. PhD thesis, University of Amsterdam, 2017.

[PVL18] Riccardo Pinosio and Michiel Van Lambalgen. The logic and topology of Kant's temporal continuum. *The Review of Symbolic Logic*, 11(1):160–206, 2018.

[PZK07] Hanna M Pasula, Luke S Zettlemoyer, and Leslie Pack Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research (JAIR)*, 29:309–352, 2007.

[Ray09] Oliver Ray. Nonmonotonic abductive inductive learning. *Journal of Applied Logic*, 7(3):329–340, 2009.

[RGRS11] Christophe Rodrigues, Pierre Gérard, Céline Rouveirol, and Henry Soldano. Active learning of relational action models. In *International Conference on Inductive Logic Programming*, pages 302–316. Springer, 2011.

[RR16] Tim Rocktäschel and Sebastian Riedel. Learning knowledge base inference with neural theorem provers. In *Proceedings of the 5th Workshop on Automated Knowledge Base Construction*, pages 45–50, 2016.

[RWR$^+$17] Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems (NEURIPS)*, pages 5690–5701, 2017.

[Sel67] Wilfrid Sellars. Some remarks on Kant's theory of experience. In *In the Space of Reasons*, pages 437–453. Harvard University Press, 1967.

[Sel68] Wilfrid Sellars. *Science and Metaphysics*. Routledge, 1968.

[Sel78] Wilfrid Sellars. The role of imagination in Kant's theory of experience. In *In the Space of Reasons*, pages 454–466. Harvard University Press, 1978.

[SGHS$^+$18] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242*, 2018.

[SHS$^+$18] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[SK63] Herbert A Simon and Kenneth Kotovsky. Human acquisition of concepts for sequential patterns. *Psychological Review*, 70(6):534, 1963.

[SK07] Elizabeth S Spelke and Katherine D Kinzler. Core knowledge. *Developmental Science*, 10(1):89–96, 2007.

[SLTB+06] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. *ACM Sigplan Notices*, 41(11):404–415, 2006.

[SSS+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[Ste13] Andrew Stephenson. Kant's Theory of Experience. PhD thesis, University of Oxford, 2013.

[Ste15] Andrew Stephenson. Kant on the object-dependence of intuition and hallucination. *The Philosophical Quarterly*, 65(260):486–508, 2015.

[Ste17] Andrew Stephenson. Imagination and inner intuition. *Kant and the Philosophy of Mind*, 2017.

[Str18] Peter Strawson. *The Bounds of Sense*. Routledge, 2018.

[Swa16] Link R Swanson. The predictive processing paradigm has roots in Kant. *Frontiers in Systems Neuroscience*, 10:79, 2016.

[Tar67] Alfred Tarski. The completeness of elementary algebra and geometry. 1967.

[TT41] Louis Leon Thurstone and Thelma Gwinn Thurstone. Factorial studies of intelligence. *Psychometric Monographs*, 1941.

[TTL05] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the Condor experience. *Concurrency and Computation*, 17(2-4):323–356, 2005.

[UGT12] Tomer D Ullman, Noah D Goodman, and Joshua B Tenenbaum. Theory learning as stochastic search in the language of thought. *Cognitive Development*, 27(4):455–480, 2012.

[VEK76] Maarten H Van Emden and Robert A Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM (JACM)*, 23(4):733–742, 1976.

[VLH08] Michiel Van Lambalgen and Fritz Hamm. *The Proper Treatment of Events*. John Wiley & Sons, 2008.

[WACL05] John Whaley, Dzintars Avots, Michael Carbin, and Monica S Lam. Using Datalog with binary decision diagrams for program analysis. In *Asian Symposium on Programming Languages and Systems*, pages 97–118. Springer, 2005.

[Wax14] Wayne Waxman. *Kant's Anatomy of the Intelligent Mind*. Oxford University Press, 2014.

[Wit09] Ludwig Wittgenstein. *Philosophical Investigations*. John Wiley & Sons, 2009.

[Wol63] Robert Wolff. *Kant's Theory of Mental Activity*. Harvard University Press, 1963.

[Wol83] Stephen Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55(3):601, 1983.

[XLS+19] Zhenjia Xu, Zhijian Liu, Chen Sun, Kevin Murphy, William Freeman, Joshua Tennenbaum, and Jiajun Wu. Unsupervised discovery of parts, structure, and dynamics. In *Proceedings of the International Conference on Learning Representations (ICLR)*, pages 1418–1424, 2019.

[ZLS+18] Amy Zhang, Adam Lerer, Sainbayar Sukhbaatar, Rob Fergus, and Arthur Szlam. Composable planning with attributes. *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.