# Event Memory for Intelligent Agents

David H. Ménager

B.S. Computer Science, University of Kansas, 2015

M.S. Computer Science, University of Kansas, 2018

Submitted to the graduate degree program in Electrical Engineering and Computer Science and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Committee members

Arvin Agah, Chairperson

Michael Branicky, Committee Member

Prasad Kulkarni, Committee Member

Andrew Williams, Committee Member

Sarah Robins, External Committee Member

Date defended: June 01, 2021

The Dissertation Committee for David H. Ménager certifies

that this is the approved version of the following dissertation :

Event Memory for Intelligent Agents

Arvin Agah, Chairperson

Date approved: June 01, 2021

# Abstract

This dissertation presents a novel theory of event memory along with an associated computational model that embodies the claims of view which is integrated within a cognitive architecture. Event memory is a general-purpose storage for personal past experience. Literature on event memory reveals that people can remember events by both the successful retrieval of specific representations from memory and the reconstruction of events via schematic representations. Prominent philosophical theories of event memory, i.e., causal and simulationist theories, fail to capture both capabilities because of their reliance on a single representational format. Consequently, they also struggle with accounting for the full range of human event memory phenomena. In response, we propose a novel view that remedies these issues by unifying the representational commitments of the causal and simulation theories, thus making it a hybrid theory. We also describe an associated computational implementation of the proposed theory and conduct experiments showing the remembering capabilities of our system and its coverage of event memory phenomena. Lastly, we discuss our initial efforts to integrate our implemented event memory system into a cognitive architecture, and situate a tool-building agent with this extended architecture in the Minecraft domain in preparation for future event memory research.

# Acknowledgements

I would like to thank all those who guided and supported me throughout my time as a student. First and foremost, I would like to thank God for giving me the opportunity to pursue this degree and the dedication to complete the program. I would also like to thank the Self Graduate Fellowship and the GEM Fellowship program for providing me with funding, professional and personal development opportunities during the course of my studies. Thank you to my advisors Arvin Agah and Dongkyu Choi for continually supporting me and guiding my research. They have been invaluable resources. I would like to thank my committee members Michael Branicky, Prasad Kulkarni, Andrew Williams, and Sarah Robins for their support, contributions, and feedback on my work. I would also like to thank Christopher MacLellan, David Aha, Laura Hiatt, and Mak Roberts for their insightful comments and discussions over the years. Next, I would like to thank my current and former my lab mates for their support. We were strangers when we first met, but the bond of common experience has transformed us into colleagues and friends. I would like to also thank my fiancée Érica, my parents Henri and Mona, my siblings Jean and Rachel, my sister in-law Christina, and my friends for their unwavering support and encouragement. I could not have done this without them. Finally, I would like to thank Ben Gripka who was my first mentor in high school who taught me the fundamentals of programming. He gave me the tools and confidence I needed to succeed.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Event memory is a general-purpose storage for personal past experience. Literature on event memory reveals that people can remember events by both the successful retrieval of specific representations from memory and the reconstruction of events via schematic representations. In psychology, Tulving (1983) initially argued for the existence of an episodic memory responsible for storing and retrieving the specific what, when, and where of everyday life experiences. Researchers later on (Schacter & Addis, 2007a,b) began favoring a more constructive approach to remembering that responds to influences present at encoding and retrieval.

Psychological perspectives on memory typically focus on using empirical data to define the observable mental behaviors involved in remembering. Philosophical perspectives on event memory borrowed some of these insights to make arguments for the essential faculties that event memory requires, including specifying the cognitive architecture necessary for remembering along with the requisite representations that architecture manipulates to produce recollections. Event memory theories in philosophy can largely be grouped into two different categories of perspectives. The more established views, called causal theories (Martin & Deutscher, 1966), contend that the memory for events necessitates storing and maintaining representations of discrete past events, and that remembering is a causal process beginning from a past event to the obtained recollection. The more contemporary category of theory treats the memory for events as a construction system. These views, called simulationist theories (Michaelian, 2016b), deny the necessity of stored representations and favor utilizing generalized schemas for recombining and synthesizing plausible recollections.

Although each view has its own merit and strengths, neither is totally successful at explaining and accounting for all uses of event memory as revealed by memory science. Causal theories initially aimed to explain cases of success, describing what ought to happen when remembering goes right. In contrast, simulationist views arose in response to discoveries in psychology showing the ways in which memory for events can be manipulated and give rise to errors. From these disparate starting points, proponents of causal and simulation theories have tried to adapt aspects of their respective view to accommodate both the successes and failures of usage of event memory, but have achieved limited degrees of success. For example, a causal theorist may be able to describe what a wholly inaccurate recollection is in their view, but cannot explain the conditions under which the memory system would produce such an outcome. Similarly, it is unclear how a simulationist system built around generalized schemas can reliably recover or reconstruct information about a unique event since unique features that compose any one experience wash out in the generalized schema. Further complicating the debate is that neither view has an associated implementation that supports experimentation, so it is difficult to determine whether each view truly boasts the coverage which it claims.

## 1.1 Approach

Our position is that prominent philosophical theories of event memory, i.e., causal and simulation theories, fail to capture the full range of event memory usage because of their reliance on a single representational format to explain all event memory phenomena. We therefore propose a novel theory, which builds upon our previous work (Ménager & Choi, 2016; Ménager, 2016, 2018; Ménager *et al.*, 2018), that remedies this issue by unifying the representational commitments of the causal and simulation theories, thus making it a hybrid theory. We posit that event memory is a structured store which houses both discrete representations for events, as well as generalized schemas that can combine and reconstruct recollections. We further hold that retrieval cues which are operative during remembering modulate which representation is used. This behavior in turn allows our system to capture both the strengths of causal and simulationist theories, while avoiding their pitfalls.

Additionally, we conduct experiments in a simulated environment on a computational model that embodies the theoretical claims of our view. We utilize the Blocks World (Winograd, 1971) domain on a recognition task to test three hypotheses. First, that our hybrid event memory system stores discrete representations of observed exemplars and forms generalized schematic representations of the exemplar classes. Second, that the system supports remembering using both of these representations, and third, that qualities of the retrieval cue influence which representation is used during remembering. The results of the experiment suggest that our view is a novel hybrid theory which unifies both causal and simulationist theories.

Next, we conduct a second experiment to determine whether the theory covers the success and failure cases of event memory usage. To do this, we collect system recollection performance data on a recognition task in Blocks World and perform cluster analysis over the dataset to observe the produced recollection phenomena. Our clustering results suggest that our hybrid event memory system can produce cases of success and failure when attempting to remember previously seen exemplars. Then, we utilize decision tree analysis to discover what internal system parameters give rise to each phenomenal category. These experimental findings support the claim that the proposed hybrid theory provides a broader coverage than existing theories and also can account for how these phenomena are produced by the memory system.

Finally, we move to integrate our event memory system within a cognitive architecture in preparation for building event memory-enabled agents that can interact and cooperate with humans in complex environments. We show how the system fits within the larger context of the architecture, and then build an agent that exists in the popular video game Minecraft (Johnson *et al.*, 2016). We then demonstrate that the agent can build compound tools in this environment. After this, we discuss our plans for future event memory experiments we would like to conduct in this domain.

## 1.2 Contributions

Our research makes theoretical and applied contributions to the field of artificial intelligence. First, this research introduces a novel theory of event memory for computational agents that addresses the shortcomings in our knowledge about the nature of event memory and its related phenomena (Ménager *et al.*, 2021a). Specifically, our theory provides insight into the cognitive architecture that supports remembering and the necessary representations that it manipulates. Additionally, our theory produces both successes and failures of event memory usage and gives an account of the conditions under which they occur, which is a capability other similar theories have yet to demonstrate (Ménager *et al.*, 2021b).

Our theoretical contributions culminate in a psychologically plausible implementation of our event memory system. This implementation is written using a high-level computer programming language adhering to the ANSI Common Lisp language specification. Our event memory system is built to operate both as a stand-alone memory system and as a module in a cognitive architecture namely, ICARUS (Choi & Langley, 2018).

Finally, the extended ICARUS platform takes steps toward building agents that cooperate and interact with humans through an agent that plays Minecraft. We believe that with continued research, agents extended with our event memory system will be able to rely on event memory capabilities to facilitate better cooperation with humans. This in turn will speed adoption of autonomous systems in everyday life.

This dissertation is organized into five chapters. The second chapter details the commitments of our hybrid theory and presents the implementational details of our computational model, describing its episodic and schematic representations, as well as event memory processes that operate over them which include inserting new experiences and remembering past events. Additionally, we present results from experiments conducted in a simulated environment that provide empirical support that our system affords remembering via both episodes and schemas and that properties of the retrieval cue used to during remembering modulate the chosen representation.

In the third chapter, we examine event memory phenomena elicited by the remembering process. We show in experiments conducted in a simulated environment that our event memory system provides a complete coverage of the full range of event memory phenomena and additionally provide an explanation for how these phenomena are produced in the system.

Lastly, event memory is an important aspect of cognition, but only a few cognitive architectures support event memory reasoning capabilities. The fourth chapter discusses our initial efforts to integrate our hybrid event memory system into a cognitive architecture and situate an agent with this extended architecture in the Minecraft domain. Minecraft is a rich application domain that allows intelligent agents to pursue high-level goals, execute multi-step plans, and interact with other players, and thus, is a fertile ground for event memory research. We discuss our Minecraft agent and walk through a couple of demonstrations of the agent building compound tools.

We believe this work on event memory is an important advance in artificial intelligence and will continue to drive basic research in this area in the foreseeable future. The fifth and final chapter discusses the limitations of our system and the future work which includes building on our system to enable interactive, collaborative agents that can communicate about their past, infer the goals and intentions of humans, and learn from experience.

# Chapter 2

# A Hybrid Theory of Event Memory

## 2.1 Chapter Summary

People can remember past events they have experienced with varying degrees of accuracy. Amongst philosophers, there is ongoing debate about the best characterization and explanation of this ability – in particular, what the cognitive architecture looks like or what the person has to store in their mind in order for the generation of these event representations of varying accuracy to be possible. According to causal theories, recalling events is made possible by the storage of event instances. According to simulation theories, event instances are generated from stored schemas. There is evidence in favor of each view, but neither can account for the entirety of this ability. This has led many to speculate that the best account would be one that combines both instance and schematic elements in the event memory, but there are not yet any models that give a clear account of what such a hybrid theory would look like. In this chapter, we put forward a novel hybrid theory of event memory and provide an implementation of our theory in the context of a cognitive architecture. We also discuss an agent we developed using this system and its ability to remember events in the Blocks World domain.

## 2.2 Introduction

Humans have the ability to recall individual past events (Tulving, 1983; Rubin & Umanath, 2015). This appears to include both the successful retrieval of past events that one has experienced and reconstructions of such events from schematic information, the latter of which are often but not always accurate. There are two prominent philosophical approaches to event memory – causal theories (Martin & Deutscher, 1966) and simulation theories (Michaelian, 2016b). Causal theories focus on the importance of storing discrete instances of specific events, while simulation theories focus on the use of stored schemas to construct event representations. While each approach attempts to capture the entire range of event memory within its framework, neither is successful. We suggest that this difficulty stems from the reliance on a single representational format as well as the absence of detailed implementations of these theories. In this chapter, we propose a hybrid event memory that unifies the theoretical postulates of each theory and their event representations. We further argue that this novel hybrid theory allows us to explain the full range of event memory abilities. Importantly, we introduce an implemented system that embodies our theoretical proposal and demonstrate its event memory capabilities in a simulated environment. We describe this implementation in detail and discuss the initial results from our experiments.

In the next sections, we briefly review the two main philosophical theories of human event memory in the literature and introduce our novel hybrid theory. Then we describe our implementation of this theory in detail and discuss some experiments carried out in Blocks World to evaluate the system's ability to remember events. Finally, we conclude after a discussion of related work.

## 2.3 Existing Theories of Event Memory

Theorizing about memory is a focus of research throughout the cognitive sciences. Our work here focuses on philosophical accounts of remembering, which aim to unify the concept of memory and empirical evidence about remembering. This focus allows us to bracket, at least for the time being, numerous empirical accounts of event memory. Accounts of event memory in psychology and neuroscience focus primarily on characterizing the activity of our memory system(s), explaining how they work. Researchers in these areas have played a critical role in identifying and demonstrating the phenomenon of false memory - where event memories fail to be accurate. Their focus is on documenting these false memories, the conditions under which they occur and how participants experience and incorporate these states into the rest of their thoughts and actions. There is little attention given to accounts of what successful remembering requires or theoretical distinctions between kinds of false memory. Philosophical accounts of remembering, in contrast, aim to articulate what remembering requires, and then situate various memory activities as meeting or failing those requirements. They offer possible frameworks for organizing the myriad forms of event remembering well-documented by memory science. These philosophical theories have implications for implementation. That is, they place constraints on what the underlying cognitive architecture of the memory system must be. Our interest is in exploring these theories to understand their architectural commitments, and whether and how they could be improved upon to better account for the range of event memory phenomena.

Theoretical approaches to event memory in the philosophical literature are often sorted into causal and post-causal theories (Michaelian & Robins, 2018), where simulationism is the most prominent and detailed post-causal account. This labeling of the two approaches reflect the fact that the causal theory is the better-established, traditional view. Simulationism and other post-causal accounts have emerged more recently, as part of an effort to address inadequacies in the causal theory. Below we offer a brief review of each theory.

### 2.3.1 Causal Theory

The causal theory of memory is built up from the intuitive idea that remembering is a causal process. The ability to recall something now is due to what was previously learned and stored.

For event memory, the cause of our recall must be a representation of that previous event, which derives from the event itself. This is the basic idea Martin & Deutscher (1966) laid out in their paper introducing the causal theory. They argued that remembering requires an accurate representation of the past event and that the person who is remembering be the person who experienced the previous event. What distinguished their account—and made it distinctly causal—was the argument for an additional requirement: that there be a causal link between the past event and the subsequent remembering. Much of their paper was then focused on spelling out exactly what kind of causal link was needed. After all, there are many ways that the two events could be causally connected that would not be instances of remembering. They emphasize that the right kind of causal link for event memory will be one that is sustained by a representation of that event. The representation of the past event serves as the memory trace for that event—its content is roughly equivalent to what occurred during the event. In this way, the causal theory makes the storage of individual episodes essential for event memory.

Recent versions of the causal theory have updated and revised the framework in various ways, but retain the commitment to event memory as a system that stores episodic representations of past events. As Debus (2018) explains in a recent defense of the view: "it is necessary that a relevant event of information acquisition has left some 'trace', a trace which has been preserved and is causally relevant for the occurrence of the mental state or event which should count as a memory" (p. 68). Causal theorists have worked to update our understanding of the stored representation and retrieval process so as to accommodate the host of empirical evidence (Wells, 1982; Schacter & Addis, 2007b) that event memory is highly reconstructive—often recombining elements in ways that can alter or distort memory contents. For example, Debus (2007) claims that event memory representations can be systematically modified at the time of encoding while explaining perspective switches in observer memories, and Hintzman (1986) argues for a theory that reconstructs events on the fly by retrieving a number of experiences from memory that match a retrieval cue and averaging them together. As a result, most causal theorists now accept that neither stored episodes nor produced recollections need to faithfully retain all of the information from the initial event; loss of information is possible and consistent with the retained ability to remember (Bernecker, 2008). Most causal theorists, however, continue to resist adding content to the episode. Some researchers also propose changing the representational format of the trace—from a localized representation to a distributed one (Bernecker, 2010; Michaelian, 2011).

These modifications help the causal theorist to explain some instances of reconstructive remembering. However, they fail to address many of the central forms of reconstruction. Reconstructive remembering, for example, often involves the incorporation of information from other events (Suddendorf *et al.*, 2009). These additions often influence the accuracy of a memory, and they violate the causal theorists' constraint on episodic representations containing only information less than or equal to the information in the original event. In addition, the causal theorists cannot account for the possibility that reconstruction could lead to an accurate representation of the past event without any connection to the episodic memory trace. For these reasons, many theorists have shifted their efforts from revising the causal theory to looking for alternatives. We discuss the most prominent post-causal alternative, simulation theory, in the next section.

### 2.3.2 Simulation Theory

Simulation theory (Michaelian, 2016b; Michaelian & Robins, 2018) is a theory of remembering, built around an endorsement of the evidence about reconstructive remembering and a rejection of the causal condition on remembering and the causal theory more broadly. Simulation theorists view the act of remembering as a constructive process of simulation, where one builds events representations from a wide network of available information about past events. To illustrate the nature of this simulation, Michaelian (2016b) emphasizes that remembering should be seen as a form of imagination.

Like the causal theorists, simulation theorists continue to defend the idea that remembering requires accuracy. In order for one's memory of receiving a yellow bike for their eighth birthday to count as an instance of event memory, it must be the case that he or she received such a bike for that birthday. What is not necessary, however, is an episodic representation of that birthday stored in memory since the occurrence of the event. Instead, one's overall knowledge of past events, organized in terms of schematic representations of bicycles and birthday parties, etc., can be used to reconstruct a plausible representation of the past event. Without reliance on an episodic representation of the event itself, Michaelian insists instead that "the only factor that distinguishes remembering an episode from merely imagining it is that the relevant representation is produced by a properly functioning episodic construction system" (Michaelian, 2016b, p.97).

Appealing to schematic representations of events instead of episodic representations of particulars allows the simulation theorists to capture a range of event memories that were left unexplained by the causal theory. Simulation theorists can explain how instances of remembering include more information than was present in the original event and also instances where the information derives from a range of distinct sources. Cases like one's memory of the *yellow* bike at the birthday party, however, remain problematic. In such cases, Michaelian insists that remembering is possible even without the episodic representation (Michaelian, 2016a, p.118). But how is this possible? It seems highly unlikely that one could construct a memory that just happens to be accurate about details of a particular past event without storing information from that event itself. Schematic information about birthday parties could help me to build this memory, but such information would presumably provide features of the event that are common to many birthday parties—things like cake, gifts, and balloons. While it is not unheard of to receive a yellow bike, it is far from expected of a birthday party. In order to address this issue, the simulationists would need to provide a much more detailed account of the underlying cognitive architecture of the schematic event system. No such implementation has been provided.

## 2.4 Hybrid Theory of Event Memory

As discussed above, both the causal and the simulation theories are unable to capture the full range of human event memory performance, which includes both the ability to remember particular past events and the construction of possible past events on the basis of schematic information. It is critical to develop a theory that can accommodate all these aspects of human memory. We suspect that the limitations faced by causal and simulation theorists, while different, stem from the same basic problem—reliance on a single representational form to explain all of event memory. In response, we propose a new theory that retains the virtues of both the simulation and the causal theories. The core argument of this hybrid theory is that the human memory for events stores both episodic and schematic representations. More specifically, our theory posits that:

**Event memory is a long-term memory that stores both episodes and schemas.** This commitment to both representational forms illustrates the hybrid nature of the theory, incorporating the elements of causal and simulationist approaches. The event memory in our framework is a long-term memory that stores records of events experienced by the remembering agent. The contents of this event memory are stable, but incremental changes occur over time to schematic representations as new events are encountered.

**Episodes are propositional representations of specific events.** The contents of which are the agent's internal and external state descriptions, including the agent's perceptions, beliefs, goals, and intentions. Further, episodic representations are causally dependent on past events in the sense that a specific past event was operative in producing the episodic representation. In this way, they reflect the causal theory aspect of our hybrid theory.

**Schemas are first-order propositional templates with probabilistic annotations.** They summarize episodes by embedding probability distributions associated with the summarized episodic content. In so doing, schemas employ probability to represent a range of possible events. They can also summarize similar event schemas in the same manner. In this way, they reflect the simulationist aspect of our hybrid theory.

**Event memory elements are organized in hierarchies.** At the lowest level, event memory records episodes. Over this, it stores schemas that summarize the elements of the episodes at the lower level in a probabilistic manner. Event memory elements are connected by ISA links from a child node to its parent. This indicates that the child is a specialization of its parent. Hence, the event memory elements gradually become more specific at progressively lower levels in the hierarchy.

**Retrieval cues play a central role in remembering.** Our theory characterizes remembering as a response to a retrieval cue. Cues are often a subset of the agent's current observations of the world. When presented with such a cue, we claim that the goal of the event memory system is to remember an event that is at least consistent with information contained in the cue.

**Remembering an event involves performing probabilistic inference.** Given a retrieval cue, the system searches for a memory element that best matches it. Sometimes the best match will be an episode and other times the best match will be a schema. When the found match is an episode rather than a schema, it is returned as a remembered event as it provides a deterministic description of the past event. In contrast, if the match is a schema, the system performs probabilistic inference to output the most likely instance of the schema conditioned on the retrieval cue provided. This reconstructive process is inherently approximate, allowing for inaccuracies in the representation that is returned.

Our novel theory as described above unifies aspects of causal and simulation theories by employing hybrid event representations. It includes both specific episodes and generalized schemas as event memory elements. The generalization hierarchy they form can afford remembering that is consistent with the full range of human memory for events. In the next section, we describe our implementation of this new theory, beginning with the hybrid representation and continuing on to the processes that work over the event memory elements.

## 2.5 Hybrid Event Memory System

The core commitments described above dictate the computational implementation of our theory. We believe that the hierarchical organization of event memory elements, which includes specific episodes at the bottommost level and partially generalized schemas at higher levels, results in an elegant combination of causal and simulation theoretic aspects of memory storage and retrieval. In this section, we describe our implemented system in detail and discuss its implications, starting with the representation and continuing to the processes that work over it.

### 2.5.1 Event Representation and Generalization Structure

We begin our discussion of representation with some definitions that will set the foundation for the event memory structures we will discuss. In our framework, the world is composed of a potentially infinite set of *objects*, and this set can be partitioned using object classes. Such classes impose representational constraints over its members, allowing objects in a class to be described with the same set of attributes. Hence, object classes are templates for generating grounded representations, or instances, of objects. We represent objects with lists that include the object type, a unique name, and attribute-value pairs.

These objects and their attributes satisfy certain hierarchical *relations* in the world, which we define as predicates. Relations may also be partitioned into a set of relational classes. For example, we can consider an `on` relation defined with a block stacked on top of another block and a second `on`

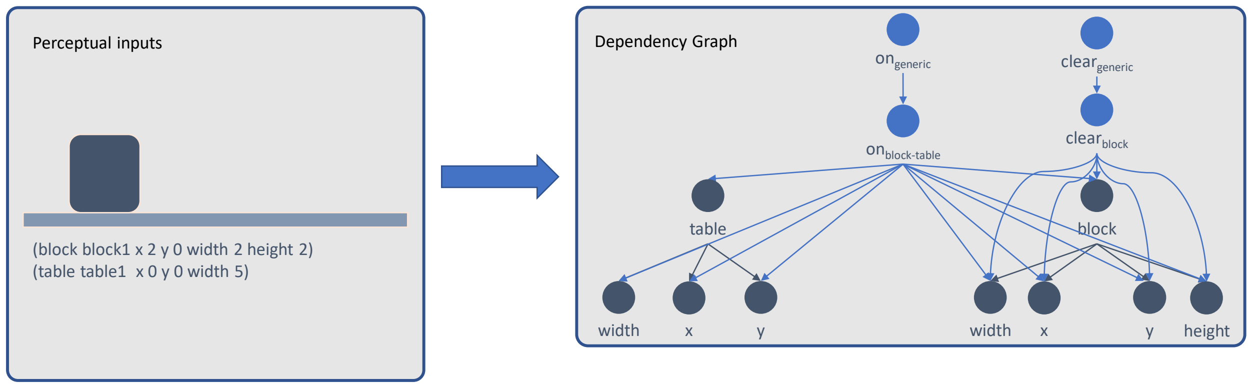

Perceptual inputs
(block block1 x 2 y 0 width 2 height 2)
(table table1 x 0 y 0 width 5)
Dependency Graph
$\text{on}_{\text{generic}}$
$\text{clear}_{\text{generic}}$
$\text{on}_{\text{block-table}}$
$\text{clear}_{\text{block}}$
table
block
width
x
y
width
x
y
height

Figure 2.1: Representing an episode as a dependency graph.

relation defined with a triangle stacked on top of a block as members of a larger class, `on`$_{\texttt{generic}}$. We represent instantiated relations, or beliefs, as lists that include the relational class and a subset of component object identifiers as arguments.

Then an *episode* in our framework is a set of grounded objects as well as the corresponding beliefs derived from them. Since beliefs are inferred from hierarchically defined relations like `on` and `next-to` based on perceived objects like `blocks` and their attributes, episodes form a *dependency graph* composed of objects, their attributes, and beliefs. Figure 3.2 shows a sample state in Blocks World and the corresponding dependency graph. The system represents a perceived object like `block1` as a tree of height 1 with the object class as the root. There are directed edges from this root to the nodes that correspond to the attributes of the object like `width`, `height`, `x`, and `y`. Each attribute node stores its perceived value in it. Based on such objects, we define relations like `on` by adding a node to the dependency graph. The outgoing edges from this node link to the component objects and their attributes over which the relation is defined. Similarly, for higher-level relations, the outgoing edges link to the relevant lower-level relations as well as component objects and their attributes. In addition, we represent relational classes as parent nodes of relations.

In an episode, the dependency graph describes the event that actually occurred, by storing a set of value assignments for relation, object, and attribute nodes. But when multiple episodes are aggregated into a schema, the resultant dependency graph stores probability distributions in these nodes that specify the joint probability of correlated variables. More formally, let the nodes in the graph be a set of $m$ random variables, $\{x_1, x_2, ..., x_m\}$, with the set of valid assignments as their domains. If we assume that these variables are organized hierarchically from top to bottom, and impose conditional independence on them, namely, for any three variables, $x_i$, $x_j$, and $x_k$, we assume $p(x_i, x_j | x_k) = p(x_i | x_k) p(x_j | x_k)$, then their joint distribution is:

$$\begin{aligned} p(x_1, x_2, ..., x_m) &= p(x_1 | x_2, x_3, ..., x_m) p(x_2 | x_3, x_4, ..., x_m) ... p(x_m) \\ &= p(x_m) \prod_{t=1}^{m-1} p(x_t | \mathbf{x}_{pa(t)}), \end{aligned} \tag{2.1}$$

where $\mathbf{x}_{pa(t)}$ are the parent nodes of $x_t$. This equation specifies a structure known as a *Bayesian network*, where $\mathbf{x}$ is the collection of state variables. The advantage of using Bayesian networks to encode event schemas stems from their systematic reliance on conditional independence assumptions. They are the key to compactly representing complex joint distributions since they reduce the number of parameters necessary to encode the full joint probability distribution. In the case of hierarchical graph structures, it ensures that the probability of a variable taking on a specific value is determined only by the variables that came immediately prior, namely, its parents. These conditional independence assumptions are made manifest by the directed edges between nodes in the network. If the $m$ nodes of a network have $O(F)$ children which can take $K$ possible values, then the number of parameters needed in the model is $O(mK^F)$, which is much less than $O(K^m)$ that would be needed if no conditional independence assumptions were made. Although this is a drastic improvement, $O(mK^F)$ is still exponential, and therefore using large Bayesian networks can be problematic. We address this issue later in Section 2.5.2.2 when we discuss inference in this setting.

In our framework, event schemas as Bayesian networks consist of trees that represent objects and relations. Object trees contain two different types of nodes. The root node $r$ denotes an object class and follows a categorical distribution parameterized by a $k$-dimensional vector encoding the probability that $r$ instantiates one of its members. Attribute nodes express conditional probabilities of taking on a specific value, given the object, $r_k$. In other words, every attribute node $a_i$ with domain $\{v_1, v_2, ..., v_K\}$ specifies $p(a_i = v_j | r = r_k)$. This is a useful result, since:

$$\begin{aligned} p(r_k | a_1, ..., a_F) &\propto p(a_1|r_k)p(a_2|r_k)...p(a_F|r_k)p(r_k) \\ &\propto p(r_k)\prod_{i=1}^{F} p(a_i|r_k). \end{aligned}$$

This means that every object the agent perceives is represented as a Naïve Bayes classifier in the dependency graph, which will capture the class-conditional correlations between object attributes and their associated objects. Importantly, the attribute nodes of Naive Bayes classifiers are assumed to be independent from each other. This, however, may not be true in the physical world. Object attributes may, in fact, be correlated with each other, but the influence may not be unidirectional like in Bayesian Networks. For example, two attributes may be correlated with each other via an undirected edge, but we do not capture these potential correlations among the variables in our current representation. This, however, is usually not an issue because Naïve Bayes classifiers tend to perform well in practice.

We represent relations as trees of height 1, where the root node, $z$, represents the relational class parameterized by a k-dimensional probability vector $\boldsymbol{w}$ following a categorical distribution. Each

Table 2.1: Notional Conditional Probability Distribution.

| | | $x$ | |
|---|---|---|---|
| $y$ | $z$ | A | B |
| A | D | 1/2 | 1/2 |
| A | E | 0 | 1 |
| B | D | 1/3 | 2/3 |
| B | E | 1 | 0 |
| C | D | 1 | 0 |
| C | E | 2/5 | 3/5 |

child node in the tree corresponds to a specific disjunction of the relation, and $\boldsymbol{w}$ describes the probability distribution of these definitions. Hence, the variable $z$, once inferred, acts as a selector for which definition to generate in the current state. When $z$ is known, an object or attribute, $x$, given $z$ follows a categorical distribution parameterized by an n-dimensional probability vector $\boldsymbol{p}_z$, which is the distribution over the values of $x$ where $z$-th disjunction is considered.

In our implementation, episodes and schemas employ conditional probability distribution (CPD) tables. Table 2.1 shows a notional CPD tabulating the conditional probability distribution of different state elements. The distribution specifies $p(x|y,z)$ defined over categorical variables $x$, $y$, and $z$. The two left-most columns enumerate the range of the independent variables, while the range of the dependent variable, $x$, is in the second part of the top row. The joint probability of variable assignments are displayed in the center part of the table. For episodes, these probabilities deterministically describe a single event, but for schemas, the distributions capture a range of possible events.

Finally, our system stores episodes and schemas in a generalization tree. As Figure 3.1 shows, the root node is a schema that summarizes all the episodes the agent has experienced. The leaf nodes in this tree are the individual episodes, and there are layers of event schemas on top of these. In this hierarchy, higher-level elements are probabilistic summaries of their children, and an edge from a node to its parent indicates that the child belongs to the kind of its parent. The representational choices we outlined so far allows our system to capture both specific episodes and their generalized schemas. In the next section, we describe various processes that work over this representation to store new events in memory and retrieve past events for remembering.

### 2.5.2 Event Memory Processes

Agents with event memory routinely store their experiences and retrieve the records during their operation. In our framework, we define two multi-step processes, episodic insertion and event

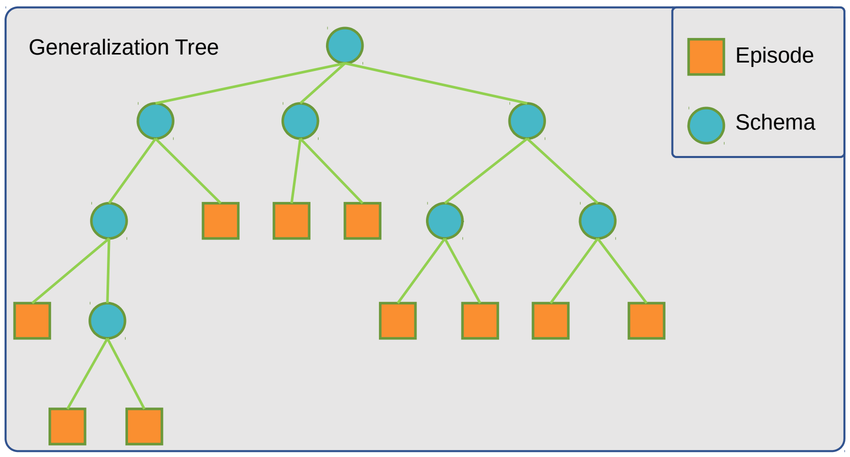


Figure 2.2: Notional generalization tree.

schematization, and retrieval and remembering, for this purpose. The insertion process introduces new episodes into the event memory. As instances are placed into the event memory, the system updates existing contents and maintains the hierarchical organization of schemas over episodes through schematization. When the system needs to remember an episode, it invokes its retrieval process to produce a remembered event. In this section, we discuss details of these processes that work over event memory contents.

#### 2.5.2.1 Episodic Insertion and Event Schematization

Insertion occurs in an online fashion, incorporating episodes as they are encountered. When the agent experiences a new event, the system attempts to find the most similar element in the generalization hierarchy. It begins from the root node and works recursively down toward the leaf nodes, as it updates probability distributions along the way. Once the system identifies the best matching element, it inserts the new episode as a child of this element. By doing this, the system incrementally clusters and classifies the episodes into event schemas that summarize them.

Table 2.2 shows this process in detail. The system sorts a new episode using a recursive level-order traversal through the hierarchy. Initially, the system matches the new episode with the root node

Table 2.2: Procedure for inserting an episode into the event memory.

```
1: procedure INSERT-EPISODE(eltm, ep)
2:     (eltm, cost-of-merging-p) $\leftarrow$ EP-MERGE(eltm, ep)
3:     if HAS-BRANCHES(eltm) then
4:         mappings $\leftarrow \emptyset$
5:         costs $\leftarrow \emptyset$
6:         best-child-cost $\leftarrow \infty$
7:         for each child branch in eltm do
8:             for each factor, $P$, in ep
9:                 and each factor, $Q$, in branch do
10:                  $\langle sol, cost \rangle \leftarrow$ STRUCTURE-MAP$(P, Q)$
11:                  costs $\leftarrow$ APPEND(cost, costs)
12:                  mappings $\leftarrow$ APPEND(sol, mappings)
13:            if SUM(costs) $<$ best-child-cost then
14:                  best-child-cost $\leftarrow$ SUM(costs)
15:                  best-child $\leftarrow$ branch
16:    if eltm is empty then
17:        eltm $\leftarrow$ LIST(ep)
18:    else if ep $=$ ROOT(eltm) then return eltm with in-
   creased count
19:    else if cost-of-merging-p $<$ best-child-cost then
20:        eltm $\leftarrow$ ADDCHILD(eltm, ep)
21:    else INSERT-EPISODE(best-child, ep)
```

in its event memory. If the root node has any children, the system sequentially checks each child to find the locally best matching node to the current episode (Line 7 – 15). If the root node is a better match to the episode than its children (Line 19), the new episode is inserted as a child of the root (Line 20). If not, one of the children nodes is a better match than the root, and the level-order traversal continues by recursively running the procedure on the best child (Line 21). In a degenerate case where the new episode and the root node of the tree are identical, the episode is absorbed into the root without continuing the traversal (Line 18).

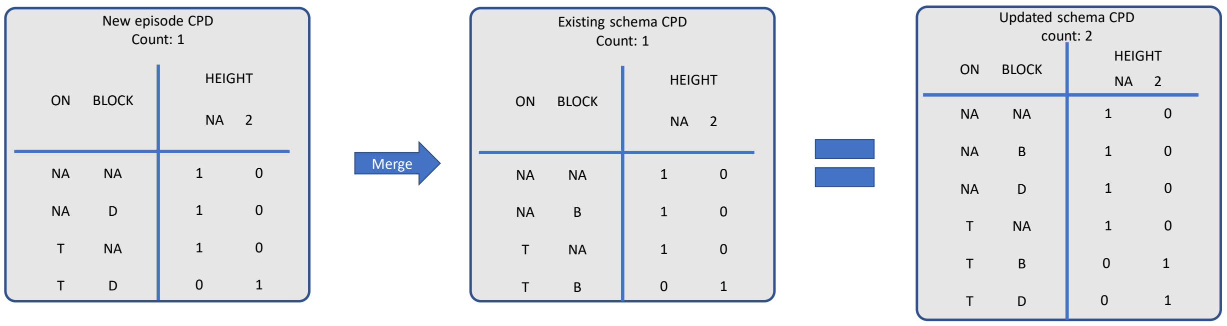
During insertion, the system attempts to match a new episode against an existing event memory element (Line 10) using structure mapping (Gentner, 1983). This allows the system to determine the similarity of the two structures and measure the quality of match between them. The system attempts to map every node in the new episode's dependency graph to a node in the dependency graph of the memory element. During this process, we impose an additional constraint that requires a matching pair of nodes to be of the same class and to have CPDs that contain matching dependent variables, but the independent variables may vary. For example, a valid match for a node from a new episode that has the CPD, $p(x|y,z)$, can only match a node from the existing element that has the CPD, $p(x|a,b)$. This not only ensures that the entity types match but also allows different relations to be defined over these entities.

The structure mapping procedure returns the lowest-cost mapping between an episode and an event memory element. In our framework, we define cost to be the probability that the event memory element does not generate the episode, namely, $(1 - score_{BIC}(Q : P))$, where $P$ and $Q$ are the dependency graphs of the new episode and the event memory element, respectively, and $score_{BIC}$ is the Bayesian Information Criterion (BIC) specified as:

$$score_{BIC}(Q : P) = \ell(\hat{\theta}_Q : P) - \frac{logM}{2} Dim[Q] \tag{2.2}$$

Here, $\ell(\hat{\theta}_Q : P)$ is the likelihood of $P$ under $Q$, $\hat{\theta}_Q$ are the parameters of the conditional probability distribution in $Q$, $M$ is the number of episodes summarized by the event memory element, and $Dim[Q]$ is the number of free parameters in $Q$ that are not in $P$. Simply put, this score computes a trade-off between how predictive the event memory element is and how structurally complex it is. When $M$ is small, $score_{BIC}$ favors more predictive event memory elements, but as $M$ grows, it prefers more parsimonious event memory elements. This is a built-in regularization that avoids overfitting while maximizing likelihood, especially as $M$ grows. This effectively applies Occam's Razor (Jefferys & Berger, 1992) to selecting a good match for the new episode, since event memory elements with high $score_{BIC}$ are the structurally simplest elements with the highest probability of generating the new episodes.

Moreover, in Bayesian networks, the BIC score decomposes to a series of local computations over


New episode CPD
Count: 1

| ON | BLOCK | HEIGHT NA | HEIGHT 2 |
|---|---|---|---|
| NA | NA | 1 | 0 |
| NA | D | 1 | 0 |
| T | NA | 1 | 0 |
| T | D | 0 | 1 |

Existing schema CPD
Count: 1

| ON | BLOCK | HEIGHT NA | HEIGHT 2 |
|---|---|---|---|
| NA | NA | 1 | 0 |
| NA | B | 1 | 0 |
| T | NA | 1 | 0 |
| T | B | 0 | 1 |

Updated schema CPD
count: 2

| ON | BLOCK | HEIGHT NA | HEIGHT 2 |
|---|---|---|---|
| NA | NA | 1 | 0 |
| NA | B | 1 | 0 |
| NA | D | 1 | 0 |
| T | NA | 1 | 0 |
| T | B | 0 | 1 |
| T | D | 0 | 1 |

Figure 2.3: Updating event schema with merge operation.

the CPDs themselves (Koller & Friedman, 2009). This means that we can compute the BIC score of the entire network during the structure mapping process. Using the decomposed BIC score, we can reformulate our cost function as:

$$\sum_{i=1}^{N} height(p_i) \cdot (1 - score_{BIC}(q_i : p_i)), \tag{2.3}$$

where $N$ is the number of nodes in the episode's dependency graph, $p_i$ is the i-th node in that graph, $height(p_i)$ is the height of the i-th node in the episode's dependency graph, and $q_i$ is the corresponding match to $p_i$ in the event memory element. Considering $height(p_i)$ in the cost function forces the structure mapping procedure to prioritize higher-level matches in the dependency graph and prefer to match more general schemas given all the other conditions are equal. When $p_i$ is matched, the $score_{BIC}$ dominates the cost. In contrast, when $p_i$ does not have any corresponding node in $Q$, then the height of the unmatched node determines the cost of match. As a result, our system prefers to match states that are qualitatively similar at a higher level. For example, the system will prefer to match an episode that involves two blocks, $A$ and $B$ in an `on` relation to another episode with two blocks, $C$ and $D$ with the same relation, instead of matching to a third episode with two blocks $A$ and $B$ in a `next-to` relation. Although the first and the third episodes contain the same blocks, their higher-level relations are different. Therefore, the cost of matching these two episodes will be higher than matching the first episode to the second.

Our system interleaves the insertion process we described so far with event schematization, which uses the results of structure mapping to merge two event memory elements into a schema. The system performs normalized factor addition by filtering the nodes from the episodic dependency graph with their associated matches. When the matching element from the event memory is also an episode, the merging operation will yield a new schema. In cases where the existing event memory element is already a schema, the merging operation will update the distributions of that schema. It is important to note that the CPDs from the new episode and their counterparts in the matched element are not necessarily defined over the same domains. For example, the episode might contain a new predicate like `(tower C D)` not previously modeled in the matching schema. Therefore, before the merging operation takes place, the system temporarily renames the variables in the episode CPDs with the names of their corresponding matches. This enables the system to easily identify which variables to update and to what extent. Once the CPD variables from both sides are made directly comparable in this manner, the system proceeds to update the domains and the variable assignments.

Then the system uses a filtering procedure that passes an episode CPD through a schema CPD. Figure 2.3 shows an example, where the system updates the schema distribution for $p(height|on, block)$ with evidence contained in a new episode. Note that the `block` variables in the two CPDs have different domains. Namely, the episode describes a block, `D`, while the schema describes another block, `B`. In order to merge these two CPDs properly, the system expands the domain of the `block` variable in the schema and puts placeholders for the additional value, `D`, for this variable, while it similarly adds placeholders for `B` in the episode. Then, the merging operation simply performs element-wise addition over these expanded structures to update the distribution for $p(height|on, block)$.

Table 2.3 shows a pseudocode for this element-wise addition process. The system constructs a new CPD $\psi$ with the expanded domain (Line 2 – 3). To determine the merged CPD's variable assignments, it takes the sum of the counts of each variable assignment in the schema and episode (Line 6 – 11). Then, the system calculates the total counts (Line 13) and the probabilities (Line 14)

Table 2.3: Procedure for filtering one CPD with another comparable CPD.

```
1: procedure FACTOR-FILTER($\phi_1$, $\phi_2$, op)
2:     $\psi \leftarrow$ MAKE-CPD($\phi_1$, $\phi_2$)
3:     $\psi$-assignments $\leftarrow$ MAKE-ARRAY(num-assignments)
4:     j $\leftarrow$ 0
5:     k $\leftarrow$ 0
6:     for i from 0 to num-assignments - 1 do
7:         if op = "+" then
8:             $\psi$-assignments[i] $\leftarrow$ (CPD-ASSIGNMENTS($\phi$1)[j] $*$ CPD-COUNT($\phi$1)) $+$ (CPD-ASSIGNMENTS($\phi$2)[k] $*$ CPD-COUNT($\phi$2))
9:         else if op = "*" then
10:            $\psi$-assignments[i] $\leftarrow$ CPD-ASSIGNMENTS($\phi$1)[j] $*$ CPD-ASSIGNMENTS($\phi$2)[k]
11:        (j, k) $\leftarrow$ ADVANCE-INDICIES($\phi_1$, $\phi_2$, j, k)
12:    if op = "+" then
13:        count $\leftarrow$ CPD-COUNT($\phi$1) $+$ CPD-COUNT($\phi$2)
14:        normalized $\leftarrow$ {val/count : val in $\psi$-assignments}
15:    else if op = "*" then
16:        count $\leftarrow$ CPD-COUNT($\phi$1)
17:        normalized $\leftarrow$ NORMALIZE($\psi$-assignments)
18:    cpd-assignments($\psi$) $\leftarrow$ normalized
19:    return $\psi$
```

of the variable assignments. This update procedure ensures that the system maintains the correct empirical probability distributions for CPDs because it preserves the count information. In cases when a variable from a dependency graph does not have a corresponding match, it inherits the domains of the parent node variables because those variables may have matched counterparts. Doing this ensures that the variable domains remain consistent throughout the episode or schema.

As outlined above, the system inserts new episodes into the best matching location in its memory through structure mapping and uses its merging process to incrementally schematize the event memory elements. This results in a spectrum ranging from individual episodes at the lowest level to progressively more general schemas at higher levels. Our system retrieves elements from this hierarchical structure and produces remembered events in response to retrieval cues. In the next section, we describe these processes in detail.

#### 2.5.2.2 Retrieval and Remembering

When an agent with event memory operates in the world, it stores experienced episodes and forms the episodic generalization hierarchy as described in the previous section. At any given point, the agent might need to remember a past event, and the system invokes a retrieval process to produce a remembered event. Our event memory system performs retrieval in a level-order fashion using a retrieval cue. Depending on the situation, a retrieval cue may be a fully observed state, or only a partial description of an event. The system tries to find an episode or a schema in its memory that best matches the given retrieval cue. This search involves the same machinery as what we use for episodic insertion, except that the system does not update the probability distributions of the existing event memory elements using the contents of the retrieval cue.

If the search yields an episode as the best match to the retrieval cue, the event memory system returns that episode immediately, since the episode is a deterministic representation of a specific event to be remembered. In contrast, if the system finds an event schema as the best match, it needs to further process the schema to return the most likely instantiation as the remembered event. The process involves performing probabilistic inference over the probability distributions contained in the schema. The inference step discovers the values of the unobserved state elements of the schema using the retrieval cue as a set of observed state elements. We describe this process more formally as in:

$$P(\mathbf{x}_h|\mathbf{x}_v,\theta) = \frac{P(\mathbf{x}_h,\mathbf{x}_v|\theta)}{P(\mathbf{x}_v|\theta)} \tag{2.4}$$

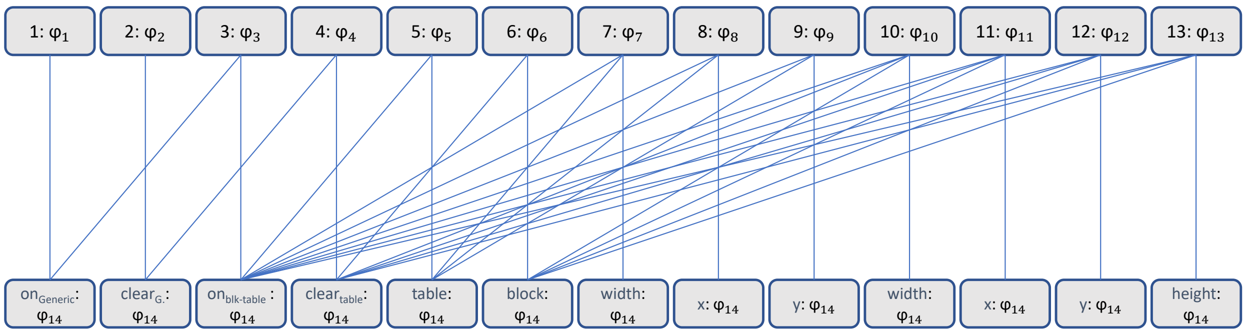
where $\mathbf{x}_v$ are the observed state elements, $\mathbf{x}_h$ are the unobserved state elements, and $\theta$ are the parameters of the retrieved schema. This equation shows that the system can divide the joint distribution of all the schema variables by the probability of the retrieval cue to infer the posterior distribution of unobserved variables. But computing the exact posterior distribution is not trivial, and computing the probability of the retrieval cue can involve intractable integrals if some of the

Table 2.4: Procedure for performing probabilistic inference.

```
procedure CALIBRATE-FACTOR-GRAPH(factors, op, edges, evidence)
    messages ← INITIALIZE-GRAPH(edges, evidence)
    repeat
        calibrated ← true
        for i from 0 to LENGTH(edges) - 1 do
            j ← edges[0]
            k ← edges[1]
            sepset ← CPD-IDENTIFIERS(factors[j]) ∪ CPD-IDENTIFIERS(factors[k])
            current-message ← messages[j][k]
            new-message ← SEND-MESSAGE(j, k, factors, op, edges, message, sepset)
            messages[j][k] ← new-message
            if new-message $\neq$ current-messgae then
                calibrated ← nil
    until calibrated
    if op = "+" then
        for i from 0 to LENGTH(factors) - 1 do
            collect COMPUTE-BELIEF(i, factors, edges, messages)
    else if op = "max" then
        constraints ← nil
        for i from 0 to LENGTH(factors) do
            constraint ← COMPUTE-BELIEF(i, factors, edges, messages)
            constraint ← THRESHOLD-ON-MAX(constraint)
            constraints ← ADD-CONSTRAINT(constraints, constraint)
        CONSTRAINT-SATISFACTION-PROBLEM-SOLVER(constraints)
```

schema variables are continuous. Even in cases where all the variables are discrete, computing the posterior distribution takes an exponential time to the tree-width of the Bayesian network (Murphy, 2012).

To remedy this, we turned to an approximate class of inference strategies called *variational inference* (Murphy, 2012). This kind of inference strategy is appropriate whenever the true joint distribution, $p^*(\mathbf{x})$, of variables is complex, but can be approximated by a simpler, variational family of distributions, $q(\mathbf{x})$, which can be made close to $p^*(\mathbf{x})$ by minimizing the KL divergence between the two. Hence, using a variational inference strategy turns the inference problem into



| | | | | | | |
|---|---|---|---|---|---|---|
| $\varphi_1 = p(\text{on}_{\text{Generic}})$ | $\varphi_2 = p(\text{clear}_{\text{Generic}})$ | $\varphi_3 = p(\text{on}_{\text{block-table}} \mid \text{on}_{\text{Generic}})$ | $\varphi_4 = p(\text{clear}_{\text{table}} \mid \text{clear}_{\text{Generic}})$ | $\varphi_5 = p(\text{table} \mid \text{on}_{\text{block-table}})$ | $\varphi_6 = p(\text{block} \mid \text{clear}_{\text{table}})$ | $\varphi_7 = p(\text{width} \mid \text{table}, \text{on}_{\text{block-table}})$ |
| $\varphi_8 = p(\text{x} \mid \text{table}, \text{on}_{\text{block-table}})$ | $\varphi_9 = p(\text{y} \mid \text{table}, \text{on}_{\text{block-table}})$ | $\varphi_{10} = p(\text{width} \mid \text{block}, \text{on}_{\text{block-table}}, \text{clear}_{\text{table}})$ | $\varphi_{11} = p(\text{x} \mid \text{block}, \text{on}_{\text{block-table}}, \text{clear}_{\text{table}})$ | $\varphi_{12} = p(\text{y} \mid \text{block}, \text{on}_{\text{block-table}}, \text{clear}_{\text{table}})$ | $\varphi_{13} = p(\text{height} \mid \text{block}, \text{on}_{\text{block-table}}, \text{clear}_{\text{table}})$ | $\varphi_{14} = \mathbf{1}$ |

Figure 2.4: Bethe cluster graph for Bayesian network in Figure 1.

an optimization problem, for which many efficient strategies exist. In our context, $p^*(\mathbf{x})$ is the distribution captured in the schema, and we chose the variational distribution, $q(\mathbf{x})$, to be the distribution represented by a Bethe cluster graph (Koller & Friedman, 2009). As shown in Figure 2.4, a Bethe cluster graph is a bipartite graph that has the CPDs of the Bayesian network as its first set of nodes, and the individual variables in the Bayesian network as the second set of nodes. Each node in the Bethe cluster graph is initially assigned a *factor*, $\phi_i$. For the nodes in the first set, their factors are the corresponding CPDs, while the nodes in the second set have the initial factors of all 1's. This ensures that all of the variable's outcomes are equally likely at the outset. But the variables corresponding to the observations given in a retrieval cure are set to the observed values. Finally, an edge exists between a node $i$ from the set of variables, to a node $j$ from the set of CPDs, only when the variable $i$ participates in the CPD $j$.

Given the initialized Bethe cluster graph, we use a message passing procedure, loopy belief propagation (Koller & Friedman, 2009), to perform probabilistic inference over this structure. Table 2.4 outlines this process. It begins on Line 2 by setting the values of observed variables in the cluster graph. The body of the procedure contains a loop, in which messages are passed from cluster to cluster. The loop continues until the messages converge. At that point, two adjacent clusters will contain the same marginal distribution for variables they have in common, so the loop exits. Following this, the system can either return a list of the marginal probabilities of all variables as shown in Line 17 or return the most probable state based on the posterior marginals as shown on Line 21. We are interested in remembering which corresponds to the latter. In this case, the inference procedure produces a list of constraint rules that are fed into a constraint satisfaction problem solver to uncover the specific values of the variables.

Running probabilistic inference procedures over Bethe cluster graphs provides us a more efficient inference process than those running directly over Bayesian networks. But this advantage comes at a price. This is due to the fact that messages from one node in the cluster graph to another can only carry information about one variable. For example, for *node1* to send a message to *node2* in Figure 2.4, it must marginalize out information about every variable except for the variable in *node2*. For this reason, the joint behavior of variables is not considered during the inference procedure. This can lead to errors when our event memory system attempts to remember events. This is an unavoidable consequence of approximate inference. We accept this consequence, since this strategy makes inference over Bayesian networks tractable.

We have now covered our system in detail, describing its theoretical commitments, its representations, and processes for storing and remembering events. In the next section, we will turn our attention to testing and evaluating it in a simulated domain. We describe our experimental objectives and present the application domain. Then, we describe our experimental procedure and present our results.

## 2.6 Experimental Analysis

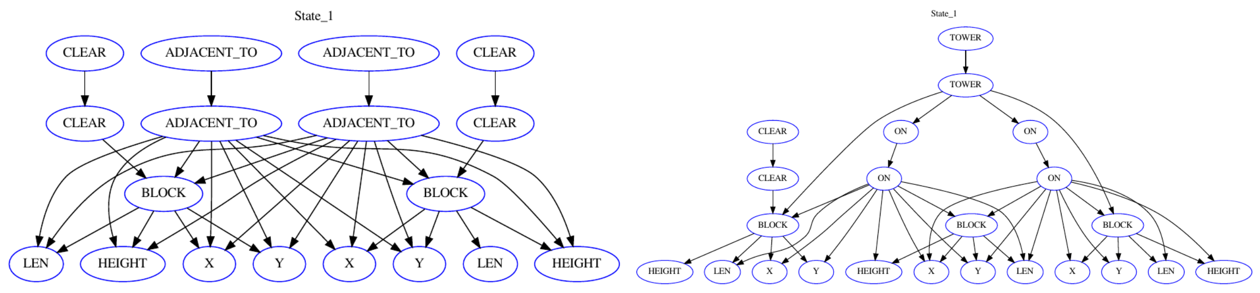
As described so far, our novel event memory system uses hierarchically organized elements annotated with probability distributions, to store both specific episodes and event schemas in its generalization tree. The system uses event memory processes that work over this representation to insert and schematize new episodes, and retrieve episodes from the event memory when needed. We argue that the hierarchy in our system affords a level of representational flexibility that neither the causal nor simulation theoretic perspectives capture. We hypothesize that: 1) our system stores and categorizes events in qualitatively distinct schemas; 2) the memory supports event remembering using both episodic and schematic representation; and 3) the retrieval cue dictates which representation the system will use for remembering. To verify these, we designed two experiments in a version of Blocks World. In the first experiment, the agent observes situations with several blocks of different sizes, infers hierarchical relations based on the sensory input, and store the inferred states as events. Then, in the second experiment, we took a typical event generalization hierarchy from the first experiment and asked the system to retrieve stored events from this hierarchy using partial states as retrieval cues.

More specifically, we first generated 50 sequences of 50 states, randomly sampled from a distribution of two state classes, `adjacent` and `tower`, with 50% probability for each class. As Figure 2.5 shows, the two classes represent two distinct kinds of situations in the world. The former describes states where two blocks are next to each other, and it has a relatively flat dependency structure that includes `clear` and `adjacent_to` relations over perceived objects and their attributes. In contrast, the latter includes states where three blocks are stacked vertically, and it has a taller structure that includes `clear`, `on`, and `tower` relations. In both cases, the blocks vary in their lengths, heights, and positions in x and y directions. For the first experiment, we presented each of these 50 state sequences to the system, which incrementally built its event generalization hierarchy from its observations. We then inspected the 50 hierarchies our system built and analyzed their structure and contents.

Out of these 50 generalization hierarchies, we chose one typical example for our second experiment. We gave this to the system and re-supplied the original 50 states used to build this hierarchy as retrieval cues at different levels of completeness, to see if the system can successfully retrieve and remember the original events. Our experimental procedure consisted of ten epochs, with the completeness of the retrieval cues gradually decreasing from 100% to 10%. Namely, the first epoch uses each of the full states as a retrieval cue, the second epoch uses 90% of each state, and the percentage goes all the way down to 10% of each state for the last epoch. Every epoch included one trial for each set of the original 50 states, in which we presented 20 different partial states of the original state as retrieval cues for remembering. We used partial states that we generated by randomly removing the amount of nodes from the original states that corresponds to the epoch. We recorded several different performance measures during the 10,000 runs. These measures include: the percentage of original state elements provided as retrieval cues, the total number of nodes in the retrieval cues, whether the system retrieved an episode or a schema for remembering, the match distance between the retrieval cue and the retrieved element, and so forth. In the following sections, we analyze our results from the two experiments to verify each of our hypotheses presented above.

### 2.6.1 Storing and Schematizing Episodes

Our first hypothesis is that our system can store and categorize events in qualitatively distinct schemas. Since it is not practical to study the qualitative meaning of every single schema from all 50 cases, we instead checked the overall structures of the generalization trees and compared the probability distributions of siblings at a few different levels. Each of the 50 randomly generated



(a) Dependency graph for states that belong to `adjacent` class.

(b) Dependency graph for states that belong to `tower` class.

Figure 2.5: Dependency graph for the two event classes used for our experiment.

sequences of events resulted in a generalization hierarchy like the one shown in Figure 2.6 that includes observed episodes at the leaf nodes (red boxes) and several layers of event schemas (blue boxes) over them. The two highest-level schemas under the root node are a generic schema for `adjacent` class of episodes (schema (1) in the figure) and another (2) for `tower` class of episodes, respectively. Further, we found a binary sub-tree under the former, which represents different sub-classes of `adjacent` events. The sub-tree includes three to four levels of schemas and then episodes at the leaf nodes.

Given such a structure, an important question then is whether or not the schemas our system generates group episodes and lower-level schemas properly according to the observed state's qualitative aspects. For this, we chose some schemas from some sample sub-trees and inspected them in detail. We found that there exist a few dominant variables that distinguish the sibling schemas at each level. Figure 2.7a shows how the two schemas, (5) and (6), differ in their distributions. The probabilities for the first schema's variables are plotted in blue, whereas those for the second schema's variables are plotted in red. The result shows that most of the variables have purple-colored plots, implying that the two schemas have very similar distributions. However, a small number of variables, including $x_0$, $x_1$, and $height_0$, shows noticeable red or blue colors where the two schemas differ from each other. This means that these two schemas, (5) and (6), are distinguishable by the x locations of `block0` and `block1` and the height of `block0`. Namely, one schema represents cases where the blocks are located closer to the origin and the first block is shorter, compared to the other schema where the blocks are farther out on the x axis and the first block is taller. The two siblings schemas under schema (4) also exhibited similar characteristics, being distinguishable by the x locations, heights, and lengths of `block0` and `block1` as shown in Figure 2.7b. This shows that our system indeed categorizes events in qualitatively distinct, hierarchical schemas while storing them in its event memory.

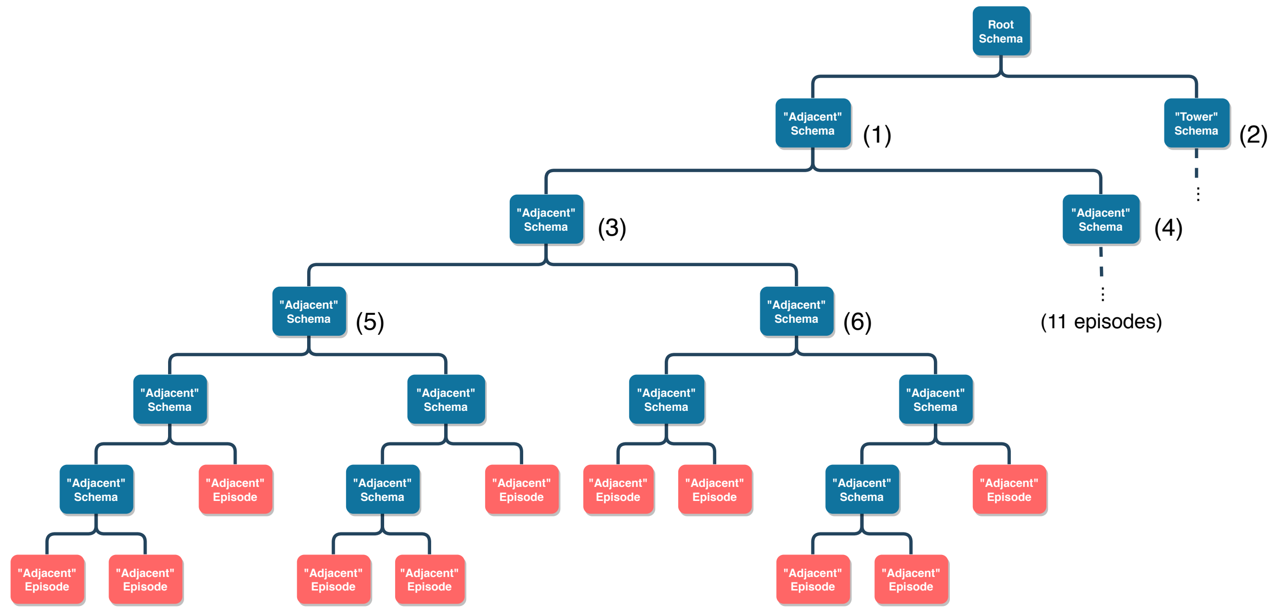
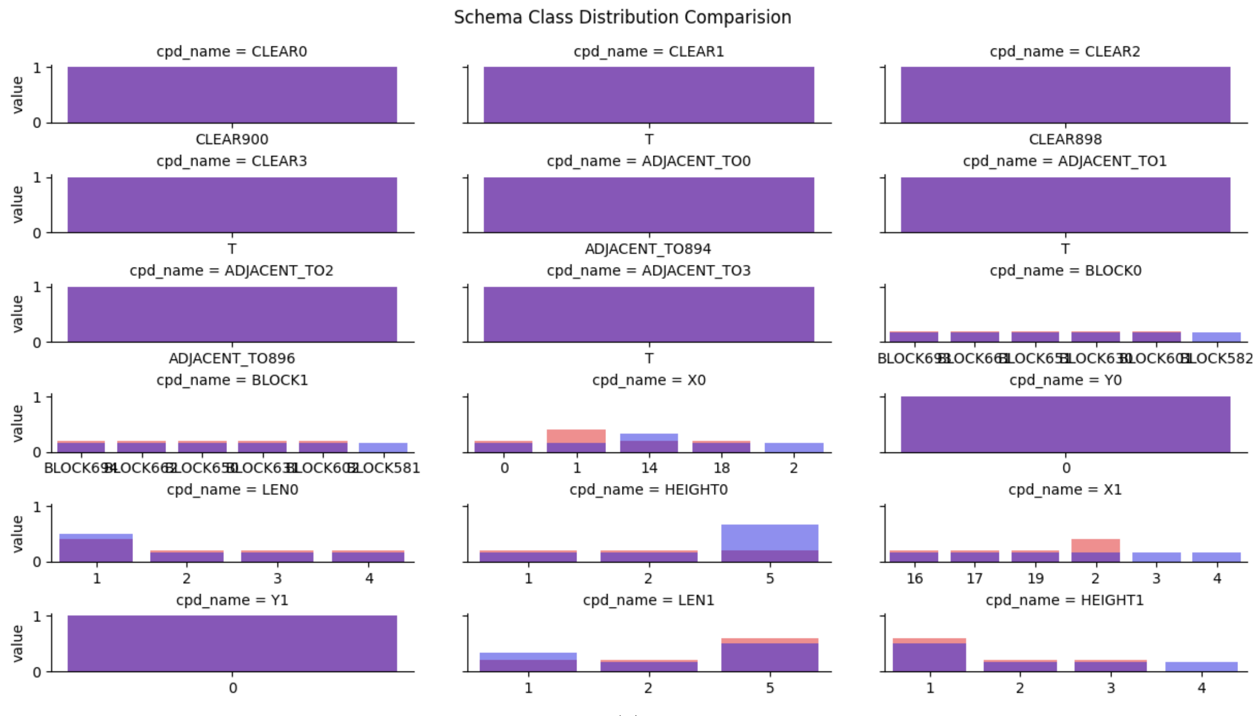


Figure 2.6: A typical generalization hierarchy after observing a mixed sequence of `adjacent` and `tower` classes totaling to 50 states.

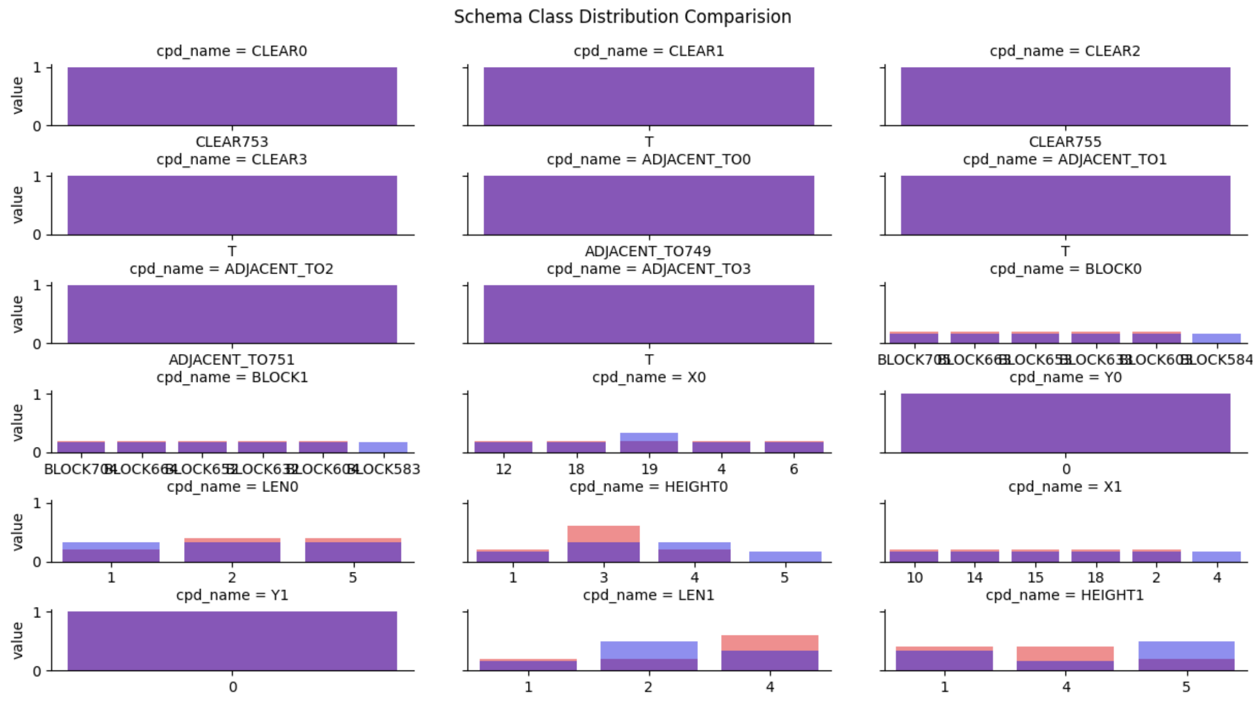
### 2.6.2 Remembering using Episodes and Schemas

Based on the evidence that our system stores and categorizes events into schemas properly, we then proceed to verify the two remaining hypotheses related to remembering. We believe that our system can remember events using either episodic or schematic elements in memory depending on situations. Further, we suspect that there is a parameter that dictates which of these representations the system will retrieve for remembering, and argue that some aspects of the retrieval cue play an important role for this.

To verify these hypotheses, we first studied the type of event memory elements that our system uses during the second stage of our experiments. Figure 2.8a shows the percentages of episode usage and schema usage in each of the epochs during our remembering experiment. The plot at the top depicts the percentages of episodes (orange) and schemas (blue) used to remember `adjacent` events, whereas the plot at the bottom shows those used to remember `tower` events. In the epochs



(a)



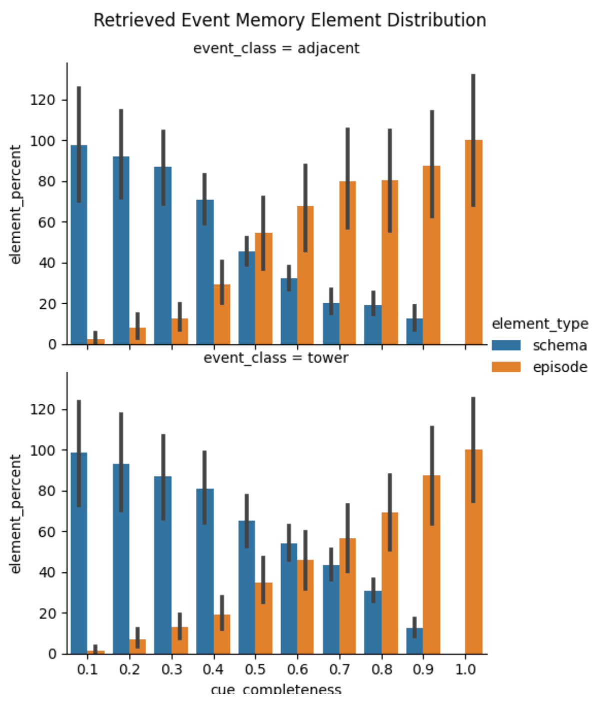
(b)

Figure 2.7: Comparison of marginal probabilities of two second-level sub-schemas of `adjacent` class.

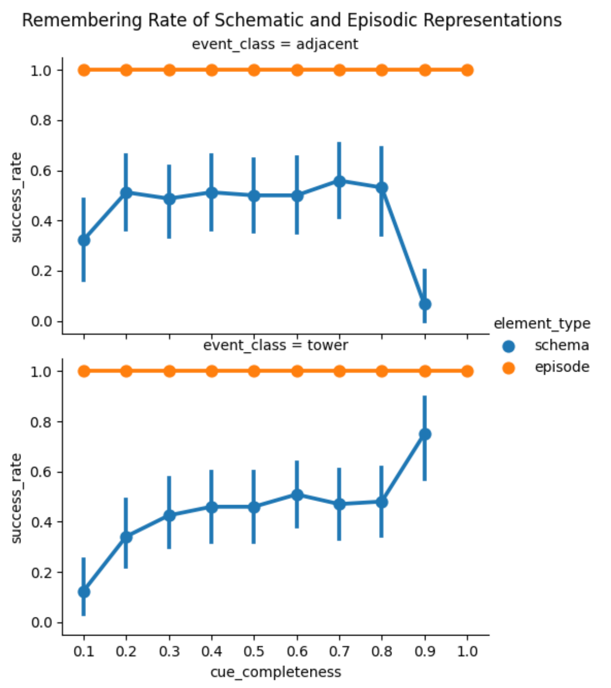
where the completeness of retrieval cue is low, the system relies mainly on event schemas to produce remembered events, but it uses more episodes than schemas as the retrieval cues become more complete. In other words, the system prefers schemas when the cue provides only a small amount of information, but it switches to episodes when the cue gives more information. For `adjacent` class of events, this switch happens near 50% cue completeness, while it occurs around 63% for the `tower` class of events. We believe that the switch from schemas to episodes for more complete cues is the right strategy to remember properly. On one hand, it is reasonable to fill in the details from the aggregated previous experience reflected in event schemas, when the retrieval cue gives very little information. On the other hand, it makes sense to retrieve an episode that matches situational details, when the cue provides more complete picture of the state to be remembered.

It is important to note that retrieval of an element does not automatically mean that the system is able to remember. In case of episodic retrieval, an element returned from event memory is indeed what the system returns as a remembered event. However, in schematic retrieval, the system needs to generate a specific instance from this schema to produce a remembered event. For this reason, we also measured the rates of successful remembering when the system is using episodes and schemas, respectively. As expected, Figure 2.8b shows that the system is always able to produce a remembered event when it retrieves an episode (orange) for both `adjacent` and `tower` classes. The episodes stored in event memory are full descriptions of events and the system can simply return them when its retrieval process deems these to be the best match.

In contrast, schemas require probabilistic inference based on the retrieval cue to produce a remembered event, and the rate of successful remembering is less than 100%. As shown with blue color in the figure, the system is able to remember an event with a retrieved schema about half of the time, except for cases when the retrieval cues are too vague or too specific. Indeed, the system behavior is very stable in the range from 20% to 80% cue completeness. This implies that the system can remember events in a reliable manner not only when using episodes, but also when using schemas. Further, we believe that the rates of successful remembering in the latter case are about 50%, not



(a) Retrieved event memory element distribution.



(b) Remembering success rate of retrieved element type.

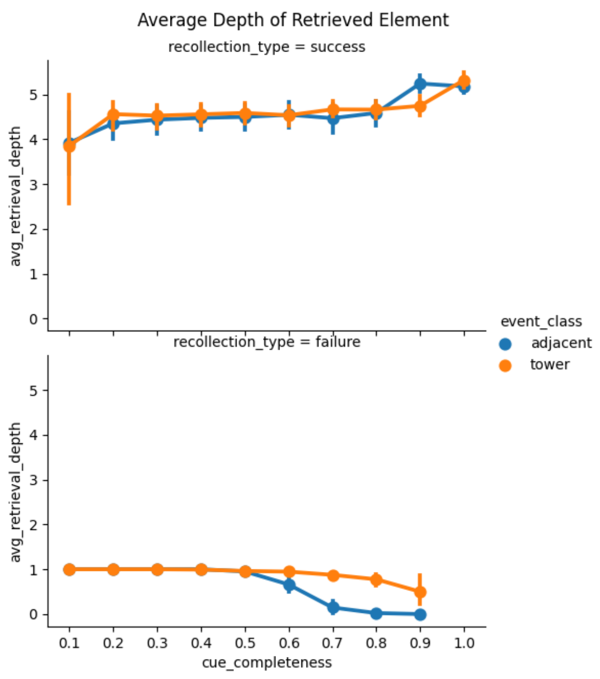
Figure 2.8: The system remembers events using both episodic and schematic representations.

higher, because we enforce a complete constraint satisfaction during remembering. In this default setup, we require the system to return failure when the probability distributions in event schemas do not yield a consistent set of assignments for the unobserved variables, namely, the variables not mentioned in the retrieval cue. Later, when we relax this condition, we expect the system to be able to remember events more often. This will be at the expense of the system sometimes producing remembered events that are not exact and fully consistent, but we know that humans often do the same and, in fact, this is central to our planned approach for modeling human memory errors.

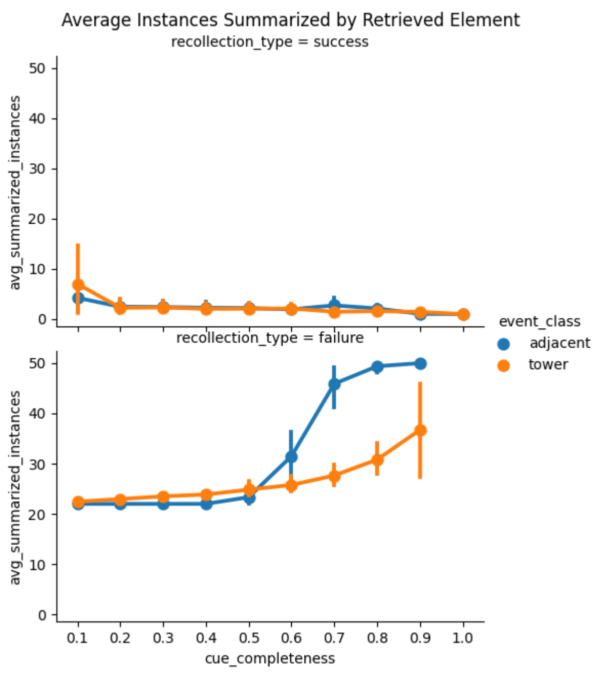
In the extreme cases where the completeness of retrieval cue is very low, we observed that the rate of successful remembering when using a schema harbors lower at around 33% for the `adjacent` class and around 12% for the `tower` class. This is reasonable because the retrieval cues provide very little information in this region and the system needs to produce a large, consistent set of variable assignments from probability distributions which is less likely to be successful due to the high uncertainty. The episodic retrieval is still successful all the time in this region, but this is not very meaningful since the system rarely uses episodes here anyway. In the other extreme cases where the completeness of retrieval cue is very high, we found that the two classes yield opposite system behavior. At 90% completeness, the rate of successful schematic remembering for the `adjacent` class is much lower than the norm, whereas that for the `tower` class is higher. We are still investigating this interesting behavior, but we suspect that the more vertical dependency graph of the latter class might be providing more useful information for successful remembering than the wider dependency graph of the former. Finally, at 100% cue completeness, the system depends entirely on episodes and does not use schemas for remembering as we suspected.

In addition, we also looked at the average depth in the hierarchy where the system's retrieval occurs for remembering and the average number of instances those retrieved elements summarize. Figure 2.9a shows the average depth from which the system retrieved event memory elements when the system succeeds in producing a remembered event through probabilistic inference (top graph) and when it fails to do so (bottom graph). For both `tower` and `adjacent` classes, the system was successful in remembering when it used schemas close to the episodes at the leaf nodes (with depths around 4 to 5), whereas the system mostly failed when it chose schemas near the top of the hierarchy. Figure 2.9b also shows a consistent result, where the smaller number of instances summarized by the retrieved schemas in successful remembering cases (top graph) and the higher number of instances in failed cases (bottom graph). As before, we believe that the strict constraint satisfaction we required in this setup played a part in this phenomenon. Furthermore, we attribute the lack of intermediate-level schemas retrieved for remembering to the two event classes being mostly distinct and not sharing many component objects and relations.

In summary, we found evidence that verifies our three hypotheses about the hybrid event memory system we developed. The system is capable of storing events in memory as specific episodes and generalized schemas at the same time, and it can retrieve either of these two to produce a remembered event. We also found that the completeness of retrieval cues modulates the system's



(a) Average depth of retrieved elements



(b) Summarized instances of retrieved elements.

Figure 2.9: Characteristics of retrieved elements for remembering.

behavior as to which representation to use for remembering. These results suggest that our system is a reasonable implementation of the hybrid event memory theory we described earlier, and that it provides a powerful computational framework for us to model various human memory phenomena. In the next section, we review some of the previous work in related directions to position our work in the proper context.

## 2.7 Related Work

As discussed in the sections above, we believe that our hybrid theory of event memory provides a novel and elegant unification between two main philosophical theories of human memory. The system we developed based on this hybrid theory can serve as a computational framework to explain various human event memory phenomena, as we hinted in our earlier review of existing theories. We trust, for the moment, that the review adequately positioned our current work in the philosophical literature. We plan to explore in a separate work how our system relates to human memory phenomena, as extensively studied in cognitive psychology and neuroscience. There, it will be of particular interest to explore how our implementation would relate to current accounts of human event memory, for example (Eichenbaum, 2017; Yonelinas *et al.*, 2019; Moscovitch *et al.*, 2016). In this section, we instead focus on previous work that are directly related to the technical aspects of our computational theory.

Our system's ability to generate schemas that aggregate numerous episodes is similar in spirit to research on concept formation, where scientists use a wide array of techniques to acquire knowledge from examples. In machine learning, there are methods to learn new concepts from examples included in a dataset (e.g., Murphy, 2012). But these typically separate the learning task from the performance task. In other words, the task of learning the domain knowledge is separated from making predictions using the learned knowledge. This is because all the examples used for forming concepts must be present at the outset of learning. Hence, this methodology can be problematic especially for cognitive systems that must learn and operate over extended periods of time. This suggests that incremental methods for concept formation, which gradually form concepts while observing new examples by interleaving the performance and learning tasks, are more appropriate for cognitive systems.

Some of the earliest incremental concept formation systems include CYRUS (Kolodner, 1983), COBWEB (Fisher, 1987), and UNIMEM (Lebowitz, 1987). These systems represent events as fixed sets of attribute-value pairs. They organize examples into hierarchies with specific instances at the bottom and generalized events at the top. Another early concept formation system, MERGE (Wasserman, 1985), extends the representation of events and decomposes them using a *fundamental* relation. Our system is similar to these systems in organizing events in a hierarchical manner, but it represents episodes as a set of predicates unlike the first three. Further, in contrast to MERGE, our system does not impose a limit on the number of relations present in the state and naturally handles cases where no fundamental relation is identified.

Among these early systems, COBWEB stands out due to its use of probabilistic concepts, which our system also features. COBWEB uses category utility (Gluck & Corter, 1985) as a heuristic for guiding search in the concept hierarchy. This enables the system to construct concepts whose member instances have high intra-class similarity. But it also causes problems in model fit due to the fact that the heuristic has a natural tendency to form categories that capture spurious relationships in the data. In contrast, our system avoids this problem by using the Bayesian Information Criterion as its heuristic evaluation function. As discussed in Section 2.5.2.1, our system considers the predictiveness and complexity of the model simultaneously. The former measures the goodness of fit of the given instance, while the latter acts as a built-in regularization to avoid overfitting.

There are a number of systems built using the COBWEB infrastructure. CLASSIT (Gennari *et al.*, 1989) substitutes categorical attributes with real-valued ones and generalizes the category utility heuristic into continuous domains, although it still suffers the problem of overfitting like COBWEB. Our system is not currently capable of handling continuous probability distributions, but we plan to extend it to use hybrid Bayesian networks (Koller & Friedman, 2009) and allow inference with both continuous and categorical variables.

Another system using COBWEB as its basis is LABYRINTH (Thompson & Langley, 1991). While most of the early systems could not handle structured information, this system is capable of doing so by extending its representational language for concepts. The system uses hierarchical concepts over primitive, ordered set of attribute-value pairs to describe complex relations in the world. LABYRINTH follows MERGE in the use of a fundamental relation that is believed to naturally decompose events in many domains. Our system uses a different representation where objects, their attributes, and relations are all included as nodes in a Bayesian network. It also uses a top-down matching of events unlike LABYRINTH's bottom-up search, ensuring more efficient insertion and retrieval.

The most recent COBWEB-based system is TRESTLE (MacLellan *et al.*, 2015). It makes many improvements to COBWEB's representational capabilities. TRESTLE can learn clusters containing numerical, categorical, relational, and component information, and it supports partial matching, allowing the system to handle partially observed elements and predict missing elements. But its concepts are flat and does not contain components, whereas our system supports hierarchical concepts. Further, TRESTLE still uses the category utility heuristic to evaluate the quality of match, and it suffers from the same overfitting issues as the original COBWEB.

A recent work that takes a different approach than the COBWEB family of systems for discovering new knowledge in structured domains is SUBDUE (Jonyer *et al.*, 2001). This system incrementally discovers substructures from input data via a lossy iterative compression procedure. To support incremental concept formation, the authors extended their system to form a *concept lattice* of substructures. Interestingly, the heuristic SUBDUE uses, namely, minimum description length, is the negation of the Bayesian Information Criterion we use in our system, which, according to (Koller & Friedman, 2009), we can interpret as the number of bits needed to encode both the model and the data given the model. But the two systems are different in other aspects, like SUBDUE's use of a directed multi-graph versus our use of a tree-based hierarchy, as well as SUBDUE's omission of specific instances in its memory versus our system's storage of both episodes and generalized schemas.

Another recent work related to our system is the Nearest-Merge algorithm (Liang & Forbus, 2014). It constructs hierarchical, probabilistic concepts via analogical generalization. The system stores both a set of generalizations and a set of unassimilated examples. Nearest-Merge extends an analogical generalization system, SAGE (McLure *et al.*, 2010), that implements structure-mapping theory (Gentner, 1983; Falkenhainer *et al.*, 1989). Our system uses a matching algorithm that is similar in spirit, although it is different from Nearest-Merge's two-stage mapping using MAC/FAC processes (Forbus *et al.*, 1995).

In the area of continual learning, Lopez-Paz & Ranzato (2017) designed a gradient episodic memory system capable of learning a diverse set of recognition tasks as cases stream in, all while avoiding catastrophic forgetting. The system accomplishes this by keeping a dedicated storage for each task that can store up to *m* examples. Deep neural nets with two hidden layers are then learned for each storage. Our system also learns multiple schemas over instance representations, but unlike the gradient episodic memory, one instance is summarized by many schemas in the hierarchy and can also be retrieved whenever appropriate.

Hintzman's MINERVA (Hintzman & Ludlam, 1980; Hintzman, 1984, 1986; Collins *et al.*, 2020) is another influential work on modelling human event memory capabilities. The system stores a trace for each experience it encounters in an archive and forms recollections in response to a retrieval cue. Unlike most archival views, MINERVA utilizes a reconstructive retrieval process. When given a retrieval cue, the system matches against all elements in its memory in parallel and forms a schema by averaging elements with high activations from each stored example. Unlike in our work, the generated schema is not stored in memory. This distinction means that MINERVA will gradually lose its ability to recollect events as forgetting takes place since it only stores the examples. Our system, on the other hand, because it stores schemas, can lose some or all of the observed examples and still use the schematic information to remember.

There have also been robotic systems with event memory capabilities. Stachowicz & Kruijff (2011) built an episodic-like memory system for a cognitive robot. Similarly, Nuxoll & Laird (2012) built an episodic memory for a simulated robot using the Soar Laird (2012) cognitive architecutre. These memory systems were built with an eye toward real-time agents in physical environments. Events are copied into an archive, then later retrieved to aid the robot achieve a goal such as answering a question, or learning a skill. Our system currently does not support a goal-driven agent, but we plan on demonstrating this capability in future work.

## 2.8 Conclusions

Event memory is a central component of human cognition. Previous models of the mental structures and processes necessary for remembering events are, however, unable to explain the full range of human uses of event memory. After reviewing the two most prominent theoretical approaches, namely, the causal and simulation theories, we introduced a novel hybrid theory of event memory. Our theory aims to address and explain the full range of human event memory phenomena. We offered a detailed description of our implementation of this hybrid theory, including the representation of episodes and event schemas and the processes that work on these representations. We also reported results from our experiment to test the system's remembering capabilities in a modified Blocks World. The results not only show that our system is able to store both specific episodes and generalized event schemas, but also validate our claim that the system can remember using hybrid representations. Our account thus provides a novel unification of existing theories, which can now be used to model various other features of human event memory.

# Chapter 3

# Modeling Human Memory Phenomena in a Hybrid Event Memory System

## 3.1 Chapter Summary

Human event memory stores an individual's personal experiences and produces their recollections with a varying degree of accuracy. To model this capacity, we have developed a hybrid event memory system that combines aspects of the two main theories proposed in the philosophical literature. We aim to model a complete range of human event memory phenomena from successful remembering to confabulations using this framework. In this chapter, we first review our hybrid event memory system and then present empirical results from a remembering experiment we conducted using this system. The results show that our system successfully models the full range of human event memory usage and errors.

## 3.2 Introduction

Event memory stores information from past experiences and makes it possible to remember those past events. When humans exercise this capacity, however, the accuracy of their recollections varies widely. Attempts at remembering can result in what is entirely correct to what is wholly inaccurate, with many degrees in between. Often times, these variations are imperceptible to the remembering subject. Philosophers who theorize about memory have recently been engaged in a debate over how to taxonomize memory errors, identifying the requirements on remembering and the ways that distinct types of memory fall short of these requirements (Robins, 2016, 2019, 2020; Michaelian, 2016a, 2020; Bernecker, 2017). Doing so generally involves a tripartite distinction between successful remembering, misremembering, and confabulation, but the two prominent accounts, the causal and the simulation theories, differ in terms of how they characterize successful remembering and how this state differs from those two forms of error. Despite a general consensus that misremembering is the less severe form of error and confabulation is the more extreme, the debate over which philosophical position is better at capturing these phenomena is at a stalemate. We believe that the lack of clear implementational details about either view contributes to this impasse because a complete evaluation of the theories is not yet possible.

Our recent work Ménager *et al.* (2021a) describes a novel hybrid theory of event memory that takes steps to address this issue. In this chapter, we argue that our view accommodates and explains the full range of event memory phenomena by combining aspects of existing causal and simulation theories, and we further provide empirical evidence to support this claim. Importantly, we do this by moving beyond just a theoretical description and provide a computational implementation making it possible to directly assess whether or not our view provides an adequate account.

The remainder of this chapter is organized as follows. In Section 2, we introduce a taxonomy of event memory phenomena and discuss how the two existing theories account for them and what their shortcomings are. Next, in Section 3, we discuss our hybrid theory and argue that it provides a broader coverage of event memory phenomena compared to the existing views. We also briefly review our implemented system which we used in our evaluations. In Section 4, we present empirical evidence showing our system's performance on a remembering task conducted in a simulated domain. During our analysis we show how the elicited recollections link back to our discussion of event memory phenomena. Then we discuss the experimental results further in Section 5 and present some future work in Section 6 before we conclude.

## 3.3 Existing Theories of Event Memory Phenomena

Causal theories (Bernecker, 2010, 2017; Debus, 2010; Robins, 2019, 2020) distinguish the three memory phenomena, remembering, misremembering, and confabulation, in terms of two features: 1) whether there is a memory trace, namely, an event-specific representation; and 2) whether its activity produces an accurate recollection. An experience of remembering that involves both of these features is an instance of successful remembering. Failures involving the first feature are confabulations and failures involving the second are misrememberings. Although causal theorists themselves do not elaborate much on the view's implementation, they give some hints as to what it would look like. One can, for instance, conceive of a system with stored event representations and a retrieval process that involves an interaction between these stored representations and the cues used to guide recall (perhaps along with other machinery). This sketch of an implementation illustrates how successful remembering would occur (i.e. retrieve and reactivate an event representation, or trace) and misremembering which would involve a malfunction of the retrieval or reactivation of the trace. The view does not, however, offer any guidance for how confabulation comes about. The only gestures at an implementation are in the direction of states that involve traces; nothing is said about what actually happens in cases where the representation produced occurs without traces, as is the case in confabulation.

Simulation theories, in contrast, have encouraged a broader taxonomy of memory states, including not only misremembering and confabulation, but veridical and falsidical forms of confabulation (Michaelian, 2016a). Simulationists also use two key features to divide up various states, but simulationists differ from causal theorists in that they do not appeal to stored traces. Instead, simulationists appeal to the reliability of the system. Reliable and accurate representations are instances of remembering. Failures of reliability produce confabulations, which are veridical if accurate and falsidical if not. When representations are reliable, but not accurate, it's a case of misremembering. Without traces, they draw the distinction between misremembering and confabulation in terms of the reliability of the event memory system from which the error emerged. Like the causal theorist, the simulationist does not have much to say about implementation. The system will not include memory traces or event representations, instead will rely on more generalized and schematized knowledge. Other than these general suggestions, however, the simulationist does not provide any guidance about the internal parameters of the event memory system that will make it reliable or not.

Our aim in this chapter is to move beyond these theories, not only with the details of the hybrid theoretical model we propose, but with its implementation. Since our theory has an implementation, we can ask how it carves up errors in the system and see whether it provides a clear account of event memory phenomena. We are particularly interested in questions of how confabulation comes about and how distinctions are made within the system between misremembering and confabulation. In the next section, we review our hybrid theory before continuing our discussion in this direction.

## 3.4 Hybrid Theory of Event Memory

Our recent work Ménager *et al.* (2021a) introduced a hybrid theory of event memory that brings together important aspects of both the causal and the simulation theories, aiming to cover the entire range of human event memory phenomena. Additionally, our theory went beyond prior theories because we implemented our hybrid theory in the context of a cognitive architecture Choi & Langley (2018), enabling researchers to evaluate the commitments and claims of the view. In this section, we briefly review our hybrid theory of event memory and its associated implementation, and then introduce the extensions made for the purpose of the current work.

### 3.4.1 Theoretical Assumptions

Existing theories of event memory commit to a single representational form to explain all event memory phenomena. In our previous work, we argued that problems accounting for the full range of event memory phenomena stem from this commitment, and further, that the problems of the causal and the simulation theories exhibited reciprocal strengths and weaknesses because they center on distinct phenomenon and corresponding representational form. The complementary nature of the two theories suggested to us that the theoretical account of memory could be improved by combining the representational commitments of both views into a hybrid theory. In this hybrid view, we hypothesize that:

- Event memory is a long-term memory that stores episodes and schemas;
- Episodes are propositional representations of specific events;
- Schemas are first-order propositional templates with probabilistic annotations;
- Event memory elements are organized in hierarchies;
- Retrieval cues play a central role in remembering; and
- Remembering an event involves performing structural matching and probabilistic inference.

More specifically, our hybrid theory commits itself to the storage, maintenance, and use of episodes and schemas in a hierarchy. Episodes correspond to the causal representation of events, while schemas correspond to the simulationist representation. Episodic contents describe the remembering agent's external and internal states, which include perceived objects and their attributes, as well as hierarchical beliefs inferred from these percepts. Like in the causal theory, experienced events are operative in producing an episode and, in that sense, a causal link exists between a specific event and its episodic representation. In contrast, schemas are probabilistic summaries of episodes and other schemas. Rather than describing a specific event, schemas represent a range of possible outcomes and are therefore aligned with the simulationist representation for events. Both episodes and schemas are stored in a generalization hierarchy, as shown in Figure 3.1, such that the leaf-level elements are episodes (shown in red color) while layers of schemas (shown in blue color) exist at

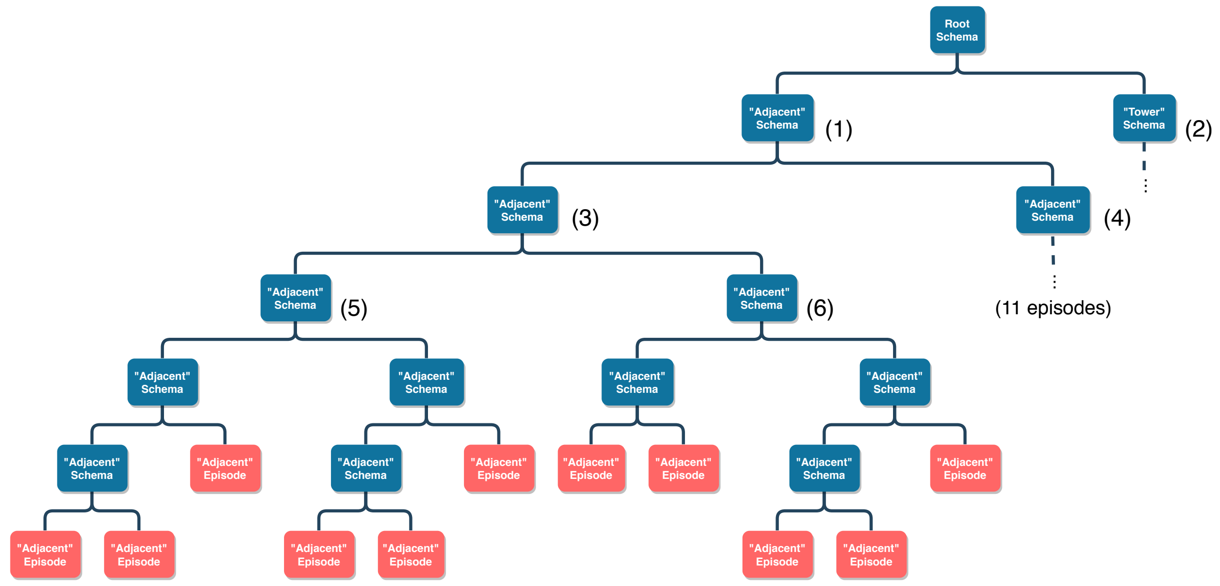


Figure 3.1: A sample generalization tree from Blocks World with `adjacent` and `tower` classes.

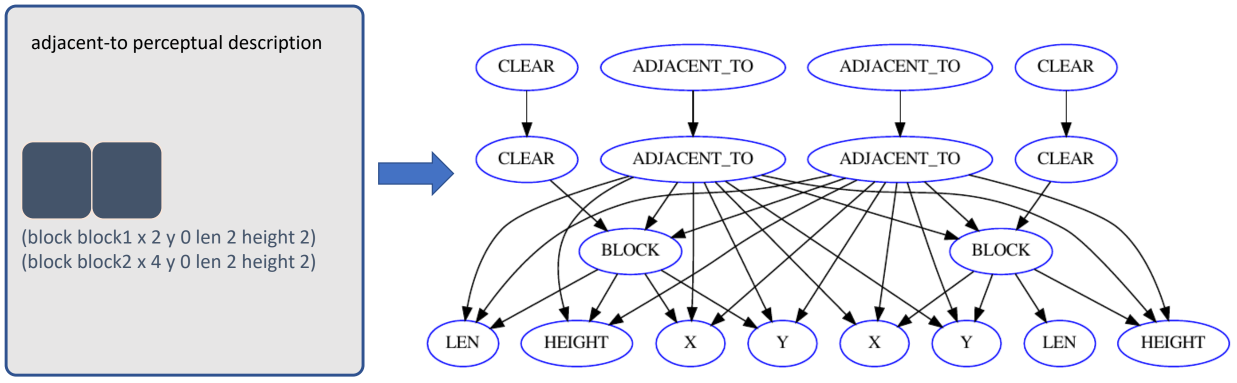
higher levels. The schemas summarize their children by aggregating all their contents and storing probabilistic annotations with them. This event hierarchy forms a general-to-specific taxonomy in which the event memory elements are connected by IS-A links from a child node to its parent.

Furthermore, our theory assumes that remembering occurs in response to a retrieval cue. Given a cue, the event memory attempts to produce an event that is consistent with the cue contents. A structural matching process decides where in the event hierarchy the retrieval should happen, using a type of similarity metric. When retrieval happens from an episode stored in memory, remembering is a straightforward return of that episode. However, when the system deems that a schema is the best match, remembering an event involves probabilistic inference over the schema to collapse the probability distributions and produce a specific instance of that schema. This inference process is inherently approximate and can result in inaccurate recollection of events, providing ways to model human event memory errors.

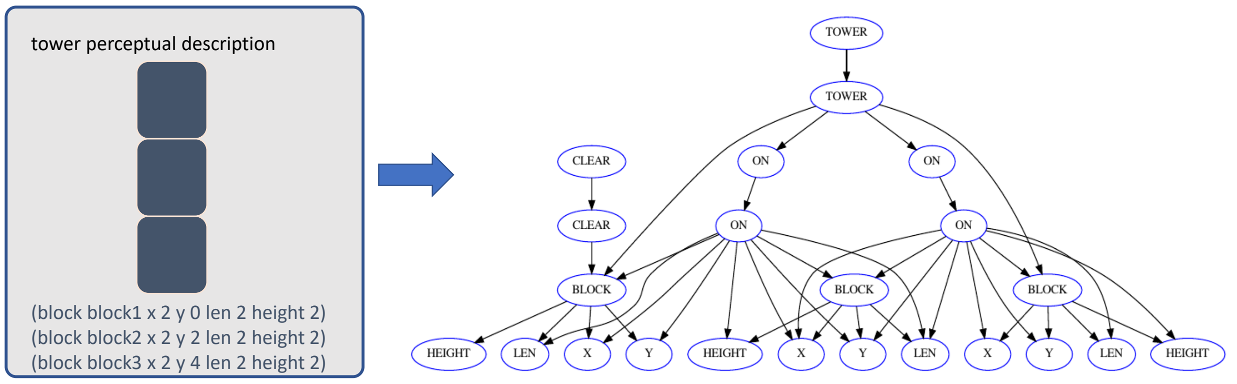
### 3.4.2 Hybrid Event Memory System

We implemented our theory within the context of a cognitive architecture, enabling us to build event memory-enabled agents and test various aspects of our theory. The system encodes each episode or schema as a dependency graph like shown in Figure 3.2, where we give two distinct examples from Blocks World. Figure 3.2a depicts a situation where two blocks are adjacent to each other, touching at one side, while Figure 3.2b shows three blocks stacked to form a vertical tower. In each case, the cognitive architecture represents perceived objects as typed predicates with attribute-value pairs. Our event memory system converts these perceptual inputs to trees of height 1, where the root node encodes the type of the perceived object (e.g., block) and the children nodes are its attributes (e.g., length, height, x position, and y position) connected to the root by a directed edge. The system represents relations like `on` and `tower` defined using perceptual inputs or other relations as trees of height 1 over their component nodes. The root node of this tree holds the type of the relation, and edges connect it to the components that participate in the relation. We also added an additional generic node which covers all the definitions of a relation, because a relation may be defined disjunctively over different objects, In the examples shown, each of the relational definitions, `on`, `tower`, `clear`, and `adjacent`, exist under their respective generic root node with the same name. Finally, nodes in an episode store the specific values of the state they encode, whereas nodes in schemas contain conditional probability distributions over possible values. In this manner, the system forms a Bayesian network for each schema.

As the agent encounters new situations in the world, our system inserts new events as episodes into its event memory. During this process, the system sorts the new episode through its event generalization hierarchy using a structural mapping procedure that attempts to match every node in the new episode to a corresponding node in an existing event memory element. The system employs Bayesian Information Criterion (Koller & Friedman, 2009) as its similarity metric to compute the quality of match between an existing event memory element and the new episode. It considers the lowest-cost match under this criterion as the most similar element in the event


adjacent-to perceptual description
(block block1 x 2 y 0 len 2 height 2)
(block block2 x 4 y 0 len 2 height 2)
CLEAR
ADJACENT_TO
ADJACENT_TO
CLEAR
CLEAR
ADJACENT_TO
ADJACENT_TO
CLEAR
BLOCK
BLOCK
LEN
HEIGHT
X
Y
X
Y
LEN
HEIGHT

(a) A sample state with two adjacent blocks and its corresponding dependency graph


tower perceptual description
(block block1 x 2 y 0 len 2 height 2)
(block block2 x 2 y 2 len 2 height 2)
(block block3 x 2 y 4 len 2 height 2)
TOWER
TOWER
CLEAR
ON
ON
CLEAR
ON
ON
BLOCK
BLOCK
BLOCK
HEIGHT
LEN
X
Y
HEIGHT
X
Y
LEN
X
Y
LEN
HEIGHT

(b) A sample state with a tower of three blocks and its corresponding dependency graph

Figure 3.2: Dependency graphs representing `adjacent` and `tower` states from Blocks World

hierarchy.

The insertion process starts from the root node of the hierarchy. The system uses its similarity metric and compares the match costs of the current node and its children with respect to the new episode being inserted. If the current node is the lowest-cost match among them, the new episode becomes a new child of this schema. The system then updates the schematic structure and the probability distributions accordingly and the insertion is complete. If, however, the lowest-cost match is one of the children, the system first updates the current schema to reflect the addition of the new episode and recursively moves to the lowest-cost child as its new current node. During this recursive process, if the system reaches a leaf node and finds the existing episode there as the best match, the memory system schematizes this element to incorporate the new episode. Then this new schema stores probability distributions that cover both the existing and the new episodes.

Using the populated event memory, the system is capable of producing a recollection of an event in response to a retrieval cue. This involves a two-step process. First, the system finds an event memory element (an episode or a schema) in the generalization hierarchy that best matches the target event, through the same structural mapping procedure used during the insertion process. Then, it produces a fully instantiated event from the best matching element. If the retrieved element is an episode, producing an event instance is a trivial process since the episode itself is a fully instantiated event. If the retrieved element is a schema, however, the system must find a consistent set of variable assignments for the schema through probabilistic inference and constraint satisfaction before it can produce an event instance.

In our previous work Ménager *et al.* (2021a), our event memory system required a complete constraint satisfaction during this process, thereby producing a fully consistent event at all times and simply returning with failure if that is not possible. But this prevents the system from generating partially correct recollections, which is necessary to model human memory errors. For this reason, we extended our system for the current work by switching from an all-or-nothing constraint satisfaction to a flexible one that can return partial solutions. With this extension, the system is able to recollect partial states, rather than failing to remember an event altogether if it cannot assign values to all the variables. This, in turn, enables us to model incomplete or erroneous recollections humans often generate.

## 3.5 Experimental Analysis

As mentioned earlier, the goal of our current work is to demonstrate that our hybrid theory explains the full range of memory phenomena, including successful remembering, misremembering, and confabulation. To do this, we conducted an experiment in a simulated Blocks World. As we briefly described in Section 3.4.2, the agent perceives blocks and infers relations among them, generating a collection of perceived objects and the inferred beliefs at any given time as a state. The event memory system encodes such states as episodes and inserts them into memory. We used the experimental setup from our previous work Ménager *et al.* (2021a) as a starting point and extended it with several additional test measures to demonstrate our theory's broad coverage of event memory phenomena. We first presented a random sequence of state observations to the agent and then tested to see if it can remember those events when presented with a retrieval cue. Through this experiment, we aim to verify the following two hypotheses:

- Our event memory system shows three distinct groups of behavior, and
- These three groups map to the three human memory phenomena, successful remembering, misremembering, and confabulation.

In the rest of the section below, we first describe our experimental design in detail and explain the data generation process. We then discuss the clustering technique we used to verify the first hypothesis and present the result. After that, we show how those clusters map onto the philosophical account of event memory phenomena using decision tree analysis.

### 3.5.1 Experimental Setup

For this experiment, we generated ten random sequences of 50 states in Blocks World. These states were drawn from two distinct classes of situations, shown in Figure 3.2, with 50% probability for each. The first class, called `tower`, describes a scenario with three blocks arranged in a vertical tower. Situations that belong to the second class, called `adjacent`, contain two blocks placed adjacent to each other, touching on one side. When we drew samples from these classes, the configuration of the blocks was determined by the drawn class, but the dimensions, the positions, and the names of the blocks could vary. We provided each sequence of 50 states to our event memory system. For each sequence, we initialized the event memory to empty and let the system incrementally populate its memory as it encountered the events in the sequence.

Once the system finished storing all the events from the sequence, we generated retrieval cues by sequentially choosing each of the 50 states and taking 20 random subsets of the selected state according to a specified degree of completeness. To generate these 20 subsets, we randomly removed a portion of nodes and their incoming and outgoing edges from the corresponding full state until they met the specified completeness requirement. We provided these retrieval cues to the system and measured its recollection responses.

We considered this process as one epoch and repeated it ten times for each sequence. In every epoch, we modulated the completeness of the retrieval cue. We initially supplied full states as retrieval cues in the first epoch and gradually reduced their completeness by 10% as we moved to the subsequent epochs. By the tenth epoch, the completeness of retrieval cues drops to only 10% of the original states. This would result in a total of 100,000 recollection trials (10 sequences $\times$ 10 epochs $\times$ 50 states $\times$ 20 subsets as retrieval cues). After inspecting the generated partial-state retrieval cues, however, we realized that some trials used ambiguous retrieval cues, so we filtered them out before running the memory system over them. We labeled a cue ambiguous whenever its contents were a subset of both `tower` and `adjacent` classes. A good example of such a retrieval cue is `((block A) (block B) (on-table B))` because both classes can contain these elements. We removed these cues because it would not have been fair to ask the memory system to remember the correct event when the retrieval cue itself does not uniquely identify the target class. As a result of this filtering, we had 88,839 recollection trials available for the memory system.

### 3.5.2 Generating Recollection Trial Data

As our system ran over these trials, we collected various measurements regarding the recollection performance of the system, capturing whether or not the retrieved memory element had the same structure as the target event specified in the cue, as well as the accuracy, precision, recall, and F1 score of the recollected event. We performed two analyses over this dataset as we describe in the next sections. Each data point from our recollection trials consists of six features: 1) recollection coverage; 2) recollection reverse coverage; 3) accuracy; 4) precision; 5) recall; and 6) F1 score. These are defined as:

**Recollection coverage** Percentage of nodes in the target event that are matched to nodes in the recollection. For example in Figure 3.3, each node in the target event matches to a unique node in the recollection. The match only considers whether the nodes play the same role in their respective graphs, and does not check whether their assignments are equal. So, the recollection coverage is 100% in this case.

**Recollection reverse coverage** is the percentage of nodes in the retrieved event memory element that are mapped to nodes in the target event. Since the recollection contains two nodes unmatched in the target event, the recollection reverse coverage is 3/5. Both of these coverage measures only check whether the nodes are present in their respective event, rather than checking the value assigned to these nodes. Hence, these features in combination give a sense of whether the target event and the recollected event share the same structure.

**Accuracy** is the correctness of the recollection response in terms of the number of nodes in the dependency tree with correct value assignments over the total number of nodes. More formally, it is defined as:

$$Acc = \frac{tp+tn}{tp+tn+fp+fn}, \tag{3.1}$$

where $tp$ is the number of true positives, $tn$ is the number of true negatives, $fp$ is the number of false positives, and $fn$ is the number of false negatives. In our context, a true positive occurs when

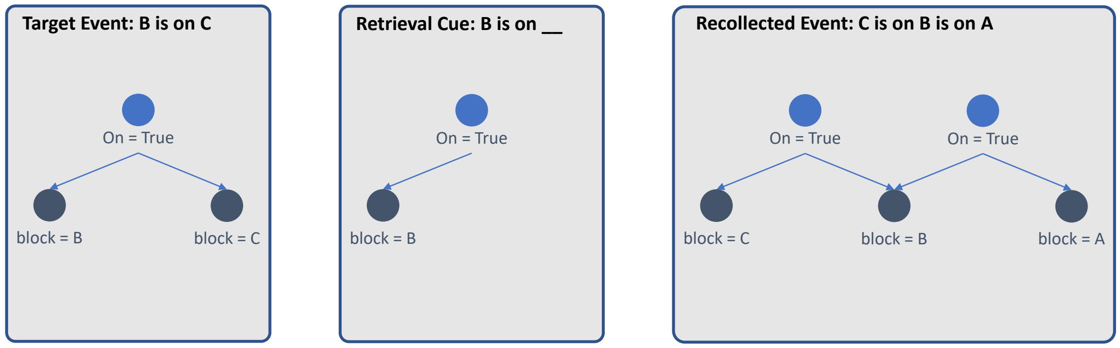


Figure 3.3: Example target event with its associated retrieval cue and recollected event.

the value of a node in the recollection matches the value of its counterpart in the target event. A true negative occurs when a node in the recollection does not have an assigned value indicating that the node is not present in the recollection, and there is no corresponding match for the node in the target event. False positives can occur in two ways. In one case, they occur whenever the recollected event contains a value-assigned node that is not present in the target event. In the other case, false positives occur when both the target event and the recollected event contain the same node, but the assigned values differ. Similarly, a false negative can occur in two ways. The first is when the recollected event is missing a node that exists in the target event, and the second occurs when the node in the recollection does not have an assigned value, but the target event assigns a value to that node. In our example in Figure 3.3, the recollection has an accuracy of 40%.

**Precision** measures the system's ability to avoid false positive responses. This is defined as:

$$Prec = \frac{tp}{tp+fp}. \tag{3.2}$$

To see that precision measures the system's ability to avoid false positives, observe that it is maximized whenever false positives are 0, and is strictly less than 1 for any case where there are more than 0 false positives. The recollection in our example obtains 40% precision.

**Recall** is a measure of the system's ability to avoid false negatives. That is,

$$Rec = \frac{tp}{tp + fn}. \tag{3.3}$$

Similarly to precision, recall is maximized when no false negatives exist in the recollection. In the case of our example recollection, we see that recall is 100% since no false negatives exist.

**F1 score** is the harmonic average of precision and recall and measures the overall quality of the recollection. It is defined as:

$$F1 = 2 \cdot \frac{Prec \cdot Rec}{Prec + Rec}. \tag{3.4}$$

For the example shown in Figure 3.3, the recollection has an F1 score of 57.14%.

### 3.5.3 Clustering Recollection Trial Data

We first analyzed the recollection trial data to find any groups of data points that share unique characteristics. To do this, we needed to plot the data points in the multi-dimensional feature space and attempt to identify any meaningful clusters among them. Then, the unique characteristics of each cluster helped us map it onto a memory phenomenon.

Having six features to analyze in a dataset is challenging for many clustering algorithms, however, one common approach for dealing with multi-dimensional data like ours is to project the high-dimensional feature set down to a lower dimensional space using dimensionality reduction. In our case, this facilitated our cluster analysis by removing noisy or non-predictive features so that we could identify clusters in the data more easily. We used Principal Component Analysis (PCA) (Jolliffe, 2002) to reduce our six-dimensional feature set to a two-dimensional space. Figure 3.4 shows the contribution of each original feature to the two principal components. The first component is a combination of all the original features with roughly equal amounts of contribution.

The first component roughly gives equal weight to each component, having a slight negative bias

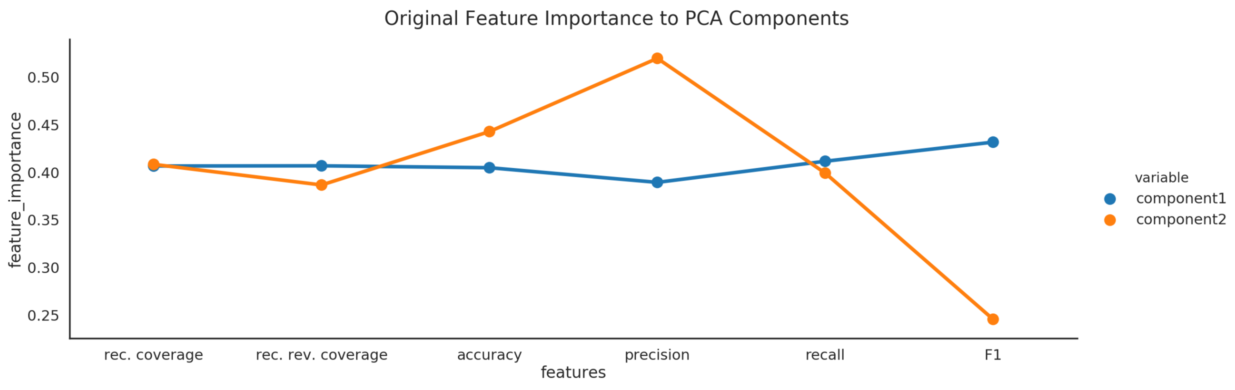


Figure 3.4: Contributions of original features to principal components.

against precision and a preference for F1. The second principal component, however, receives more significant contributions from accuracy and precision, and heavily discounts the influence of F1.

These principal components accounted for about 98% of the variance in the original seven features. Given these principal components, we plotted the data in Figure 3.5 and began our analysis with visual inspection. We saw one distinct cluster at the bottom right of the figure separated from a larger mass of data points on the left. Closer inspection of the latter revealed that the data points on the left split into two clusters near Component1 = 1 on the horizontal axis.

Although a visual inspection of this data gave us an intuition for the clusters that exist in the data, we needed to go beyond this and assign each data point to a cluster in an analytical manner in order to understand how the event memory phenomena correspond to these clusters. We chose a clustering algorithm, OPTICS (Pedregosa *et al.*, 2011), to do this task. OPTICS is a density-based clustering algorithm that can identify clusters of any shape, size, and density, unlike other clustering strategies including Gaussian mixture models (Reynolds, 2009) or k-means (Lloyd, 1982). The algorithm can also handle noisy data points that the other clustering algorithms cannot easily deal with. These characteristics are important in our context because the potential clusters in our data can vary in all of these aspects. Furthermore, tuning the OPTICS model only requires setting a couple of intuitive parameters[1] and, importantly, the number of desired clusters is not one of them.

[1]We carried out a grid search for the optimal parameter settings for `eps` and `min_samples`. The `eps` parameter

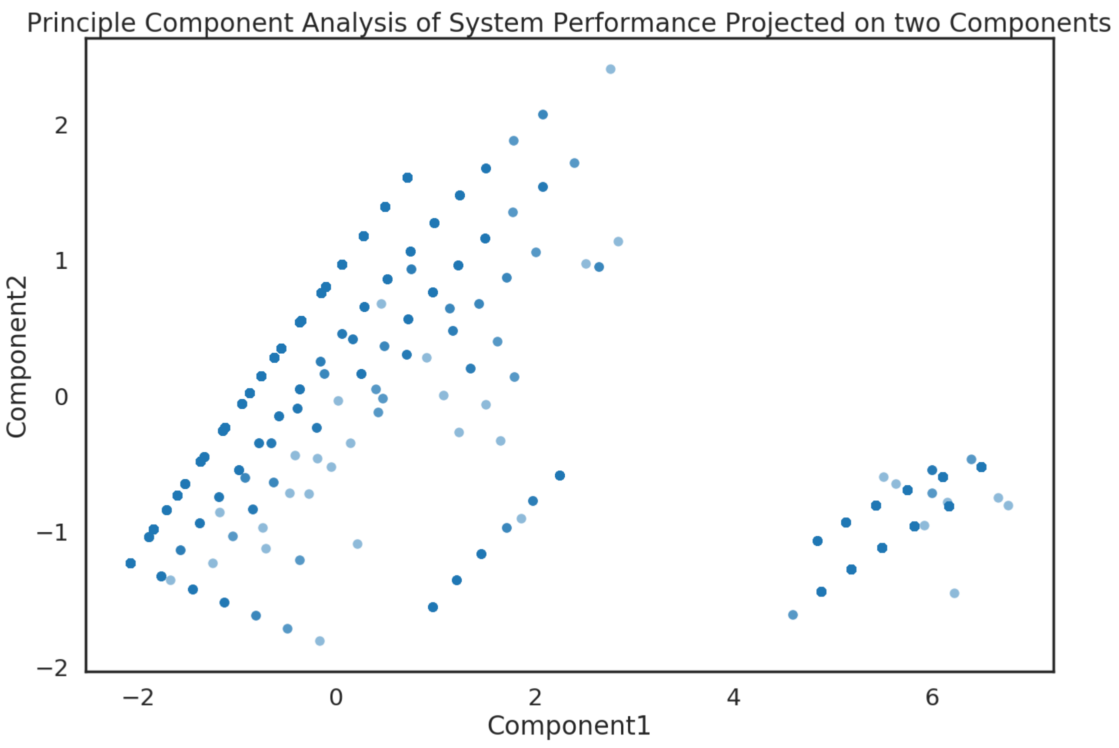


Figure 3.5: Recollection performance data scattered across principal components.

Figure 3.6 shows the discovered clusters in our recollection trial data. The scatter plot displays the data points assigned to each cluster with a different color. The result indicates that OPTICS identified three clusters in our data, shown in blue, green, and orange colors, respectively. There are also a number of noisy data points, shown in red color. The first cluster, Phenomenon 0, is in the left corner of the scatter plot and contains 78,443 data points. Phenomenon 1 is the green cluster in between the other two and contains 52 members. Lastly, Phenomenon 2 is to the left side containing 10,336 data points. The remaining eight data points are labeled as noise. The plots near the component axes display the density of each cluster projected along the principal components. These show that each cluster is well-defined and separated from the others.

sets the maximum Minkowski distance between two data points in the same neighborhood, and `min_samples` sets the minimum number of required data points in the neighborhood of a point in order for it to be a core node of a cluster. This parameter was set as a fraction of the total number of samples in the data. Our grid search yielded the `eps` of .001 and the `min_samples` of $7 \times 10^{-5}$. We also set the cluster extraction method to 'dbscan'.

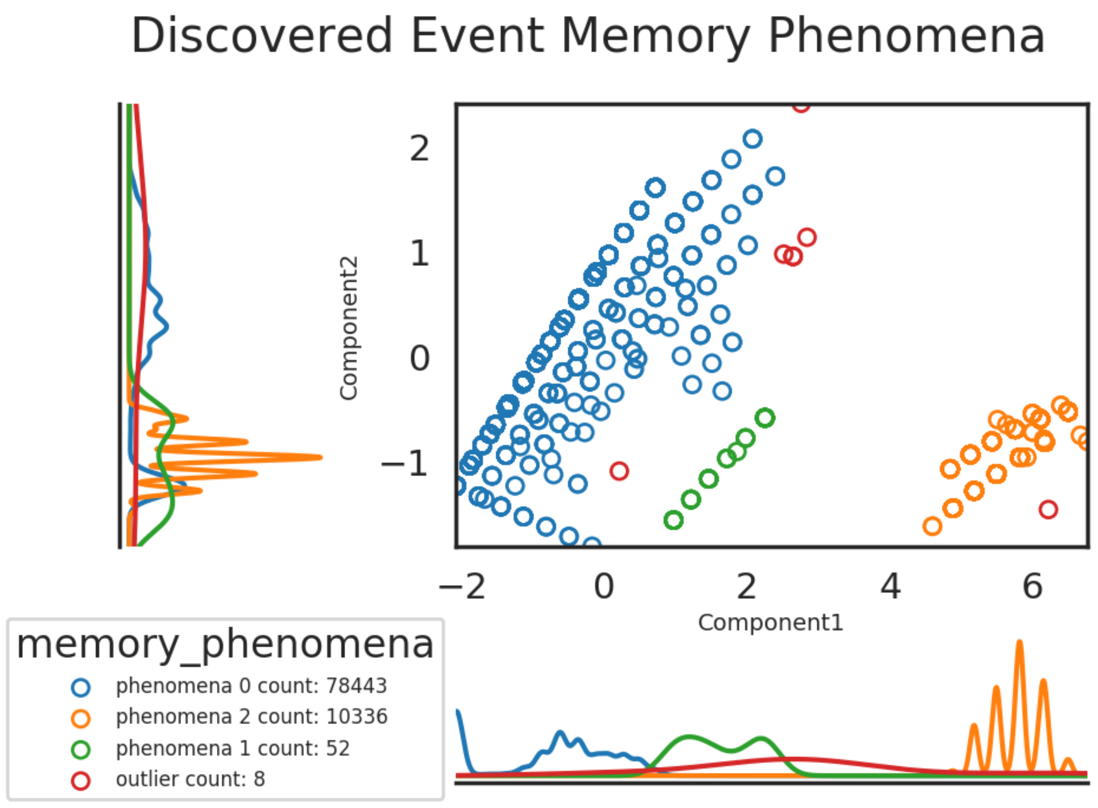


Figure 3.6: Recollection performance data annotated with cluster assignment.

### 3.5.4 Mapping Clusters onto Event Memory Phenomena

Discovering the three clusters in our experimental data is an encouraging sign, since we hypothesized that our system can model three kinds of human event memory phenomena. But Figure 3.6 only shows that there are three groups of data points and does not characterize these clusters. To obtain such information, we augmented the data points in our original dataset with their cluster assignments and generated Table 3.1 summarizing each cluster in terms of the recollection performance.

From this elaboration on each cluster, we can now attempt to characterize the three corresponding recollection phenomena. The table shows that Phenomenon 0 has near perfect recollection coverage and perfect reverse coverage. Additionally, the recall is close to 100%, and the precision

Table 3.1: Cluster Performance Characteristics (%)

| Accuracy | | Recall | | Recollection Cvg. | |
|---|---|---|---|---|---|
| Phenomenon 0 | 70.35 | Phenomenon 0 | 99.94 | Phenomenon 0 | 99.97 |
| Phenomenon 1 | 49.65 | Phenomenon 1 | 63.55 | Phenomenon 1 | 61.11 |
| Phenomenon 2 | 14.38 | Phenomenon 2 | 29.51 | Phenomenon 2 | 55.05 |
| **Accuracy (std)** | | **Recall (std)** | | **Recollection Cvg. (std)** | |
| Phenomenon 0 | 19.72 | Phenomenon 0 | 0.93 | Phenomenon 0 | 0.56 |
| Phenomenon 1 | 7.91 | Phenomenon 1 | 3.76 | Phenomenon 1 | 0.00 |
| Phenomenon 2 | 3.82 | Phenomenon 2 | 5.96 | Phenomenon 2 | 5.78 |
| **Precision** | | **F1** | | **Recollection Rev. Cvg.** | |
| Phenomenon 0 | 70.37 | Phenomenon 0 | 81.06 | Phenomenon 0 | 100.00 |
| Phenomenon 1 | 68.98 | Phenomenon 1 | 65.99 | Phenomenon 1 | 100.00 |
| Phenomenon 2 | 22.35 | Phenomenon 2 | 24.95 | Phenomenon 2 | 63.79 |
| **Precision (std)** | | **F1 (std)** | | **Recollection Rev. Cvg. (std)** | |
| Phenomenon 0 | 19.71 | Phenomenon 0 | 13.32 | Phenomenon 0 | 0.18 |
| Phenomenon 1 | 10.98 | Phenomenon 1 | 7.11 | Phenomenon 1 | 0.00 |
| Phenomenon 2 | 6.68 | Phenomenon 2 | 5.80 | Phenomenon 2 | 5.78 |

and accuracy are around 70%. Further, the F1 score is at 81% indicating that the members of this cluster are high-quality recollections that cover the target event without adding extraneous details.

Then we compared the other two clusters. Unlike Phenomenon 2 that has 55.05% recollection coverage and 63.79% reverse coverage on average, Phenomenon 1 achieves 61.11% recollection coverage and has perfect reverse coverage. This implies that recollections in this cluster recover much of the structure from the target event, but still fail to capture it completely. Looking at the other performance measures, we see that the accuracy, precision, and recall are 49.65%, 68.98, and 63.55, respectively. These moderate values suggest that Phenomena 1 captures middle-of-the-road recollections that contain a mixture of accurate and erroneous responses.

Finally, the recollection coverage and the recollection reverse coverage for Phenomenon 2 at 55.05% and 63.79%, respectively, suggest that the retrieved event memory element and the target event do not share the same structure. This implies that the event memory system remembered the wrong kind of experience in response to the cue. This is also reflected in its low accuracy (14.38%), precision (22.35%), recall (29.51%), and F1 score (24.95%).

Having described the phenomenal characteristics of each cluster, we now ask how these clusters map onto the phenomenal categories that researchers use to classify types of event memory phenomena. There might be a number of ways to construct a mapping from cluster to event memory phenomenon, but our analysis above suggests that one plausible assignment is to label Phenomenon 0 as successful remembering, Phenomena 1 as misremembering, and Phenomenon 2 as confabulation. With this mapping in mind, we can interpret Figure 3.6 as showing the clusters for successful remembering and misremembering close to each other on the left and having the confabulation cluster further away at the bottom right corner of the scatter plot. Moreover, most of the recollections are instances of successful remembering, followed by confabulation, and then misremembering with the least frequent occurrences in the data.

While we believe this is the most straightforward way to do the mapping, it is not perfect. When we checked how well the mapping fits, namely, whether the clusters as we have labeled them truly reflect the phenomena we are trying to model, a concern arose pertaining to the average accuracy rate. The highest reached for any phenomenon was 70%. Should 70% accuracy be considered successful remembering? Without a baseline implementation from previous work in the form of an implemented human event memory system, it is difficult for us to judge this. One might have hoped that successful remembering would involve 100% accuracy, but this seems to be an unreasonable standard. Studies of human memory often determine success based on whether key features of an experience are retained. Without knowing precisely what is encoded and how much information is available to be encoded, it is difficult to determine what proportion is retained in cases of success. For this first implementation, we are satisfied with 70%, because it is sufficiently distinct from the accuracy rates observed in the other clusters. We suspect that this may be attributed to the limited amount of variations we could produce in the events our system experienced in the Blocks World. The system may have had trouble identifying the target event amongst very similar events, resulting in this accuracy result.

Despite this limitation, we are still encouraged by the mapping we found between our clusters and the widely accepted categories of event memory phenomena. We believe that our results provide an important baseline for further research in this area.

## 3.6 Discussion of Mapped Memory Phenomena in Our System

In the section above, we presented results that suggest our event memory system models the full range of human event memory phenomena. We did this by demonstrating the extent to which performance measures, like accuracy and precision, relate to known categories of memory phenomena. Given this encouraging outcome, we are further interested in providing an account of how each phenomenon arises in our system. We will do this by showing how internal system parameters correspond to our discovered phenomena. In this stage of our analysis, there are four pertinent parameters as follows:

**Weighted distance:** quantifies the quality of match between the cue and the retrieved event memory element. This parameter was z-score normalized and ranges from [-1.18, 3.47].

**Retrieved element count:** indicates the number of episodes summarized by the retrieved event memory element thus giving a measure of stability for the retrieved event memory representation. Stable event memory elements summarize many episodes and less stable elements summarize fewer. This parameter was z-score normalized from [-0.26, 23.32].

**Retrieved element depth:** describes the depth of the retrieved event memory element from the root of the root of the tree. This parameter was z-score normalized and ranges from [-3.4, 3.97].

**Retrieved element type:** specifies whether the retrieved event memory element was an episode or a schema.

To begin our analysis we annotated each record in our dataset with the event memory phenomenon it belongs to. Then, we dropped all the performance measure features from the dataset, leaving us a dataset containing the four system parameters from which to predict event memory phenomena. Since each record in the dataset now had an associated label specifying the elicited phenomenon, we applied a supervised machine learning technique, called a decision tree (Pedregosa *et al.*, 2011), to discover the conditions under which our event memory system produced the different event memory phenomena. We balanced the class weights of the decision tree which ensured that it would not use the relative frequency of the phenomena to bias the predictions toward a particular class. To learn this decision tree, we split our dataset into a training and testing set, where the training data comprised 80% of the data. We conducted 10-fold cross-validation over the training set and obtained 93.28% average accuracy. For the test set we observed 93.37% accuracy indicating that the model generalized well.

The final decision tree was quite large so we are not able to visualize the complete model here, but Figure 3.7 shows the top portion of the learned decision tree. Although we do not display the entire tree the not shown class assignments in lower-level nodes do not vary much.

Each node in the tree contains fields for the condition, entropy score, number of samples, value, and class. The condition field shows a Boolean condition on a specific event memory system parameter, and the entropy score is a measure of node purity. An entropy score of zero means that the node contains examples of one and only one phenomenon. Entropy scores greater than zero indicate that the node contains some mixture of phenomena. The mixture proportions in the node are subsequently shown in the value array. The samples field shows the total number of records in the dataset under consideration at that node. The class field indicates the class in which most of the examples satisfying the condition belong, and the color and intensity of the node match the predicted class and purity of that node. For example, the root node is clear because all the classes are equally probable. The adjacent node along the 'false' branch is colored green because most of the records there are cases of misremembering. Finally, we truncated the rest of the tree in order to present the most essential information in the figure.

The rules in the decision tree are conjunctions of conditions and can be discovered by sequentially

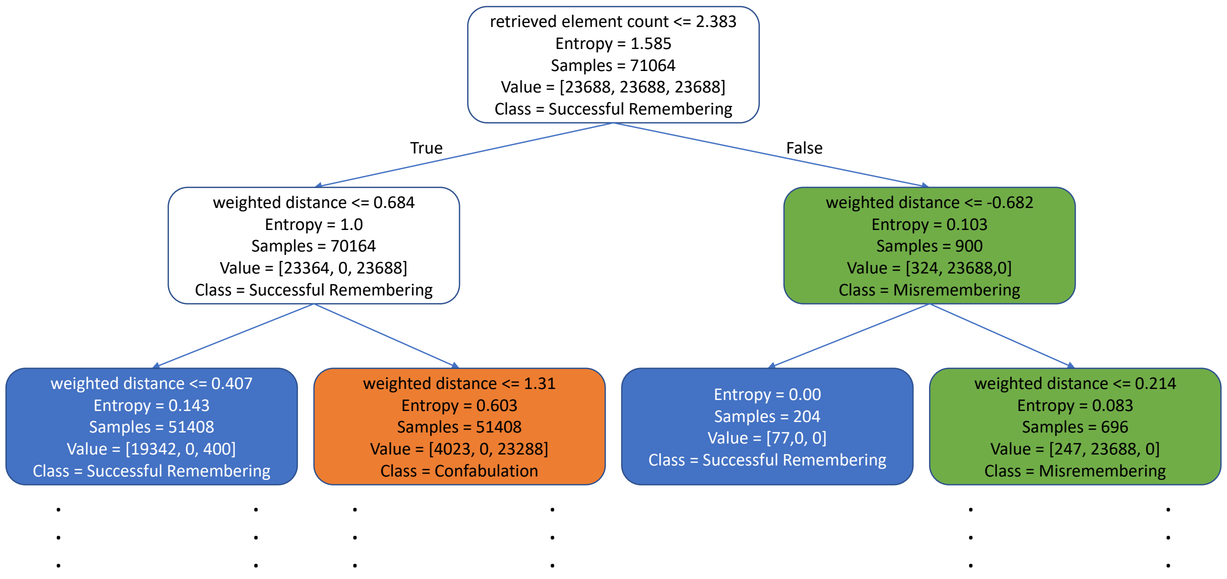


Figure 3.7: First three levels of the learned decision tree.

evaluating the condition in each node, beginning from the root. If the condition evaluates to ‘true’, then the next condition to evaluate is found at the adjacent node along the ‘true’ branch. Otherwise, the next condition of the rule is the node along the corresponding ‘false’ branch. The complete decision rule constitutes any path starting from the root of the tree down to the leaf.

With this in mind, we examined the decision rules in the tree to obtain our theory’s account of the event memory phenomena. The first rule states that successful remembering occurs whenever the retrieved event memory element count and cost of matching to the cue are low. This rule fits our intuition because a low event memory element count means that the retrieved representation was a schema that summarizes a small number of episodes, or that it was an episode. Further, the low distance means that the quality of match between the cue and the retrieved element was very good. In such cases, we would expect successful remembering to occur.

The next decision rule states that confabulation happens in cases where the retrieved element count is low, indicating the system retrieved an episode, or lower-level schema, but the weighted distance is not low. This set of conditions may arise due to a poorly specified retrieval cue which is incomplete and missing information. Hence, during retrieval candidate branches may seem to be roughly equally bad matches to the cue, causing the system to traverse the wrong path.

Another kind of successful remembering occurs whenever the retrieved event count is not low, but the weighted distance is low. This implies successful remembering from a stable schema. Since our view is a hybrid theory, we expected to observe two cases of success in our analysis. One is consistent with causal accounts requiring the retrieval and activation of a stored representation, while the other aligns with the simulationist view relying on an inference procedure to reconstruct the state.

The last decision rule, which predicts misremembering, fires when the retrieved element count and weighted distance are not low, suggesting that the system remembered using a schema higher in the event hierarchy. Such an element may produce a range of possible experiences and, given the cue, the produced recollection recovered the state and introduced errors as well.

This decision tree demonstrates our theory's ability to account for the different event memory phenomena. Despite only showing a few levels of the decision tree, we showed that our account's broad coverage which includes elements of both preservative and constructive views. For example, our theory allows for successful remembering to occur using both schemas and episodes and distinguishes confabulation from misremembering. Therefore, we believe our view is a true a hybrid theory that can explain both successes and failures of event memory usage.

## 3.7 Future Work

As discussed so far, the experimental results showed that our system demonstrated three kinds of recollection outcomes which, through our analysis, we linked to the successful remembering, misremembering and confabulation. We additionally showed our theory's explanation of the conditions under which each phenomenon was produced. These results led us to claim that our hybrid theory of event memory provides a plausible account of human event memory and our implementation successfully models its phenomena.

Given these promising results, we plan to continue our work along three main dimensions. First, we would like to capture the temporal dimension in our event memory representations. Currently, our system only handles state-of-affair representations, which describe perceived objects and relations defined over them. Many events, however, unfold over time and our system's inability to capture this is a serious limitation that we need to overcome. Doing so will open the door for us to build even richer event memory-enabled intelligent agents that can talk about their past, make predictions about their future, or even understand the goals and intentions of other agents.

Consequently, our second goal is to apply this extended system to plan, activity, and intention recognition problems (Ménager *et al.*, 2017; Pei *et al.*, 2011; Mirsky *et al.*, 2018). Traditionally, work in this area has focused on recognizing the top-level goals and intentions, as well as low-level actions of an agent. This, while interesting, may not be very useful for interactive collaborative agents. These systems will need to infer not only top-level goal and intention information, but also predict the specific methods used to complete tasks. Toward our goal of building such agents, we would like to extend our system to store hierarchical, temporal event representations which would allow it to not only infer the top-level goal of an observed agent, but also all the intermediate subgoals and intentions associated with them.

Both of these extensions fundamentally rely on the underlying probabilistic inference system. Currently, it cannot perform inference over real-valued variables. Instead, it treats real numbers categorically. Hence, our third goal is to extend our system to handle both continuous and discrete probability distributions during inference. We believe this will require us to utilize hybrid Bayesian Networks (Koller & Friedman, 2009), but will payoff by allowing our system to operate in real-world environments where reasoning over numerical information is important.

## 3.8 Conclusions

Human recollections of events exhibit a range of event memory phenomena. Previous theories attempting to explain this have done so with partial success. In this work, we presented a novel hybrid theory of event memory which, we argued, explains the full spectrum of event memory phenomena. To support this claim, we conducted experiments in Blocks World and showed that our system accounts for both the successes and failures of human event memory usage. Although continued research in this area is necessary, we believe that our work provides a more complete model of the memory phenomena and represents an important step toward a complete understanding of human event memory.

# Chapter 4

# A Cognitive Architecture with an Event Memory Module

## 4.1 Chapter Summary

Event memory is an important component of human cognition. Over the years, researchers across the fields of artificial intelligence, philosophy, and psychology have studied the role of event memory within the mind. In this chapter, we continue this research thrust and describe our initial efforts to extend a cognitive architecture with an event memory and situate an agent in the popular videogame, Minecraft. We argue that our agent can serve as a basis for future experiments on event memory by demonstrating how the agent receives perceptual inputs and executes plans to build compound tools in the world of Minecraft.

## 4.2 Introduction

Event memory is an important aspect of human cognition. It allows people to store their personal experiences and recall them at later times for a variety of purposes. Over the years, there have been numerous efforts in psychology (Tulving, 2002; Hellerstedt, 2015; Schachter, 1987), philosophy (Martin & Deutscher, 1966; Michaelian, 2016b; Robins, 2016), and artificial intelligence (Doshi *et al.*, 2015; Kelley, 2014; Lopez-Paz & Ranzato, 2017) to understand the nature of event memory. Within artificial intelligence research, few examples of computational models of integrated event memory systems exist.

One of the few architectures with event capabilities, Soar (Laird, 2012), stores individual states represented as graphs in a flat container (Nuxoll & Laird, 2012, 2004). Elements are input into the memory system for each agent action and they persist there without any generalization taking place over them. Elements may be retrieved from this memory by sequentially matching a retrieval cue to the elements, and retrieving the best match. In another architecture, architecture FAtiMA, researchers integrated an autobiographical memory for building virtual agents that can report on past experiences (Dias *et al.*, 2007; Ho & Dautenhahn, 2008). The autobiographical memory system stores episodes independently, where each episode captures a sequence of emotion-relevant actions and state observations. When the system wants to generate a summary of a past experience, it searches each episode, matching it against a retrieval cue, then returning the most relevant match. Lastly, Brom *et al.* (2007) describe an agent architecture with episodic memory capabilities for building virtual agents in role-playing games. The architecture's episodes represent the agent's perceptions along with the agent's hierarchical goals and intentions. During insertion, episodes are hashed in memory, then the agent can retrieve those experiences in constant time to give explanations about its past behavior.

While the discussed agent architectures have their unique virtues and strengths, none have been able to represent and maintain generalized experiential knowledge in the event memory. We have demonstrated that our previous work on event memory (Ménager *et al.*, 2021a,b) is well-suited for demonstrating this capability. In this chapter, we present our initial efforts to integrate our event memory system as a cognitive module inside the ICARUS cognitive architecture (Choi & Langley, 2018) which provides a psychologically plausible infrastructure for building intelligent agents. We first review the ICARUS architecture, describing its representations for semantic, procedural, and episodic knowledge. We then describe the architecture's control strategy which leverages these to drive goal-driven behavior. Next, in Section 3, we introduce the Minecraft domain which is the test environment for our ICARUS agent. Section 4 describes this agent, and we show that the agent can operate in the environment to achieve goals. We end with considerations of future work involving our event memory module in section 5 before concluding.

## 4.3 ICARUS Review

The ICARUS cognitive architecture is a computational theory of human-level intelligence. The theory aims to provide a qualitative account of the broad range of human behaviors while remaining consistent with broad-stroke findings in psychological literature. Consequently, some of its core postulates are widely shared across other architectures like Soar (Laird, 2012), ACT-R (Anderson *et al.*, 2004), PRODIGY (Carbonell *et al.*, 1991), and other theories. These are that: memories are collections of symbolic structures; short-term memories are distinct from long-term stores; relational pattern matching accesses long-term memory content; cognitive processing occurs in recognize-act cycles; and cognition dynamically composes mental structures (Choi & Langley, 2018). In addition to these common assumptions, ICARUS makes a number of additional claims about the nature of cognition which position the architecture well for modeling embodied intelligent systems. The theory posits that:

- Cognition is grounded in perception and action;
- Semantic and procedural knowledge are distinct cognitive structures;
- Short-term memory elements are instantiations of long-term structures;
- Long-term knowledge is organized in a hierarchical manner; and
- Inference has primacy over execution, which in turn has primacy over problem solving.

Although we recognize that some of these tenets are not unique to the architecture, only ICARUS combines them into one unified theory, that when taken together, represent a novel theory of cognition. Figure 4.1 shows a block diagram of the ICARUS architecture which we will use to guide our discussions of how the architecture implements the above-mentioned postulates. We begin by discussing the knowledge structures and their associated memories, then move to explore the architecture's mental processes which occur over these to drive intelligent behavior.

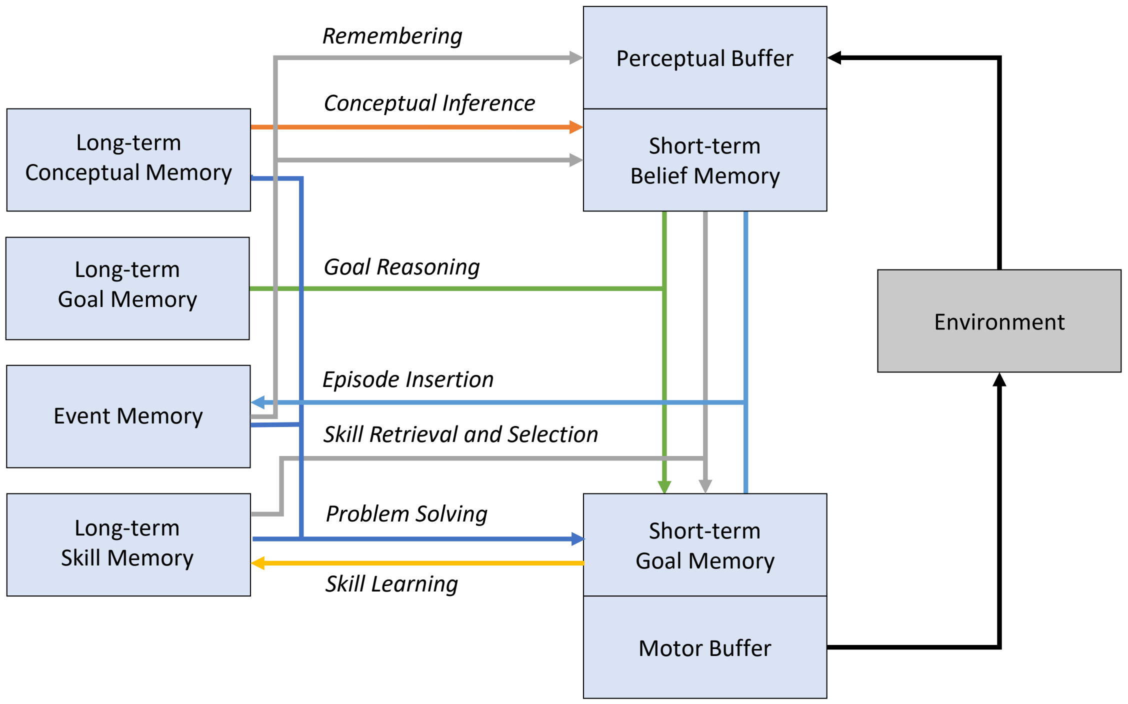


Figure 4.1: ICARUS diagram with event memory module.

### 4.3.1 Representations and Memories

As mentioned above, the architecture distinguishes between long-term and short-term memories, and differentiates semantic memory from procedural memory. ICARUS contains four long-term memories. Figure 4.1 shows that one of these is a procedural memory, called the Long-term Skill Memory, for storing skills that achieve goals. The remaining three are declarative memories which are the conceptual, goal, and event memories. The Long-term Conceptual Memory houses concept definitions for objects and relations. The Long-term Goal Memory contains goal nomination rules for dynamically nominating and adjusting the agent's top-level priorities in response to the environment. And, the Long-term Event Memory is a general-purpose storage for the agent's experiences, capturing instantaneous descriptions of both internal memory contents, as well as perceptions and beliefs about the external state.

In addition to these long-term stores, the architecture includes two short-term memories constituting its working memory. The Short-term belief memory stores the agent's beliefs about current perceptions of its environment, and the short-term goal memory holds the current goals and intentions the system will to carry out to achieve them. In the following, we describe these memories and their corresponding knowledge structures. We start with discussing the conceptual and belief memories, then move to the skill memory and goal memories. Lastly, we cover the event memory and its representations.

Conceptual memory stores concepts which describe various aspects about a perceived situation. Table 4.1a shows a notional concept which resembles a Horn clause (Horn, 1951) having a predicate head with variableized arguments, variableized perceptual matching conditions, optional references to any subrelations in the `:conditions` field, and zero or more tests against matched variables. Concepts without subrelations are called *primitive concepts* and are defined directly over a set of perceived objects and their attributes, while *non-primitive* concepts describe more complex situations by specifying at least one subrelation. The subrelations are references to the heads of other concepts and thereby impose a hierarchical order amongst a concept and its lower-level counterparts. Concept definitions may be instantiated as *beliefs* when a consistent assignment binds variables to specific values from the environment. Instantiated concepts exist in the Short-term Belief Memory.

ICARUS skills are procedures for achieving goals in the environment. Table 4.1b shows that they are hierarchical STRIPS operators (Fikes & Nilsson, 1971). The knowledge structure includes variableized descriptions of several fields. These include a named head which identifies the skill, perceptual matching conditions and preconditions that indicate when the skill is ready to execute, motor-level actions to carry out or subskills to decompose that prescribe the work to perform, and the expected environmental effects once that work is complete which may involve matching attributes in the effects against test functions in the `:tests` field. Similar to concepts, skills in the architecture may be primitive or non-primitive. A *primitive skill* contains an `:actions` field and describes the effects of the specified actions when the skill is executed. A *non-primitive* *skill* describes a composite procedure by specifying the execution order of lower-level subskill decompositions. The `:subskills` field contains these decompositions which are references to the heads of other skills. This again enforces a hierarchical order among skills and their subskills. Instantiated skills, or *intentions* are grounded, recursive skill structures that show the complete execution path from a skill down to its primitive skills which bottom out with actions. At each decomposition step, the intention captures the current procedural operator and recurses on the active subskill, until there are no more decompositions. The executing intention, along with the goals it achieves, reside in the Short-term Goal Memory.

A goal in the architecture is a set of concept instances to make true in the environment. These can be given directly to the system or can be dynamically nominated and retracted via architectural procedures. In both cases, the Long-term goal memory fulfills a crucial role by storing goal nomination rules. Such rules contain a variableized predicate head, variableized nomination conditions, and a priority field. The head names the goal to nominate when the goal nomination conditions are satisfied. The priority field measures the importance of satisfying the goal. Goal nomination rules with no conditions always nominate their goals by default. Nominated goals live in the Short-term Goal Memory where they may be associated with corresponding intentions.

Lastly, we utilized our previous work (Ménager *et al.*, 2021a,b) to extend this architecture with an event memory module. The module stores episodes and schemas which represent long-term experiential knowledge. Episodes are propositional descriptions of individual events, while schemas introduce probability to these descriptions allowing them to encapsulate a range of possible outcomes. Table 4.1c shows the general template for episodes and schemas in the architecture. They contain a field for a unique identifier, a state description which may be a dependency graph in the case of an episode or a Bayesian network for the schema, as well as a count for enumerating the number of times the experience occurred. The identifier and count fields are evident from their names, so we do not explain them here, however, we will clarify the state description field. The state description represents the system's current perceptions and beliefs.

Table 4.1: Syntax for ICARUS long-term mental structures.

```
(⟨head⟩
  :elements   (⟨percept⟩+)
  :conditions (⟨belief⟩*)
  :tests      (⟨test⟩*))
```

(a) Concept definition template.

```
(⟨head⟩
  :elements   (⟨percept⟩*)
  :conditions (⟨belief⟩*)
  :actions    (⟨action⟩*)
  :subskills  (⟨subskill⟩*)
  :tests      (⟨test⟩*)
  :effects    (⟨belief⟩+)
```

(b) Skill definition template.

```
(:id ⟨uid⟩
 :state ⟨graph⟩
 :count ⟨#⟩)
```

(c) Episode definition template.

Given perceptual inputs and their associated beliefs, the system outputs a dependency graph describing the state. In the case of a schema, the percepts are represented as Naive Bayes classifiers, where the class node is the percept name, and the attributes are all child nodes of the class node. Beliefs are represented as mixtures where the root node is a selector for a specific disjunction of a belief. If a belief interacts with percepts and lower-level beliefs, the system adds a directed edge connecting the belief to the affected variable node. For episodes, percepts and beliefs are represented using the same structures, but the distributions captured by the network are deterministic. Therefore, in this degenerate case, the episode representation is a simply a dependency graph.

The event memory organizes episodes and schemas into a general-to-specific hierarchical taxonomy, where leaf-level elements are episodes and layers of event schema exist above them. Each event memory element connects to its parent via an IS-A link. Now, considering this discussion on memories and knowledge structures in the architecture, we move to study the architectural processes that operate over them.

### 4.3.2 Architectural Processes

The ICARUS cognitive architecture operates in cycles. These cycles begin when environmental perceptions are posited in the perceptual buffer, as shown in Figure 4.1. From there, the system engages its conceptual inference engine to infer a belief state. The inference process occurs in a bottom-up manner. Given perceptual inputs, the system infers all the relevant primitive beliefs by matching perceptual conditions and testing against the matched variables. Then the system infers non-primitive concepts at increasingly higher levels in the concept hierarchy with each new inference dependent on the previously activated beliefs and percepts at the lower levels.

After inferring the belief state, ICARUS nominates one or more goals to pursue by testing the nomination rules in its long-term goal memory against the matched beliefs. The system only nominates and prioritizes goals that are relevant to the current situation, and all others are retracted. After goal nomination, the architecture may select a skill that satisfies one or more of goals and instantiates it as its intention. If, however, the system's top-level goals did not change from the previous cycle, then the system simply continues executing its previous intention. In this way, skill execution in ICARUS is teleoreactive. When the system reaches an impasse and cannot find any appropriate skills or cannot continue executing, it attempts to find a solution via problem solving. This involves recursively identifying skills that satisfy a subset of goals and backward chaining off unsatisfied preconditions (i.e., subgoals) until the system finds a skill executable from the current state. Once the system successfully executes such plans generated, it saves them as new higher-level skills, which serves to eliminate search in the future.

After ICARUS selects an intention, but before submitting control inputs to the motor buffer, the contents of the working memory, which entails the perceptual buffer, belief memory, and goal memory, are stored as episodes into the event memory. The system inserts the newly formed episode by sorting it through the existing event hierarchy along the path most similar to the new episode. At each level, the system incrementally updates the selected schema to accommodate the new instance. After insertion, the system carries out the action prescribed by the executing intention in the environment, thus causing changes to occur. The architecture continues operating in this way until it satisfies all top-level goals or until the end-user terminates the program.

This completes the review of the ICARUS cognitive architecture. We next describe the Minecraft domain which will serve as our test environment for building intelligent agents.

## 4.4 Minecraft Domain Description

Cognitive architectures are unified theories of intelligent behavior. As such, they integrate many different functional components of the mind to produce artifacts of cognition. Because cognitive architectures feature a suite of tightly integrated memories and mental processes, it is not straightforward to evaluate them. Therefore, it is beneficial to use a rich testing environment that supports experiments that can give systems-level insights into intelligent agents built with such architectures. We are especially interested in event memory-enabled agents that rely on their past experiences to solve problems, so it is essential to go beyond simple domains and utilize a domain that affords this capability. In particular, we desire a domain that permits an agent to pursue high-level goals, execute multi-step plans, react to environmental dynamics, and co-operate with humans and artificial counterparts.

One such environment is the popular videogame Minecraft (Johnson *et al.*, 2016). In this open-world game, players can explore a 3D procedurally generated block-like environment in which they can collect resources, build structures, discover new lands, and more. Figure 4.2 shows an example scene from Minecraft displaying the blocky world. The game supports two different types of objects, blocks and entities. Blocks are the building materials which compose structures in the environment. There are many variations of blocks in the game, such as stone, grass, dirt, and bedrock, each having different physical characteristics. They can be organized to construct the physical environment. For example, blocks can be arranged to form roads, buildings, trees, rivers, and mountains. Next, entities are special kinds of blocks that represent moving objects. There are a several types of entities, but all of them contain numerical attributes for position, rotation, and velocity. Specialized entities, such as players, and mobs have additional attributes describing their health points, for example. Example entities include human and artificial agents, drops from defeated enemies such as XP Orbs, and any items found in the player's inventory. Minecraft is ideal for our purposes because of the richness of the domain. The agents live in an interactive and dynamic environment where changes to the world can happen independently of the agent.

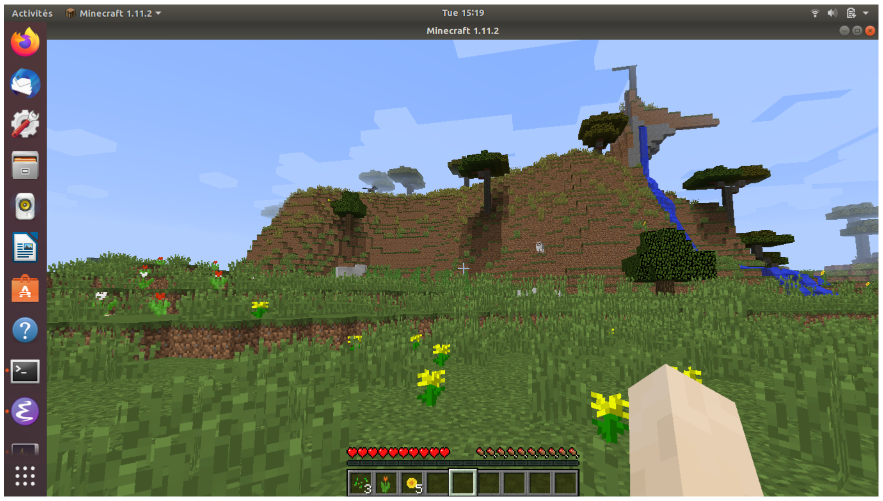


Figure 4.2: Typical Minecraft Environment.

Intelligent agents can cooperate with human and non-human players to solve complex problems. Furthermore, the complex nature of the game forces the agents to construct robust plans and rely on their ability to adapt to changes in the environment. This yields a fertile ground for building event memory-enabled agents that can remember their history of interaction with the environment and leverage those recollections to improve their problem-solving and interactive capabilities. In the next section, we describe an agent that connects to this environment and can build tools. Although we do not evaluate the event memory capabilities of the agent, we believe that the environment does support such evaluations for future experiments.

## 4.5 ICARUS Agent in Minecraft

Intelligent agents can play Minecraft through the Project Malmo mod (Johnson *et al.*, 2016) which provides channels for agents to sense the world and effect change through actions. This mod is designed to support research in cognitive systems, reinforcement learning, and deep learning and supports the testing and evaluation of AI agents by providing mechanisms for defining scenarios and various evaluation metrics. Minecraft scenarios are initialized via an XML file containing descriptions of the blocks composing the world, the numbers of playable agents and other entities,

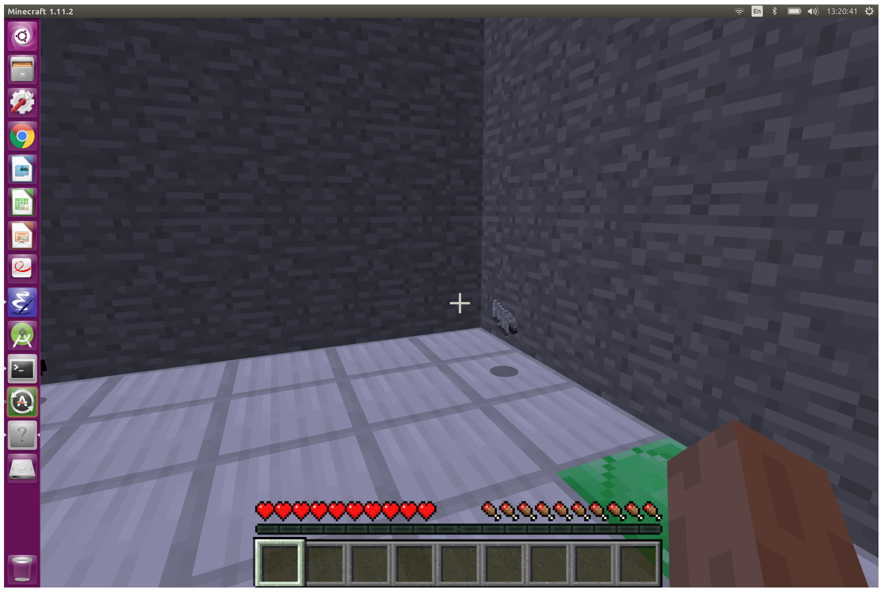


Figure 4.3: Tool-building scenario in Minecraft.

as well as any reward signals and stopping criteria for the scenario. For this work in particular, we designed a small tool-building scenario, as shown in Figure 4.3. An ICARUS agent exists in a room where each corner of the room contains a specific component for building either an arrow or a torch. The agent receives a goal to construct one of the tools from the user and walks around the room, collecting the relevant items before finally crafting the tool.

### 4.5.1 Perceptual Inputs and Effecting Change

During run-time, the agent is able to perceive all entities in the room, as well as the first nine items in its possession, stored in what Minecraft calls the *hotbar*. Observations from the hotbar include the slots of the hotbar, the types of objects in those slots, and the quantity of elements in each slot. Table 4.2 shows some sample objects and their attributes perceived as by the agent. The ICARUS agent also receives information about itself, such as its position and orientation.

Additionally, ICARUS can control the agent by submitting commands through the mod. The commands are interpreted by the Minecraft game engine to change the state of the world. ICARUS can submit commands for movement, crafting, and inventory management. In this scenario, move-

Table 4.2: Sample perceptual patterns in the Minecraft Domain.

```
(hotbar inventory-slot1 type air location 0 size 0)
(hotbar inventory-slot2 type stick-item1 location 1 size 4)
(self self1 x 12 y -21)
(stick stick1 x 2 y 0)
(flint flint1 x 12 y -21)
(feather feather1 x 21 y 0)
(coal coal1 x 22 y 15)
```

ments are discrete and the commands are unary functions named *move*, *turn*, and *strafe*. These functions accept arguments which are either $-1$ or 1. For example, $move(-1)$ makes the player move backward one step and $move(1)$ makes the player move forward one step. The turn and strafe commands work in a similar fashion. The simple craft command allows ICARUS to create new items from existing ones in its inventory. This command accepts one argument, which is the item to be crafted. If the agent possesses all the necessary items in the inventory before issuing this command, then the new item will be created instantaneously. Finally, the inventory commands allow the agent to manipulate the items in its inventory slots. This is useful for selecting which item the agent should hold in its hand.

### 4.5.2 Programmed Agent Knowledge and Demonstrations

The agent was assigned the task of crafting arrows and torches, which requires locating the component pieces of these items, moving to collect those components while avoiding picking up unwanted ones, and finally crafting the item. Figure 4.4 shows that an arrow requires one flint, one feather, and one stick while a torch requires a coal item and a stick. The multi-step building process combined with the navigational requirements in this scenario make the task of building tools cognitively complex. At any given moment, multiple routes exist for collecting components, and the agent may choose to collect them in any order it prefers.

In order to generate reasonable behavior, the agent requires some domain knowledge with which to reason about the world of Minecraft. We gave the agent 14 concepts with seven of them being

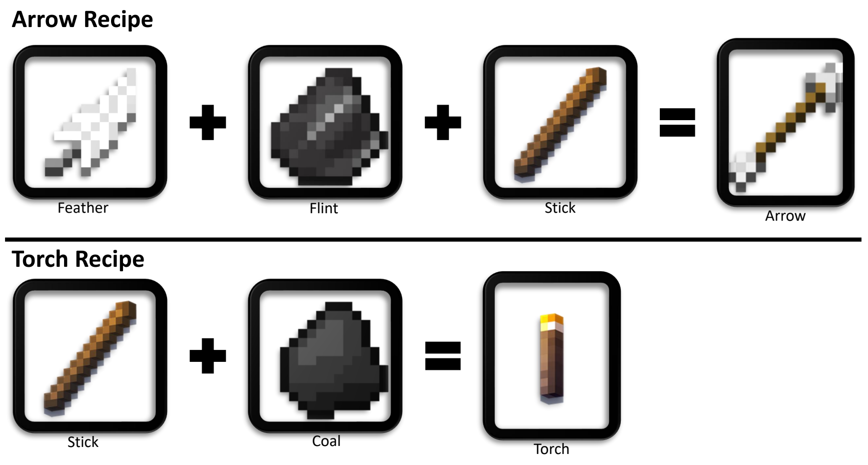


Figure 4.4: Recipes for crafting a torch and an arrow.

primitive concepts. Table 4.3 lists some sample concept definitions describing taxonomic and spatial relations. The first four concepts state that `stick`, `coal`, `flint`, and `feather` entities are all resources. Next, the `(carrying)` concept describes a situation when an item is in the agent's hotbar. Lastly, `(on_vertical_axis)` is true whenever a resource, is on the agent's Y-axis. In a similar fashion, `(on_horizontal_axis)` is true whenever the resource lies on the agent's X-axis.

Accompanying these concept definitions we authored 25 skills, including 15 primitive skills. Table 4.4 shows a sampling of these which allow ICARUS to accomplish different tasks in Minecraft. The `(gather_resource)` skill describes how to gather any given resource, given that the agent is not already carrying it. It simply needs to execute the subskill `(walk_to)`, and that will achieve the carrying concept. Next, `(walk_to)` requires that the resource of interest be in front of and to the left of the agent. In this case, walking to the resource requires moving forward, turning left, then moving forward until the effect is achieved. In this case, the effect states that the resource is not in front of or behind of or left of or right of the agent. This occurs whenever the agent is within one unit of the resource of interest and Minecraft removes the resource from the world positing an item in the agent's inventory. In our agent program, we defined eight different variations of `(walk_to)` each having a unique set of preconditions specifying the appropriate spatial relations between the agent and the resource to pick up. The last two skills in the table `(craft_arrow)` and

Table 4.3: Sample ICARUS concepts for the Minecraft domain.

```
((resource ?o1 ^type stick ^x ?x ^y ?y)
  :elements ((stick ?o1 x ?x y ?y)))

((resource ?o1 ^type coal ^x ?x ^y ?y)
  :elements ((coal ?o1 x ?x y ?y)))

((resource ?o1 ^type flint ^x ?x ^y ?y)
  :elements ((flint ?o1 x ?x y ?y)))

((resource ?o1 ^type feather ^x ?x ^y ?y)
  :elements ((feather ?o1 x ?x y ?y)))

((carrying ?o1 ^type ?type ^location ?loc ^size ?size)
  :elements ((hotbar ?o1 type ?type location ?loc size ?size))
  :tests    ((> ?size 0)))

((on_vertical_axis ?o1 ?self)
  :elements   ((self ?self))
  :conditions ((resource ?o1)
               (not (right_of ?o1 ?self))
               (not (left_of ?o1 ?self))
               (not (carrying ?o1))))

((on_horizontal_axis ?o1 ?self)
  :elements ((self ?self))
  :conditions ((resource ?o1)
               (not (front_of ?o1 ?self))
               (not (behind_of ?o1 ?self))
               (not (carrying ?o1))))
```

(craft_torch) describe how to achieve the goal of carrying a torch or an arrow. In both skills, the order in which the agent collects the components does not affect its ability to construct the items, so given different variations of these skills each with a unique permutation of the subskills, the system can decide which disjunction to fire.

Now that we have covered how the agent perceives the world, makes inferences about relations, and execute skills to achieve goals, we present a couple of demonstrations of the system in action. Figure 4.5 depicts the execution of the agent achieving (carrying arrow) in the right pane. From the starting position, the agent turns right 90 degrees, then moves forward to the stick shown at the bottom right corner. After completing that step, the agent turns left 90 degrees to face the flint, and moves to it, collecting it in the inventory. The agent turns left once more, but noticing that the coal is obstructing its path to the feather, the agent chooses to strafe to the left until the coal is no longer on the agent's vertical axis. Then, the agent moves forward until the feather sits along the agent's horizontal axis. Finally, the agent turns left, then moves forward to

Table 4.4: Sample ICARUS skills for the Minecraft domain.

```
((gather_resource ?o1 ?t)
  :conditions ((resource ?o1 ^type ?t)
               (not (carrying ?item ^type ?t)))
  :subskills  ((walk_to ?o1))
  :effects    ((carrying ?o1_item ^location ?l ^type ?t ^size ?s)))

((walk_to ?o1)
  :elements   ((self ?self))
  :conditions ((resource ?o1 ^x ?x1 ^y ?y1)
               (front_of ?o1 ?self)
               (left_of ?o1 ?self)
               (not (on_horizontal_axis ?o1 ?self))
               (not (on_vertical_axis ?o1 ?self)))
  :subskills  ((move_forward ?o1)
               (turn_left ?o1)
               (move_forward ?o1))
  :effects    ((not (front_of ?o1 ?self))
               (not (behind_of ?o1 ?self))
               (not (right_of ?o1 ?self))
               (not (left_of ?o1 ?self))))

((craft_torch)
  :conditions ((resource ?o2 ^type coal)
               (resource ?o3 ^type stick))
  :subskills  ((gather_resource ?o3 stick)
               (gather_resource ?o2 coal)
               (make_torch))
  :effects ((carrying ?torch ^type torch ^location ?l ^size ?s)))

((craft_arrow)
  :conditions ((resource ?o2 ^type stick)
               (resource ?o3 ^type flint)
               (resource ?o4 ^type feather))
  :subskills ((gather_resource ?o2 stick)
              (gather_resource ?o3 flint)
              (gather_resource ?o4 feather)
              (make_arrow))
  :effects ((carrying ?arrow ^type arrow ^location ?l ^size ?s)))
```

collect the `feather`. Having obtained all requisite components for the arrow, the agent executes `(make_arrow)`, and crafts the desired item.

The left pane of Figure 4.5 traces the execution for achieving `(carrying torch)`. In this example, the agent begins by turning right and moving to the `stick`, similar to the previous case. Then, it turns left, but before moving forward, the agent recognizes that the `feather` is blocking its path, so it strafes to the left so the path is clear. When the agent does not perceive an obstruction in the path, it moves forward to align the `coal` with its horizontal axis. After achieving that subgoal, the agent turns left and continues toward the `coal`. When the `coal` enters the agent's inventory, the agent executes `(make_torch)` and obtains the torch, thereby completing the run.

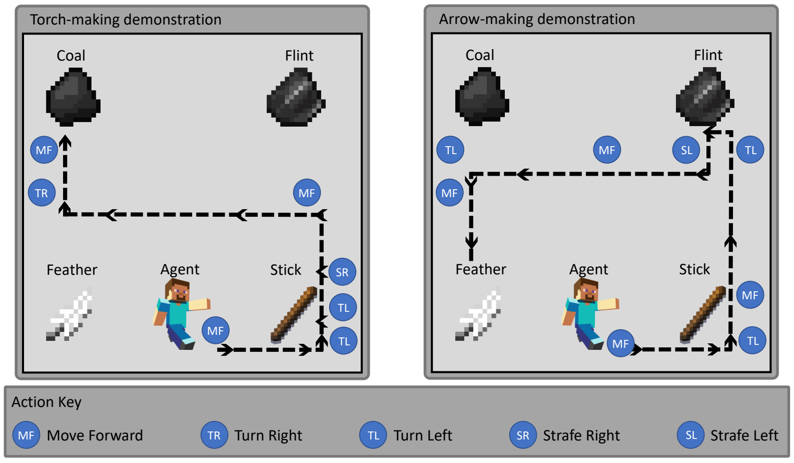


Figure 4.5: Tool-building executions in Minecraft.

## 4.6 Future Work

In the previous section, we demonstrated how our ICARUS agent moves around a room collecting resources to build tools. Our future work includes enabling the agent's event memory module to perform hierarchical goal and intention recognition. During the course of execution, the agent forms hierarchical intentions to accomplish goals and we would like to save these in the agent's event memory for later recall when the agent attempts to recognize the behaviors of other agents. Goal and intention recognition is interesting to us in this domain because many actions are shared across tasks, even in the torch-making and arrow-making tasks. A meaningful recognition system should be able to handle this and reason about which plan is most probable given the sequence of observed actions. Before we can demonstrate such a system, however, there are challenges we need to address.

Most importantly, we need to extend our event memory system to capture the temporal dimension of lived experience. Our work currently only handles remembering individual states without the proper temporal context. Additionally, there are a couple improvements we would like to make to the implementation that will improve the system performance in Minecraft. Our current implementation represents conditional probability distributions as multidimensional tables such that each variable participating in the distribution occupies one dimension. As the system updates these distributions, new variables are introduced thereby increasing the dimensionality of the distributions in an exponential manner. For example, our system represents a distribution with four variables each with a domain of size two with a table having $2^4 = 16$ parameters. A distribution with 30 similar variables will contain $2^{30} = 1,073,741,824$ parameters. Such a number of parameters not only makes learning these distributions more difficult, but also exerts a costly toll on the computer's hardware resources. In order to alleviate this exponential explosion, we would like to exploit the local structure within each distribution by switching to a rule-based representation. Using rules to describe the distributions will allow us to cover the entries of the table with a small set of rules whose left hand sides contain some assignment of relevant variables while the right hand side determines the probability of that assignment. The representational savings can be quite significant whenever one rule satisfies multiple entries in the table.

Another improvement we would like to make to the implementation is to represent continuous distributions along with the discrete ones. Our event memory system treats numerical values as if they were discrete. This limitation required us to enforce discrete movement commands in Minecraft because our system cannot represent a distribution where some variables have infinite domains. To make this change, we will need to utilize hybrid Bayesian networks (Lerner, 2003) that can simultaneously reason about discrete and continuous variables.

More broadly, this research aims to create collaborative interactive agents. Beyond the scope of goal and intention recognition, we would like to enable our agent to decide what to do once it has made inferences about an actor's motivations. Accompanying this, it would be useful if the system had capabilities for dialogue and question answering. A dialogue system could allow agents to ask clarificatory questions about behavior it is observing. As stated earlier in this section, future versions of our agent will store the agent's intentions in memory so the agent could use its dialoguing capabilities to explain its own motivations and actions to human users.

## 4.7 Conclusions

In this chapter, we discussed our efforts to integrate our event memory system within the ICARUS cognitive architecture. We first reviewed the architecture showing how it represents semantic and procedural knowledge, then explained how our event memory system fits into the broader architecture. Next, we discussed the Minecraft domain and why it is suitable for our purposes, and then demonstrated how a Minecraft-playing agent builds complex tools. After this, we discussed our future work involving the event memory capabilities of the agent. This included hierarchical goal and intention recognition as well as other capabilities that interactive, collaborative agents need. We believe the Minecraft domain and the agent we built will allow us to continue our research efforts to build event memory-enabled agents that interact with and cooperate with humans.

# Chapter 5

# Conclusions

In this thesis, we presented a novel hybrid theory of event memory that unifies aspects of causal and simulation theories, and provides a complete account of event memory phenomena. We provided an accompanying implementation of the theory and demonstrated its capabilities to remember events using both episodes and schemas. We also showed that the system produces and accounts for the three categories of event memory phenomena: successful remembering; misremembering; and confabulation. Lastly, we described our efforts to integrate our event memory system into a cognitive architecture, and situate an agent in the Minecraft domain in preparation for future experiments on event memory. The work described here are all important contributions that will help drive continued research in this important area.

## 5.1 Limitations of our Approach

Despite the strengths of our view which we previously outlined, there are some limitations to our approach. One of the longer-term aims of the research is to build collaborative interactive agents. In order to achieve this, one of the requirements of the system is that it operates in real time. Operations conducted by the event memory system currently do not complete in real time. In fact, as the system runs, the system performance slows down exponentially despite relying on a tree-structured event memory store.

We believe the limited system performance is due to an exponential explosion taking place at the representation level. The nodes describing the conditional probability distributions in episodes and schemas are represented using multidimensional tables. A distributions with four variables therefore contains four dimensions. It should be noted that a table with four dimensions requires $2^4 = 16$ parameters, assuming that all the variables are binary. Likewise, a distribution with 30 variables is represented by a 30-dimensional table containing $2^{30} = 1,073,741,824$ parameters. Such a number of parameters not only makes learning these distributions more difficult and time consuming, but also exerts a costly toll on the computer's hardware resources. The challenge that we currently face is that as the event memory system inserts new episodes into memory, it will update the internal probability distributions of the schemas it built. Doing this sometimes requires adding new variables to the existing distributions which translate to exponentially increasing the size of the tables.

This exponential explosion can sometimes be avoided depending on properties of the domain. For example, in an environment where all observed instances belonging to the same class share the same structure, but only vary in the values that the variables take, then this problem is eliminated if the size of each variable is finite. This was the case with our experiments involving recognizing `tower` and `adjacent_to` classes. It is also true of many machine learning settings where the dataset has a fixed number of features, but the values of those features change. Even if the structure of the observations vary significantly, the exponential explosion will only become prohibitive in cases where variables have many parent nodes. Thus if designers minimize the in-degree of the nodes in the network, then the event memory system should work well, even if the performance is not capable of functioning in real-time.

There are some domains, however, that do not share the characteristics of the domains mentioned above and are not amenable to clever design tricks to alleviate the exponential explosion. Minecraft is one such domain. One reason for this is that the agent operating in the world usually needs to reason about spatial relations. Another reason is that the agent interacts with the environment over time, causing many structural changes that distinguish new observations from previous ones. Considering the spatial relations, if the agent moves, then all the spatial relations may change with respect to the agent. For example, an agent's relation to a resource that is initially in front of the agent changes whenever it turns around. The dynamic behavior of the environment results in new variables being added to the event memory representations which try to encapsulate the range of possible outcomes, giving rise to the exponential explosion. Similarly, as the agent manipulates its environment, new artifacts come into and out of existence, further increasing the complexity of the event memory representations.

In addition to the limitations of the table-based representation for conditional probability distributions, the event memory system also cannot handle continuous variables. Instead of parameterizing them with some continuous function, the system treats them as if they are discrete variables. Therefore, the system cannot gracefully handle variables with infinite domains. This can, in turn, exacerbate the problems with using table-based representations. Even if a table contains a few dimensions, if one or more variable are continuously defined, the size of the this table will grow without bound.

## 5.2 Future Work

The limitations described in the previous section naturally give rise to some future work. We would like to move away from table-based representations of conditional probability distributions toward rule-based representations. We will treat the variables and their specific assignments that participate in a distribution as the left-hand side of the rule and the probability assignment as the right-hand side. Doing so will transform each conditional probability distribution into a sub-symbolic production system that will yield advantages and tradeoffs. The biggest advantage is that rules can exploit local structure in the distributions by covering multiple cases in the table. For example, removing redundant conditions from the left hand side will increase the coverage of a rule. Doing this for each rule will eliminate the exponential explosion that is currently being observed, as only a handful of rules will be necessary to capture the behavior of the distribution, even as the system adds new variables to them.

By moving to a rule-based representation, we will forego some of the advantages of table-based representations, namely the constant-time access associated with indexing tables. We accept this tradeoff because even the worst case performance of rule-based representations which require the system to check every rule to find a match to the current situation is better than the exponential behavior we are observing in other parts of the system. Additionally, we could further alleviate matching costs by utilizing generalized tries to store the rules, such as in the Rete algorithm (Forgy, 1989). We believe that after making this set of changes, our event memory system will be capable of running more efficiently.

Another future work responding to the limitations of the system would be to utilize hybrid Bayesian networks that can simultaneously reason about discrete and continuous variables. This change will require significant modifications to the underlying probabilistic inference engine in the system and may even restrict the kinds of network structures the system can handle, depending on our chosen approach.

Beyond these implementational improvements, our goal is to build interactive collaborative agents that can work with people. Toward that end, one performance task that these types of agents will need to do is to infer the goals and intentions of their human counterparts. Earlier in this dissertation, we showed that a Minecraft agent can execute multi-step plans to build tools. During the course of execution, we would like to enable our agent to store such plans in its event memory so that it can retrieve them and make predictions about the goals and intentions of other agents it observes. Because our agent's intention structures are hierarchical, our system would also be able to perform hierarchical goal and intention recognition, and can help provide explanations of behavior at varying levels of abstraction.

In order to properly track the execution of a plan over time, however, we need to extend our event memory representations to capture the temporal dimension of lived experience. Currently, they only capture states of affairs, but will need to encapsulate courses of events. We envision making these changes will result in a representation similar to hierarchical HMMS (Fine *et al.*, 1998) but having ground observations being complete Bayesian networks rather than a handful of variables.

Building intelligent agents that collaborate and interact with humans will require other capabilities, such as dialoguing skills and goal reasoning abilities, that extend beyond the affordances of an event memory system. We would like to pursue these and other capabilities over the course of our larger research agenda. For such pursuits, we will continue to base our work within the context of cognitive architectures which integrate the facets of the mind into one coherent theory of human-level intelligence.

# References

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, 111(4), 1036.

Bernecker, S. (2008). *The Metaphysics of Memory*. Springer Science & Business Media.

Bernecker, S. (2010). *Memory: A Philosophical Study*. Oxford University Press.

Bernecker, S. (2017). A causal theory of mnemonic confabulation. *Frontiers in Psychology*, 8, 1207.

Brom, C., Pešková, K., & Lukavskỳ, J. (2007). What does your actor remember? towards characters with a full episodic memory. In *Proceedings of the Fourth International Conference on Virtual Storytelling* (pp. 89–101).: Springer.

Carbonell, J., Etzioni, O., Gil, Y., Joseph, R., Knoblock, C., Minton, S., & Veloso, M. (1991). Prodigy: An integrated architecture for planning and learning. *ACM SIGART Bulletin*, 2(4), 51–55.

Choi, D. & Langley, P. (2018). Evolution of the ICARUS cognitive architecture. *Cognitive Systems Research*, 48, 25–38.

Collins, R. N., Milliken, B., & Jamieson, R. K. (2020). Minerva-de: An instance model of the deficient processing theory. *Journal of Memory and Language*, 115, 1–13.

Debus, D. (2007). Perspectives on the past: A study of the spatial perspectival characteristics of recollective memories. *Mind & language*, 22(2), 173–206.

Debus, D. (2010). Accounting for epistemic relevance: A new problem for the causal theory of memory. *American Philosophical Quarterly*, 47(1), 17–29.

Debus, D. (2018). Handle with care: Activity, passivity, and the epistemological role of recollective memories. In K. Michaelian, D. Debus, & D. Perrin (Eds.), *New Directions in the Philosophy of Memory* (pp. 119–136). Routledge.

Dias, J., Ho, W., Vogt, T., Beeckman, N., Paiva, A., & André, E. (2007). I know what i did last summer: Autobiographic memory in synthetic characters. *Affective Computing and Intelligent Interaction*, (pp. 606–617).

Doshi, J., Kira, Z., & Wagner, A. (2015). From deep learning to episodic memories: Creating categories of visual experiences. In *Proceedings of the Third Annual Conference on Advances in Cognitive Systems* (pp.$\tilde{1}$5).

Eichenbaum, H. (2017). Prefrontal–hippocampal interactions in episodic memory. *Nature Reviews Neuroscience*, 18(9), 547–558.

Falkenhainer, B., Forbus, K. D., & Gentner, D. (1989). The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, 41(1), 1–63.

Fikes, R. & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 189–208.

Fine, S., Singer, Y., & Tishby, N. (1998). The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, 32(1), 41–62.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2), 139–172.

Forbus, K. D., Gentner, D., & Law, K. (1995). MAC/FAC: A model of similarity-based retrieval. *Cognitive Science*, 19(2), 141–205.

Forgy, C. L. (1989). Rete: A fast algorithm for the many pattern/many object pattern match problem. In *Readings in Artificial Intelligence and Databases* (pp. 547–559). Elsevier.

Gennari, J. H., Langley, P., & Fisher, D. (1989). Models of incremental concept formation. *Artificial Intelligence*, 40(1-3), 11–61.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2), 155–170.

Gluck, M. A. & Corter, J. E. (1985). Information, uncertainty and the utility of categories. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society* (pp. 283–287).

Hellerstedt, R. (2015). *From Cue to Recall: The Temporal Dynamics of Long-Term Memory Retrieval*. PhD thesis, Lund University.

Hintzman, D. L. (1984). Minerva 2: A simulation model of human memory. *Behavior Research Methods, Instruments, & Computers*, 16(2), 96–101.

Hintzman, D. L. (1986). "schema abstraction" in a multiple-trace memory model. *Psychological Review*, 93(4), 411.

Hintzman, D. L. & Ludlam, G. (1980). Differential forgetting of prototypes and old instances: Simulation by an exemplar-based classification model. *Memory & Cognition*, 8(4), 378–382.

Ho, W. C. & Dautenhahn, K. (2008). Towards a narrative mind: The creation of coherent life stories for believable virtual agents. In *International Workshop on Intelligent Virtual Agents* (pp. 59–72).: Springer.

Horn, A. (1951). On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, 16(1), 14–21.

Jefferys, W. H. & Berger, J. O. (1992). Ockham's razor and bayesian analysis. *American Scientist*, 80(1), 64–72.

Johnson, M., Hofmann, K., Hutton, T., & Bignell, D. (2016). The Malmo platform for artificial intelligence experimentation. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence* (pp. 4246–4247).

Jolliffe, I. (2002). *Principal Component Analysis*. Verlag, NY: Springer.

Jonyer, I., Cook, D. J., & Holder, L. B. (2001). Graph-based hierarchical conceptual clustering. *Journal of Machine Learning Research*, 2(Oct), 19–43.

Kelley, T. D. (2014). Robotic dreams: A computational justification for the post-hoc processing of episodic memories. *International Journal of Machine Consciousness*, 6(02), 109–123.

Koller, D. & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Kolodner, J. L. (1983). Reconstructive memory: A computer model. *Cognitive Science*, 7(4), 281–328.

Laird, J. E. (2012). *The Soar Cognitive Architecture*. MIT Press.

Lebowitz, M. (1987). Experiments with incremental concept formation: Unimem. *Machine Learning*, 2(2), 103–138.

Lerner, U. N. (2003). *Hybrid Bayesian networks for reasoning about complex systems*. stanford university.

Liang, C. & Forbus, K. (2014). Constructing hierarchical concepts via analogical generalization. In *Proceedings of the Annual Meeting of the Cognitive Science Society*.

Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2), 129–137.

Lopez-Paz, D. & Ranzato, M. (2017). Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems* (pp. 6467–6476).

MacLellan, C. J., Harpstead, E., Aleven, V., & Koedinger, K. R. (2015). Trestle: Incremental learning in structured domains using partial matching and categorization. In *Proceedings of the Third Annual Conference on Advances in Cognitive Systems*.

Martin, C. B. & Deutscher, M. (1966). Remembering. *The Philosophical Review*, 75(2), 161–196.

McLure, M. D., Friedman, S. E., & Forbus, K. D. (2010). Learning concepts from sketches via analogical generalization and near-misses. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 32.

Ménager, D. (2016). Episodic memory in a cognitive model. In *Proceedings of the Twenty-Fourth International Conference on Case-Based Reasoning* (pp. 267–271). Atlanta, GA.

Ménager, D. (2018). Episodic memory foundation of explainable autonomy. In *Proceedings of the Twenty-Sixth International Conference on Case-Based Reasoning* (pp. 32–41). Stockholm, Sweeden.

Ménager, D. & Choi, D. (2016). A robust implementation of episodic memory for a cognitive architecture. In *Proceedings of the Thirty-Eighth Annual Meeting of the Cognitive Science Society*.

Ménager, D., Choi, D., Roberts, M., & Aha, D. W. (2018). Learning planning operators from episodic traces. In *2018 AAAI Spring Symposium Series*.

Ménager, D. H., Choi, D., Floyd, M. W., Task, C., & Aha, D. W. (2017). Dynamic goal recognition using windowed action sequences. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*.

Ménager, D. H., Choi, D., & Robins, S. K. (2021a). A hybrid theory of event memory. *Minds and Machines (under review)*.

Ménager, D. H., Choi, D., & Robins, S. K. (2021b). Modeling human memory phenomena in a hybrid event memory system. *Cognitive Systems Research (under review)*.

Michaelian, K. (2011). Generative memory. *Philosophical Psychology*, 24(3), 323–342.

Michaelian, K. (2016a). Confabulating, misremembering, relearning: The simulation theory of memory and unsuccessful remembering. *Frontiers in Psychology*, 7, 1857.

Michaelian, K. (2016b). *Mental Time Travel: Episodic Memory and Our Knowledge of the Personal Past*. MIT Press.

Michaelian, K. (2020). Confabulating as unreliable imagining: In defence of the simulationist account of unsuccessful remembering. *Topoi*, 39(1), 133–148.

Michaelian, K. & Robins, S. K. (2018). The causal theory of memory. In K. Michaelian, D. Debus, & D. Perrin (Eds.), *New Directions in the Philosophy of Memory* (pp. 13–32). Routledge.

Mirsky, R., Stern, R., Gal, K., & Kalech, M. (2018). Sequential plan recognition: An iterative approach to disambiguating between hypotheses. *Artificial Intelligence*, 260, 51–73.

Moscovitch, M., Cabeza, R., Winocur, G., & Nadel, L. (2016). Episodic memory and beyond: the hippocampus and neocortex in transformation. *Annual Review of Psychology*, 67, 105–134.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.

Nuxoll, A. & Laird, J. E. (2004). A cognitive model of episodic memory integrated with a general cognitive architecture. In *Proceedings of the Sixth International Conference on Cognitive Modelling* (pp. 220–225).: Citeseer.

Nuxoll, A. M. & Laird, J. E. (2012). Enhancing intelligent agents with episodic memory. *Cognitive Systems Research*, 17, 34–48.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

Pei, M., Yunde Jia, & Zhu, S. (2011). Parsing video events with goal inference and intent prediction. In *2011 International Conference on Computer Vision* (pp. 487–494).

Reynolds, D. (2009). *Gaussian Mixture Models*, (pp. 659–663). Springer US: Boston, MA.

Robins, S. (2020). Mnemonic confabulation. *Topoi*, 39(1), 121–132.

Robins, S. K. (2016). Misremembering. *Philosophical Psychology*, 29(3), 432–447.

Robins, S. K. (2019). Confabulation and constructive memory. *Synthese*, 196(6), 2135–2151.

Rubin, D. C. & Umanath, S. (2015). Event memory: A theory of memory for laboratory, autobiographical, and fictional events. *Psychological Review*, 122(1), 1–23.

Schachter, D. L. (1987). Implicit memory: History and current status. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 13(3), 501–518.

Schacter, D. L. & Addis, D. R. (2007a). The cognitive neuroscience of constructive memory: remembering the past and imagining the future. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 362(1481), 773–786.

Schacter, D. L. & Addis, D. R. (2007b). On the constructive episodic simulation of past and future events. *Behavioral and Brain Sciences*, 30(3), 331–332.

Stachowicz, D. & Kruijff, G.-J. M. (2011). Episodic-like memory for cognitive robots. *IEEE Transactions on Autonomous Mental Development*, 4(1), 1–16.

Suddendorf, T., Addis, D. R., & Corballis, M. C. (2009). Mental time travel and the shaping of the human mind. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 364(1521), 1317–1324.

Thompson, K. & Langley, P. (1991). Concept formation in structured domains. In *Concept Formation* (pp. 127–161). Elsevier.

Tulving, E. (1983). *Elements of Episodic Memory*. Oxford University Press.

Tulving, E. (2002). Episodic memory: From mind to brain. *Annual Review of Psychology*, 53(1), 1–25.

Wasserman, K. (1985). *Unifying representation and generalization: Understanding hierarchically structured objects*. PhD thesis, Columbia University.

Wells, G. L. (1982). Attribution and reconstructive memory. *Journal of Experimental Social Psychology*, 18(5), 447–463.

Winograd, T. (1971). *Procedures as a representation for data in a computer program for understanding natural language*. Technical report, Massachusetts Institute of Technology.

Yonelinas, A. P., Ranganath, C., Ekstrom, A. D., & Wiltgen, B. J. (2019). A contextual binding theory of episodic memory: systems consolidation reconsidered. *Nature Reviews Neuroscience*, 20(6), 364–375.