Episodic Memory: Foundation of Explainable Autonomy

By

David Ménager

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

Committee members

Arvin Agah, Co-chairperson

Dongkyu Choi, Co-chairperson

Andrew Williams, Member

Michael Branicky, Member

Date defended: _____

The Thesis Committee for David Ménager certifies
that this is the approved version of the following thesis :

Episodic Memory: Foundation of Explainable Autonomy

Arvin Agah, Co-chairperson

Date approved: _____

Abstract

In recent years, the inner workings of many intelligent agents have become opaque to users who wish to manage or collaborate with them. This lack of transparency makes it difficult for human users to understand and predict the behavior of such agents. We argue that computational agents that store the plans they construct can behave in a predictable and transparent manner by remembering the plans used for achieving goals and explaining their contents to users.

To investigate this issue, we present a psychologically inspired computational theory of episodic memory that explains how intelligent agents can use their personal experience to make known their internal decision-making process. We augment this theory with an implementation and show how systems with episodic memory capabilities can explain what happened in their personal past. We demonstrate this system's ability to answer questions in two Minecraft scenarios. Our preliminary findings suggest that episodic memory capabilities in computational agents plays an important role in producing explanations regarding an agent's cognitive and behavioral abilities. With continued research, we believe our approach can facilitate the harmonious integration of robots with their human counterparts, creating an environment where humans and artificial agents better understand each other.

Acknowledgements

I would like to thank God and my family for supporting and encouraging me through my Master's degree program. I would like to thank my advisor, Dongkyu Choi, for guiding me during this journey and turning me into a researcher. I would also like to thank Pat Langley for his insightful comments and guidance. David W. Aha and Mark Roberts at the Naval Research Laboratory provided insightful comments, for which I am thankful.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This research proposes to use a computational model of episodic memory as a core technology for artificial agents that can explain their own behavior. Episodic memory is the memory responsible for enabling one to remember the events of his or her life with a subjective sense of time, a sense of self, and autonoetic[1] consciousness (Tulving, 1985). In this thesis, we hypothesize that the capacity to remember the personal past using episodic memory plays an essential role in *explainable autonomy*, which enables artificial systems to communicate their motivations and reasonings to their end-users (Gunning, 2017). We present a theory that explains how episodic memory and its related processes allow an agent to summarize its experience, answer questions about its past, and engage in hypothetical thinking.

Many mainstream artificial intelligence systems today operate as black boxes that have little to no ability to explain their rationale to users. This limits the effectiveness of intelligent agents in the world, because their human users cannot predict or trust their behavior. An explainable agent will help remedy this issue by providing justifications for its behavior and allowing users to understand and predict its decision-making processes. We argue that such explainability is necessary because the agent's rationale needs to be well understood by users who wish to manage or collaborate with it. To demonstrate this capacity, an agent must be able to:

- Summarize its personal history at appropriate levels of abstraction;
- Explain why and when it chose the goals it pursued;
- Explain how the goals were achieved;

[1] Autonoesis allows humans to mentally place themselves forwards or backwards in time.

- Discuss how actions were executed in the world;
- Provide details on alternative options for achieving goals; and
- Expose any failures or difficulties it faced during planning or execution.

These requirements allow a user to understand and predict an agent's internal decision-making process, increasing the user's confidence and trust in the agent's behavior. Moreover, because humans understand and use social cues and adhere to societal norms, computational agents should model this capability, when interacting with humans. It should be natural for a human to interact with such agents, meaning that humans should not have to learn new methods for communicating with these agents. Our system uses a text-based mode of communication, but yet, adopts human-like representations and knowledge, adheres to some social norms, and reasons about the beliefs and goals of the users it is helping. To that end, we believe a cognitive systems approach is ideal to achieve a high level of explainable autonomy because these systems commit to human-like notions of beliefs, desires, and intentions. Additionally, it is desirable for explainable agents to receive and apply feedback from users about their decision-making process.

This thesis includes our previous work where we developed episodic memory for a cognitive architecture (Ménager & Choi, 2016), and characterizes the role of this memory in facilitating explainable autonomy. Despite the fundamental importance of this memory, computational models of episodic memory are not discussed very frequently, aside from some recent work (Nuxoll & Laird, 2007; Faltersack et al., 2011; Bölöni, 2011). In this research, we borrow insights from psychology and cognitive science literature to produce an integrated episodic memory module within a cognitive architecture, ICARUS (Langley & Choi, 2006). Using this new cognitive faculty, we built an agent that demonstrate explainable behavior.

In the remainder of this thesis, we first introduce the computer game we use as our domain. Next, we review the ICARUS architecture briefly. Then, describe our episodic memory extension within this system, addressing how to represent, organize, and retrieve experiences in this new memory. Then, we discuss our episodic memory extension, covering the memory representations and processes. Next, we present an ICARUS agent with explainable autonomy capabilities enabled

by episodic memory, followed by demonstrations of this agent's ability to summarize its personal past, answer questions, and engage in hypothetical thinking within a computer game. Next, we discuss related, then discuss future work before we conclude.

Chapter 2

Illustrative Domain

Cognitive architectures are unified models 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, using a rich and dynamic domain like the game Minecraft (Johnson et al., 2016) can support a systems-level understanding of these intelligent agents. Moreover, our aim is to produce a psychologically plausible system that interacts with humans in physical environments, so we need to go beyond simple domains and utilize a domain that affords many of the cognitive functions that ICARUS provides. In particular, we desire a domain that permits an agent to pursue high-level goals as well as exert low-level control in a physical environment.

2.1 The Minecraft Domain

Minecraft is a popular computer game that provides such an environment. In this open-world game, players can create and experience new environments in which they can collect resources, build structures, discover new lands, and more. Figure 2.1 shows an example scene from Minecraft where the world is composed of blocks. Blocks can be of different kinds, such as stone, grass, dirt, and bedrock, each having different physical properties. These objects can be organized to construct the physical environment. For example, blocks can be arranged to form roads, buildings, trees, rivers, and mountains. Entities are special kinds of blocks that represent moving objects with numeric attributes such as position, rotation, and velocity. Entities can also have a health attribute to represent the quality of their physical integrity. Example entities include players, mobs, boats,

Figure 2.1: Example scene from the Minecraft domain.

and items. Some of the entities like zombies, creepers, and spiders are hostile and can attack players. When such entities die, they drop items which players and other agents can collect. Items can also appear as a result of actions like mining.

Minecraft is ideal for our purposes because of the richness of the domain. The agents live in an interactive and dynamic environment where events happen outside of the agent's control. They can cooperate with human and non-human players to solve complex problems. Furthermore, the dynamic nature of the game forces the agents to construct robust plans and rely on their ability to adapt to changes in the environment. The Minecraft world is also a continuous space, and the agents need to be able to reason about continuous events.

2.2 Connecting ICARUS to the Minecraft Domain

Minecraft scenarios are initialized by a mission XML file. Using this file, the game engine establishes a server port that accepts connections from ICARUS agents. During initialization, the system sets the initial properties of the world like the topography of the terrain, the location of items, and the agent's starting position and possessions. The system is also responsible for determining which

Table 2.1: Sample perceptual patterns in the Minecraft Domain.

```
(hotbar inventory-slot1 type air location 0 size 0 belongs-to self1)
(hotbar inventory-slot2 type stick-item1 location 0 size 4 belongs-to self1)
(self self1 action nil x 12.48 y -21.16 life 20 orientation 0.0)
(cooked_porkchop food1 x 2.27 y 0.56 z .53 orientation 242.93)
(planks planks1 x 12.48 y -21.11 z ?z 0.0 orientation 0.0)
(rotten_flesh rotten_flesh1 x 21.37 y .21 z .35 orientation 73.55)
(xporb xporb1 x 22.15 y 15.02 z 1.13 orientation 30.21)
(zombie zombie1 x 11.90 y -18.49 z 0.00 orientation 184.76 life 20)
(slime slime1 x 12.95 y 43.32 z 0.0 orientation 20.45 life 10)
```

perceptions and commands are available to the agent.

During run-time, ICARUS controls a single player, for which it receives sensory input from the environment. The agent is able to perceive nearby entities within a 10×10 grid centered around it, as well as the first eight items in its possession, stored in what Minecraft calls the *hotbar*. Table 2.1 shows some sample objects and their attributes perceived by the agent. ICARUS receives information about itself like, its current action, position, and orientation. It also perceives items in its hotbar, including the slots of the hotbar, the types of objects in the slots, the quantity of elements in each slot, and the owner of the hotbar items.

Additionally, ICARUS can control the player by submitting commands through the server connection. The commands are interpreted by the Minecraft game engine to change the state of the world. ICARUS can submit commands for movement, chatting, crafting, and inventory management. The movement commands are unary functions named *move*, *turn*, and *strafe*. These functions accept a decimal value between -1 and 1 to specify the degree to which that command is to be executed. For example, $move(-1)$ makes the player move backward at full speed, $move(0)$ stops the player, and $move(1)$ makes the player move forward. The turn and strafe commands work in a similar fashion. The chat command allows the agent to output messages onto the screen to communicate with other players. It is a unary function that accepts a string message as its argument. 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 switching the object in the agents hand, whether it is a sword or another object.

In summary, Minecraft is a rich and dynamic domain to test our system for explainable autonomy. It allows researchers to construct a wide array of interesting scenarios, and enables the evaluation of explainable agents. For this reason, we use this domain to demonstrate how our system enables building explainable agents, as will be shown in the later chapters. Before we do that, however, we briefly review the ICARUS architecture as a reference.

Chapter 3

ICARUS Review

As a cognitive architecture, ICARUS (Choi & Langley, 2018) provides an infrastructure for modeling human cognition and programming intelligent agents. The architecture makes specific commitments to its representation of knowledge, the memories that store these contents, and the processes that operate over them. ICARUS shares some of these commitments with other architectures like Soar (Laird, 2012a) and ACT-R (Anderson & Lebiere, 1998), but it also has distinct characteristics that differentiate it from those architectures, including its architectural emphases on hierarchical knowledge structures, teleoreactive execution, and goal reasoning.

In this chapter, we review the ICARUS architecture that we use as a basis for our work on episodic memory and explainable autonomy. We begin by discussing the representation and memories in ICARUS. Then, we continue to the processes that operate on these memories as part of a cognitive cycle in the architecture.

3.1 Representation and Memories

ICARUS distinguishes two main types of knowledge, *concepts* and *skills*, which represent semantic and procedural knowledge, respectively. Concepts describe certain aspects of a situation in the environment. They resemble horn clauses (Horn, 1951), complete with a predicate as the head, perceptual matching conditions, tests against matched variables, and references to any subrelations. A *primitive concept* is defined directly over a set of objects and their attributes, while a *non-primitive concept* describes more complex situations, specifying how other concepts and perceived objects relate to each other.

Table 3.1 shows some sample ICARUS concepts for Minecraft. The first one, *carrying*, is a primitive concept that describes when the agent is carrying something in its inventory. It requires perceptual elements for the agent itself, *?self* and the inventory, *hotbar*. The second concept, *front-of*, is a non-primitive concept that describes the situation where an entity is in front of the agent. It requires a perceptual match against the agent itself, a sub-relation (entity *?o1*), and a test over the positions of the two. The last concept definition, *on-horizontal-axis*, is another non-primitive concept that uses both of the earlier definitions as sub-relations to define the situation where the entity is not being held by the agent and is on the same horizontal line as the agent.

Table 3.1: Sample ICARUS concepts for the Minecraft domain.

```
((carrying ?o1 ?o2 ^type ?type ^location ?loc ^size ?size)
 :elements ((self ?o1)
            (hotbar ?o2 ^type ?type ^location ?loc
                        ^size ?size ^belongs-to ?o1))
    :tests ((> ?size 0)))

((front-of ?o1 ?self)
 :elements   ((self ?self ^y ?y))
 :conditions ((entity ?o1 ^y ?y1))
 :tests      ((> ?y1 (+ padding* ?y))))

((on-horizontal-axis ?o1 ?self)
 :elements   ((self ?self))
 :conditions ((entity ?o1)
              (not (front-of ?o1 ?self))
              (not (behind-of ?o1 ?self))
              (not (carrying ?o1 ?self))))
```

ICARUS' skills describe procedures to achieve certain concept instances in the environment. They are hierarchical versions of STRIPS operators (Fikes & Nilsson, 1971) with a named head, perceptual matching conditions, preconditions that need to be true to execute, direct actions to perform in the world or any sub-skills, and the intended effects of the execution. A *primitive skill* describes the effect of specified actions given a set of satisfied conditions. A *non-primitive skill* describes a more complex procedure by specifying the ordered execution of its simpler sub-skills.

Table 3.2 shows some sample ICARUS skills for Minecraft. The first one, *move-forward-to*, is a primitive skill that describes a procedure to move the agent toward an entity, which is executable only when something is in front of the agent. This skill uses a direct action, **move-forward*,

that continuously move the agent forward at a specified speed. The next skill, *walk-to*, is a non-primitive skill that describes what steps the agent needs to take to be near an entity. Notice that executing the skill involves an ordered execution of multiple primitive skills including the first example. The last skill, *gather-resource*, is another non-primitive skill that describes how the system can gather a resource. In order to execute this skill, a resource should be present in the world and all that needs to be done is for the agent to walk to the resource using the second skill shown in the table. Upon completion, the agent will be carrying the resource as an effect.

Table 3.2: Sample ICARUS skills for the Minecraft domain.

```
((move-forward-to ?o1)
   :elements   ((self ?self))
   :conditions ((front-of ?o1 ?self))
   :actions    ((*move-forward))
   :effects    ((not (front-of ?o1 ?self))))

((walk-to ?o1)
   :elements  ((self ?self))
   :subskills ((move-forward-to ?o1)
               (move-back-to ?o1)
               (move-left-to ?o1)
               (move-right-to ?o1))
   :effects   ((next-to ?o1 ?self)
               (not (moving ?self))
               (not (turning ?self))))

((gather-resource ?o1)
   :elements   ((self ?self))
   :conditions ((resource ?o1))
   :subskills  ((walk-to ?o1))
   :effects    ((carrying ?self ?o1)))
```

ICARUS distinguishes between long-term and short-term memories. Long-term memories store stable contents, whereas short-term memory contents change frequently over time. The long-term memory in ICARUS consists of conceptual long-term memory and skill long-term memory which store concepts and skills respectively. The ICARUS short-term memory is compartmentalized into a belief memory, and an intention memory. ICARUS stores concept instances, or *beliefs*, in its belief memory. These take the form of a concept definition with associated arguments that ground the concept to the current state. Similarly, ICARUS stores skill instances, or *intentions*, inside its intention memory. Intentions consist of a skill definition and a set of bindings to ground the skill.

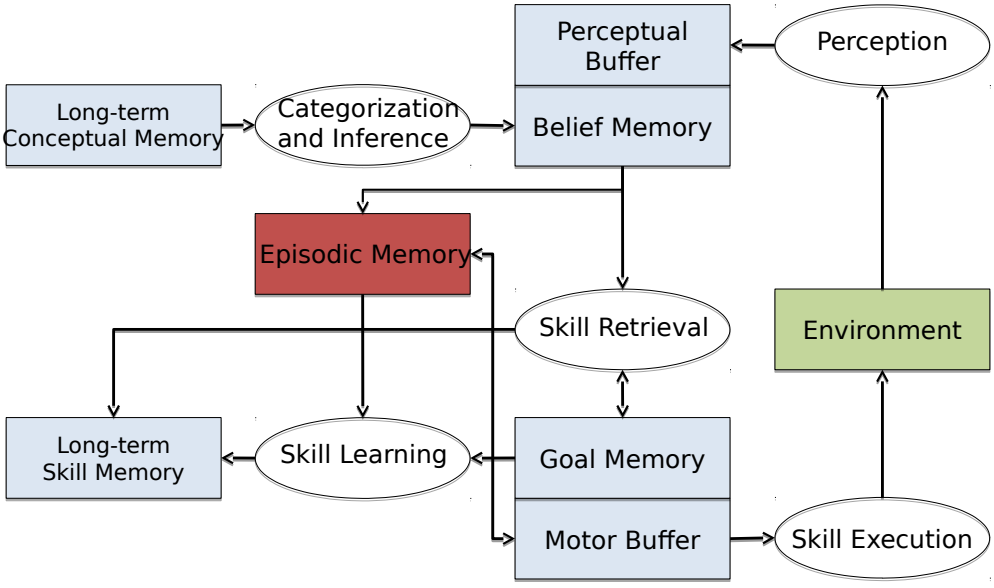

Figure 3.1: ICARUS cycle diagram with the episodic memory module colored in red.

3.2 Execution and Problem Solving

During run time, ICARUS operates in cycles as shown in Figure 3.1. At the beginning of each cycle, the architecture receives perceptual information from the world as a list of uniquely identified objects with their attribute-value pairs. After depositing them into its perceptual buffer, the system computes a belief state by invoking the conceptual inference process, during which the system finds the instances of its concepts that are true based on the current perceptual information. The process occurs in a bottom-up fashion to ensure that all possible inferences are made. ICARUS stores the inferred beliefs in its belief memory, like shown in Table 3.3.

Once the system uncovers the state of the world by inferring all possible beliefs, it proceeds to its goal reasoning step, where the agent can nominate, retract, and prioritize its top-level goals according to their relevance conditions. Only the goals that are relevant tot he current situation will be nominated and prioritized, and any other existing goals will be retracted. After ICARUS selects a set of goals, it attempts to find a skill that achieves one or more of its top-level goals. This process is teleoreactive, in that ICARUS commits to the goals it has but, at the same time, stays reactive to the environment. When the architecture finds a skill that will achieve one or more of its goals, ICARUS instantiates this skill as its intention and stores it in its intention memory. The

system will then execute the intention either by invoking its direct actions if it is a primitive, or by following one of the sub-skills and executing it recursively.

Table 3.3: Sample ICARUS beliefs from the Minecraft domain.

```
(ACTOR SELF1 ACTION ((*TURN-RIGHT)))
(ALIVE ZOMBIE1)
(ENEMY ZOMBIE1 ^X -0.91 ^Y 1.92 ^Z -0.94 ^ORIENTATION -160.2 ^LIFE 10.78)
(ENTITY ZOMBIE1 ^X -0.91 ^Y 1.92 ^Z -0.94 ^ORIENTATION -160.2)
(FRONT-OF ZOMBIE1 SELF1)
(IN-ATTACK-RANGE ZOMBIE1 SELF1)
(NEAR-BY ZOMBIE1 SELF1)
(RIGHT-OF ZOMBIE1 SELF1)
```

If, however, the architecture is unable to find a skill instance that satisfies a goal, it invokes the problem solver to find a solution to the problem. The default method for finding such a solution is a version of backward-chaining means-ends analysis, which decomposes the top-level goals into subgoals or chain skills to achieve unsatisfied preconditions. When the system finds a solution through this process, it can learn new skills upon a successful execution of the solution steps. Although these processes are important aspects of the ICARUS architecture, the current work focuses on its episodic memory and how we can use it for explainable autonomy. For this reason, we refer the interested reader to previous work (Langley & Choi, 2006; Choi & Langley, 2018) for more detailed discussions on these processes.

Chapter 4

Episodic Memory in ICARUS

Episodic memory plays an essential role in storing, organizing, and remembering the events of one's life. Using this memory, one can be mentally transported back in time to re-experience the past. In this chapter, we explore how this capacity can be modeled in ICARUS, helping us to understand the nature of episodic memory and its functional role in cognition. The psychologically inspired theory we present here assumes that:

- Episodic memory is a long-term memory that stores episodes;
- Episodes encode a set of co-occurring events as first-order propositions;
- Episodes contain descriptions of both the agent's internal state and external environments;
- Episodes are organized in a hierarchy, such that each sub-hierarchy contains a collection of structurally similar episodes;
- Episodes at the higher levels of the hierarchy variablize individual differences of the episodes at the lower levels; and
- Remembering an event involves using a retrieval cue to match against the propositional patterns stored in the episodes.

We believe episodic memory is an archive that stores an agent's personal experiences (Martin & Deutscher, 1966). The representation in this memory for such experiences are called episodes, which preserve the contents of the agent's experiences. As Tulving (1983) suggests, episodic memory is a long-term memory that stores records of the agent's experiences. Furthermore, we hold that the contents of an episode are causally linked to the event that the episode captures.

In our theory, episodes are mental images of personal events. We believe that events cannot be captured in a single instant, but rather, they happen over time, starting at one instant and ending at a later time. Episodes in our theory are an ordered pair of propositional states that, when take together, describe an event. For example, an episode can represent an event where "David kicked the ball into the net" with two states. The first describes David's foot position near the ball and its orientation toward the ball, whereas the next state describes the ball in the net. States can also include an agent's action as an attribute of that agent.

In addition to descriptions of the external environment, our theory claims that episodes can also contain descriptions of the agent's internal state, such as the agent's goals and intentions. In this way, ICARUS agents can remember not only what happened in the world, but also remember their own decision making and actions in those contexts.

Furthermore, we believe that episodes are grouped in a hierarchy. Those episodes describing similar events are grouped under the same sub-hierarchy, while episodes that are distinct belong to different sub-hierarchies. In our view, an episode is similar to another episode if its state sequences are relationally equivalent under first-order unification. Additionally, the hierarchy is ordered in such a way that each child episode is a more specific version of its parent.

To explain this further, Schiller et al. (2015) argue that episodic memory receives many of its characteristics from the hippocampus. One function of this brain region is to create a hierarchical network of experience. As the cognitive system places episodes in memory, the hippocampus is believed to dynamically change the structure of this network in order to preserve the similarity relationship between related experiences. In our theory, we model this such that each episode in the hierarchy variablizes the individual differences amongst its children, so as to preserve the structural equivalence relation.

Finally, psychological evidence suggests that episodic memory is an index-based memory (Hellerstedt, 2015; Tulving, 1983), in which episodes are organized in a way that they can be retrieved using semantic patterns or retrieval cues. A proper episodic system must be able to match against elements in episodic memory even if the retrieval cue is not fully specified, attempting to

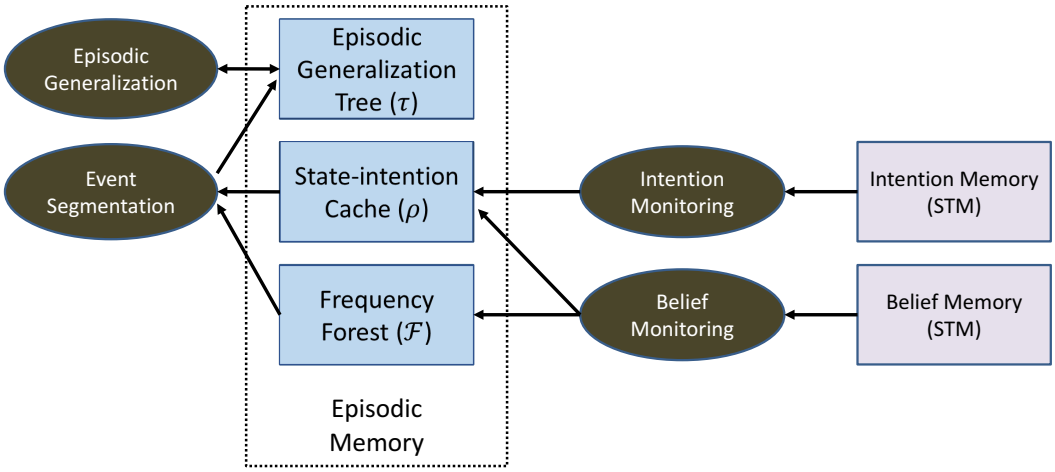

Figure 4.1: Block diagram depicting ICARUS' episodic memory components and information flow starting from the belief memory.

find the most similar episode in the memory. In cases where multiple episodes are equally similar to the cue, the memory system should employ some conflict resolution strategies.

These theoretical postulates in our theory have all been incorporated into our implementation of episodic memory in ICARUS. In the rest of this chapter, we describe the details of the implemented system starting with the organization of the memory components and continuing to the processes that work over them.

4.1 Episodic Representation and Structures

According to Tulving (1983), an episode is a mental construct composed of a sequence of observed changes, with subjectively defined start and end points. Our representation of episodes includes start and end states, as well as the system's choice for the index. There also is a field for specifying how many times the episode occurred and the temporal ordering of these episodes in memory.

More formally, an episode is a tuple that includes a head which contains pointers that temporally locate the episode in the state-intention cache; the start state of the episode; the end state of the episode; the set of significant beliefs in the end state; and a count for the number of times the episode has occurred. Table 4.1 shows a notional episode from Minecraft. In this example,

Table 4.1: Notional episode from Minecraft.

episode(ptr_1,ptr_2,...,ptr_n)
start: *((resource planks1))*
end: *((carrying me planks_item1))*
significant-beliefs: *((carrying me planks_item1))*
count: *n*

the agent initially observes a resource called planks1. Then at a later time, the observes that it is carrying a plank item.

The architecture organizes its episodic memory as a compound structure composed of an episodic beliefs-intention cache, a concept frequency forest, and the episodic generalization tree. Research suggests that there exists an episodic buffer that interfaces between the episodic memory and the working memory. This buffer is responsible for incorporating diverse sets of sensory input and creating a unified representation (Baddeley, 2000). The information deposited into the buffer are later integrated into long-term storage as an episode in the episodic memory. To model this, the extended ICARUS has a *state-intention cache*, (ρ) which stores the state-intention sequence in the order they occurred. This structure is believed to have a limited capacity but, in our current work, we assume that ICARUS can store the complete record of its experience in this cache.

ICARUS also has a *concept frequency forest* $(\mathcal{F})$ that enables the agent to maintain the statistics of what happens in the world. The system uses it to detect significant events that occur. There can be many different ways to do this, but our current implementation includes trees in this forest structure that use the agent's location as the root nodes. Each tree maintains the relative frequency of the beliefs, conditioned on the agent's location.

Finally, ICARUS' episodic memory includes an *episodic generalization tree* $(\mathcal{T})$. This structure organizes the agent's episodes into a hierarchy. The leaf nodes of this tree are fully instantiated episodes that the agent has experienced, and the non-leaf level nodes in the tree stores partially generalized episodes. An episode is partially generalized when it unifies under first-order predicate unification with another episode, i.e, the two episodes are structurally similar.

4.2 Episodic Processes

Figure 4.1 depicts how the components of the episodic memory work together. When ICARUS' inferred beliefs are posited into the state-intention cache, the system checks them for significant beliefs that violate its expectations formed by the frequency forest. Once the architecture identifies such beliefs, it creates an episode to capture the event and stores it in its episodic generalization tree. Then, the system attempts to form generalizations based on all the episodes in the tree, including the one just inserted.

To bring clarity to these episodic memory processes, we discuss each of them in detail below. We begin with how the system caches the state-intention sequence and constructs the concept frequency forest. Then, we explain how episodes are formed and inserted into memory, after which we describe how the episodes can be generalized to induce the episodic hierarchy. Finally, we discuss how episodes are retrieved when ICARUS needs them.

4.2.1 Storing State-Intention Sequence

As stated earlier, ICARUS operates in cycles. On each cycle, the system infers a new belief state. This state can represent beliefs about objects as well as other agents. Overt actions carried out by other agents are represented in the beliefs for those agents. After the system infers a belief state, it is put into the state-intention cache. The system also records the ICARUS agent's intention for each state. So, when the system acts, it perceives the overt actions of other agents, as well as has an understanding of it's own intentions. In our implementation, once the state is collected into the cache, the system processes it to find significant beliefs. These are beliefs that the agent rarely finds true. In order to determine rarity, ICARUS tracks the relative frequency of all beliefs conditioned on location and uses this information to form its expectations. The system applies two expectation thresholds to the frequency information stored in the concept frequency forest.

The agent expects to see any belief with a conditional probability, given the location, greater than the positive threshold, and it expects to not see any belief with a probability less than the

negative threshold. A belief that violates either of these expectations is a significant belief, which prompts the system to create an episode. This is consistent with Kurby and Zacks (2008) where the authors argue that humans perceive the beginning of a new episode when his or her expectations are violated.

4.2.2 Inserting Episodes

ICARUS continuously monitors incoming belief states for significant, unexpected beliefs using its concept frequency forest. When the system finds itself in an unexpected state, it interprets this as a clue that a new event is taking place. Then, the system creates an episode at the event boundary between the previous and new states. Table 4.2 shows this process in pseudocode. The first five lines declare the necessary variables needed to perform insertion. Among these are the state-intention cache, the agent's current location, the current and the previous states, and the agent's intention. Next, the system adds the current state and intention to the cache (Line 6) and checks for significant beliefs in the state on Line 7. When the system detects one or more significant beliefs, it creates an episode using the previous and the current states and inserts it into the episodic memory.

Table 4.2: Pseudocode for creating a new episode.

```
1:  ρ ← state-intention cache
2:  loc ← current location
3:  B_c ← current belief state
4:  ι ← agent's intention
5:  B_prev ← last state in ρ
6:  ρ ← ρ.add(B_c, ι)
7:  sigs ← GETINTERESTINGBELIEFS(B_c, loc)
8:  if not NULL(sigs) then
9:      ε ← MAKEEPISODE(sigs, B_c, B_prev)
10:     T ← INSERT(ε, T)
```

To insert the new episode at the proper location the episodic memory, the system searches through the episodic generalization tree in a level-order fashion as shown in Table 4.3. Starting at the root of the tree, and ICARUS checks each node, level by level to find a matching episode

whose significant beliefs unify under first-order predicate unification with those of the new episode. During this search, if the system finds an exact match that requires no substitutions, the counter for the existing episode is incremented by one, and the new episode is not inserted again (Lines 7 - 9). In case of a generalized match, where unification is only possible with substitutions, the system increases the count for the matching episode, and continues searching for further matches among the episodes that are children of the previous match (Lines 10 - 14). If at any level, the system does not find any matches between the existing episodes in the hierarchy and the new episode, the system places the latter as a child of the current parent episode (Lines 17 - 19).

Table 4.3: Pseudocode for inserting a new episode.

```
1:  queue ← ∅
2:  temp ← root of 𝒯
3:  match ← ∅
4:  p ← ∅
5:  while not NULL(temp) do
6:      match ← STRUCTURALEQ?(temp, ε)
7:      if match is exact match then
8:          temp.count ← temp.count + 1
9:          BREAK
10:     else if match is bc of unification then
11:         temp.count ← temp.count + 1
12:         queue ← ∅
13:         queue ← temp's children
14:         p ← temp
15:     temp ← queue.FIRST
16:     queue ← queue.POP
17: if null(temp) and match not exact then
18:     p ← p.ADDCHILD(ε)
19:     𝒯 ← GENERALIZE(p, ε)
```

4.2.3 Generalizing Episodes

ICARUS supports episodic generalization during insertion into its episodic tree during insertion. Through this process, the architecture maintains a taxonomic hierarchy of episodes induced by the

structural similarity of episodes. The system is able to detect new similarity relationships among sibling episodes at each level, potentially enhancing the taxonomic hierarchy of episodes at each insertion.

Table 4.4 shows a pseudocode for this process. Two sibling episodes ε_i and ε_j of the current parent ε_p, generalize if and only if there exists episode ε_g, such that ε_g is the parent of both ε_i and ε_g, but neither of them can be the parent of ε_g. If this is the case, the `variablize` function on Line 5 returns the sets of variable bindings from ε_g to ε_i and from ε_g to ε_j. This means that ε_g is a first-order predicate unifier for ε_i and ε_j, which is tested again for validity on Line 6. If such a generalized episode ε_g that is more specific than the original parent ε_p, then ε_g becomes a child of ε_p, and ε_i and ε_j become the children of ε_g. The count for the generalized episode will be the summation of the counts for its children.

4.2.4 Retrieving Episodes

The episodic memory in ICARUS supports cue-based retrieval to allow the agent to recall its personal past. Retrieval cues are propositional patterns, in the form of partially instantiated concept heads that can include some unassigned variables, the values of which are not relevant to the meaning of the cue.

The retrieval process is recursively carried out in a level-order fashion through the episodic generalization tree.. At each node in the tree, the system performs relational matching against the elements in the episode and the retrieval cue. The matching process can return an exact match, a generalized match, or no match at all. An exact match means that the cue is contained in the episode, exactly as specified. A generalized match means that there exists an episode that matches the cue in an instantiated form. If the system finds a generalized match, it adds the match to a list of matches and continues the search on that level. For each match found at that level, the system recurses on its children.

In summary, ICARUS records the belief state and executed intentions into the episodic cache and updates the frequency forest every cycle. When the agent identifies one or more significant

Table 4.4: Pseudocode for generalizing episodes.

```
1:  if NOT(NULL(ε_p)) then
2:      children ← GETSUBEPISODES(ε_p)
3:      ε_g ← ∅
4:      for ε_j ∈ children do
5:          ε_g ← VARIABLIZE(ε_j, ε_i)
6:          if VALIDGENERALIZATION(ε_g, ε_p, ε_j, ε_i) then
7:              ε_g.SETSUBEPISODE(ε_i)
8:              ε_g.SETSUBEPISODE(ε_j)
9:              ε_p.SETSUBEPISODE(ε_g)
10:             break
```

beliefs in the belief state, it constructs a new episode and inserts it into the generalization tree. After insertion, the system checks to see if the new episode can generalize with any of its siblings. As a result, the root node of the generalization tree is the most general episode, having an arbitrary number of children, and the episodes become more specific at increasingly lower level levels. The leaf nodes of the tree are fully instantiated episodes.

Chapter 5

ICARUS Agent with Explainable Autonomy

We argue that episodic memory lends itself to answering questions about the personal past, enabling explainable autonomy in artificial agents. In this chapter, we describe the episodic phenomena that we believe are essential to explainability and present an ICARUS agent that takes advantage of the new episodic extension to exhibit question answering capabilities. We also present a demonstration of such capacities in Minecraft.

5.1 Episodic Memory for Explainable Autonomy

Episodic memory enables several unique psychological phenomena that are suitable for explainable autonomy. Evidence shows that humans with episodic memory can summarize personal past experiences at appropriate levels of detail, answer questions about their intentions, goals, and beliefs, and use personal history to predict their future behavior in hypothetical situations (Atance, 2015; Kurby & Zacks, 2008). Individuals with deficiencies in their episodic memory generally cannot demonstrate such behaviors. Given this, a computational model of episodic memory, with its associated mental structures and processes, should also allow an artificial agent to remember its personal past and use those memories to expose its internal decision-making processes to its user.

The behaviors we think are most suited for explainable autonomy include summarization, question answering, and hypothetical thinking. We explain each of these episodic phenomena below and also provide insight on how to achieve explainable agents using episodic memory.

***Summarization*:** People can summarize their past experiences. They are able to provide information at appropriate levels of abstraction, rather than spewing low-level information about quotidian

happenings that may overwhelm their counterparts. People can also provide more detailed summaries whenever someone asks.

***Question answering*:** People are able to answer a variety of questions about their internal decision-making processes and overt behaviors. Among other things, people can explain how and why goals were achieved, how and why actions and behaviors were executed, and which alternative methods they have considered. Question answering is not a one-off behavior. People can provide more detailed information when asked, and they can also answer follow-up questions about their goals, beliefs, and intentions at varying levels of detail.

***Hypothetical thinking*:** People can think about and discuss with others what they might do in the future. Psychological evidence from Atance (2015) shows that young children's ability to think about their future depends on what they remember about their past. One well studied individual with episodic memory impairments, patient K.C., also showed that he could not imagine himself in the future (Tulving, 2002).

Our goal is to create agents that demonstrate this range of behaviors using episodic memory and facilitate explainability with the episodic contents. An agent records its personal experiences as episodes, which are stored and organized inside its episodic memory. Processing episodic memory contents allows an agent to expose details about its past experience. This emphasizes the role of knowledge and information processing in explainable autonomy. In contrast to popular machine learning techniques, the cognitive systems approach recognizes that explainable agency is not purely a matter of inputs and outputs, but multi-step mental processes that involve many different modules in the cognitive architecture that are domain independent.

5.2 ICARUS Agent with Episodic Capabilities

Using the extended ICARUS, we programmed an agent that plays Minecraft scenarios and answers questions about its experience achieving goals. We gave it the task of defending itself against zombies, which requires positioning and maneuvering the agent to face the zombie while maintaining

Table 5.1: Taxonomy and partonomy in ICARUS concepts for Minecraft.

```
((resource ?o1 ^x ?x ^y ?y ^z ?z ^orientation ?orientation)
   :elements ((planks ?o1 ^x ?x ^y ?y ^z ?z ^orientation ?orientation)))

((enemy ?o1 ^x ?x ^y ?y ^z ?z ^orientation ?orientation ^life ?life)
   :elements ((zombie ?o1 ^x ?x ^y ?y ^z ?z ^orientation ?orientation ^life ?life)))

((entity ?o1 ^x ?x ^y ?y ^z ?z ^orientation ?orientation)
   :conditions ((enemy ?o1 ^x ?x ^y ?y ^z ?z ^orientation ?orientation)))

((entity ?o1 ^x ?x ^y ?y ^z ?z ^orientation ?orientation)
   :conditions ((resource ?o1 ^x ?x ^y ?y ^z ?z ^orientation ?orientation)))

((carrying-sword-materials)
   :conditions ((carrying ?self ?planks ^type ?type1 ^size ?size1)
                (carrying ?self ?sticks ^type ?type2 ^size ?size2))
   :tests ((>= ?size1 2)
           (>= ?size2 1)
           (eq 'stick ?type1)
           (eq 'planks ?type2)))
```

an appropriate distance to strike the it, and sometimes also involves gathering materials and assembling a weapon. The rich interactions in the game makes the task of defending itself against a zombie cognitively complex. At any given moment, multiple options exist for choosing a good defensive position and attacking the zombies. For this reason, explaining the agent's behavior is not a trivial task.

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 53 concepts, including 15 primitive concepts and 37 skills, including 23 primitive skills. In addition to the concepts shown in Table 3.1 earlier, Table 5.1 shows some sample concept definitions that include information on how objects relate to each other taxonomically and partonomically. For example, the first four definitions say that resources and enemies are all examples of entities, and the last concept states that carrying planks and sticks are all part of carrying materials for making a sword. The sample skills shown previously in Table 3.2 allow ICARUS to accomplish different tasks in Minecraft. These skills described some of the overt behaviors the agent can demonstrate.

In addition to the suite of skills for accomplishing tasks, ICARUS also has some skill for question answering. For instance, consider the skill for answering questions about how a goal was

accomplished shown in Table 5.2. This skill allows the agent to answer questions about how it achieved a goal. Notice that ICARUS receives the user's question as a perception, where ?$o1$ is the subject of the question. This can either be the name of a belief, or the name of a skill. The type denotes whether the question pertains to a goal or an intention. The questions also include how or why the agent behaves the way it did, as well as alternative ways of achieving a goal or executing an intention. These are specified as Boolean attributes to the question percept. If the agent believes that the user is not informed about the subject of the question, then the system attempts to answer the question to change that.

Table 5.2: An ICARUS skill for question answering.

```
((inform-how ?o1)
   :elements   ((question ?o1 type goal how t))
   :conditions ((uninformed ?o1))
   :actions    ((*explain-goal-achievement ?o1))
   :effects    ((not (uninformed ?o1))))
```

The work for carrying out the explanation is encapsulated in the action **explain-goal-achievement*, outlined in Table 5.3. It starts with the agent retrieving a history of previously executed intentions from its episodic memory (Line 1). These goals that these intentions achieved are then examined for relevance to the question (Line 2). Once a match is found, the system synthesizes text that gets output to the screen (Line 3).

Table 5.3: Pseudocode for explaining goal achievements.

```
1: for intention in REMEMBER(ρ, τ, goal) do
2:     if goal ∈ intention.effects then
3:         SAY("I did ?x to achieve ?y", intention.head, goal)
```

Table 5.4 provides more details on how ICARUS remembers all relevant experiences. The system first returns a list of episodes that are relevant to the *goal*. Then, for each returned episode, the system recovers the intention information by following the cache pointers of the episode. The system returns all such intentions that are not empty.

Table 5.4: Pseudocode for remembering the agent's past through episodic retrievals.

```
1: remembered ← RETRIEVE(τ, goal)
2: idxs ← ∅
3: ints ← ∅
4: for all episode in remembered do
5:     collect episode.cache-pointers into idxs
6: for all idx in idxs do
7:     executed-intention ← ρ.intentions[idx]
8:     if not NULL(executed-intention) then
9:         collect executed-intention into ints
10: return ints
```

5.3 Demonstrations

To evaluate the new question answering capacity based on the episodic memory, we designed two scenarios. The first scenario placed an ICARUS agent in a room with a zombie and told the agent to defend itself like shown in Figure 5.1. After the agent successfully neutralized the threat, we asked the agent a series of questions about what happened. In the second scenario, the agent is tasked to make a sword. The agent must collect the necessary resources to to craft the word, build its components, then assemble the sword. After completing this task, we asked the agent about several questions to explain how it built the sword. We present is a transcript of the interactions after these two scenarios, where we write English translations of the questions in bold, the perceptual representation of the question underneath, and the agent's response in bullet points.

5.3.1 Scenario One: Simple Zombie

Goal Achievement: How did you achieve the goal where the zombie1 is not present?

(question not_enemy_zombie1 type goal how t)

- `I did (ATTACK ZOMBIE1) to achieve (NOT (ENEMY ZOMBIE1)).`
- `I did (KILL ZOMBIE1) to achieve (NOT (ENEMY ZOMBIE1)).`

In this response, the system exposes that it performed two different procedures for clearing the

Figure 5.1: ICARUS fighting a zombie in Minecraft.

room of the zombie. We discover why these two different procedures were followed in the next two questions.

Skill Selection: Why did you choose to attack zombie1?

(question attack_zombie1 type intention why t)

- `I executed (ATTACK ZOMBIE1) because I wanted to achieve ((NOT (ENEMY ZOMBIE1))).`

  `All I had to do make it executable was to have:`

  `((ARMED-AND-DANGEROUS ME) (NEXT-TO ZOMBIE1 ME))`

Skill Selection: Why did you choose to kill zombie1?

(question kill_zombie1 type intention why t)

- `I executed (KILL ZOMBIE1) because I wanted to achieve: ((NOT (ENEMY ZOMBIE1))).`

```
All I had to do make it executable was to have:
((ARMED-AND-DANGEROUS ME))
I could have done (ATTACK ZOMBIE1), but at the time
((NEXT-TO ZOMBIE1 ME) (ARMED-AND-DANGEROUS ME))
were unsatisfied conditions.
```

Notice that the agent makes known alternatives that it could have chosen and explains why they were not selected. This example shows that when presented with two or more skills to achieve the same goal, ICARUS will select the skill with the the fewest number of unsatisfied preconditions. This heuristic shortens the amount of time the agent spends problem solving, resulting in faster reaction times. Next, we find out about the structure of (KILL ?O1) and (ATTACK ?O1).

Skill Execution: What are the subskills or actions to execute attack zombie1?
(question attack_zombie1 type intention how t)

- `I executed (ATTACK ZOMBIE1) by performing motor functions:`

  `((*ATTACK 'ZOMBIE1))`

The attack skill is an appropriate in the situation, and the agent tells the user that it executed the motor functions to complete the task.

Skill Execution: What are the subskills or actions to kill zombie1?
(question kill_zombie1 type intention how t)

- `I executed (KILL ZOMBIE1) by following steps:`

  `((OFFENSIVE-GO-TO ZOMBIE1) (ATTACK ZOMBIE1))`

The agent reveals that (KILL ?O1) is a more abstract description of how to remove a zombie than (ATTACK ?O1), which is contained as the second subskill in the former. In this Minecraft scenario, a zombie arbitrarily moves around until it sees the ICARUS agent at a distance. Then, it tries to

get closer and attack the agent. In the ICARUS agent's perspective, when the zombie is far from it, the most appropriate skill to select is (KILL ?O1), which includes a plan for moving toward the zombie followed by one for attacking it. When the zombie is next to the ICARUS agent, the appropriate skill is simply to (ATTACK ?O1), because the zombie is already in proximity, and the system can execute primitive attack actions to achieve its ends.

Subgoal Decomposition: Why did you nominate the goal of being armed and dangerous?
(question armed-and-dangerous_me type goal why t)

- `I nominated (ARMED-AND-DANGEROUS ME) because it was a necessary condition for (KILL ZOMBIE1).`

In this response, the agent explains its reasons for trying to achieve situations in the world. In the current implementation, goals are either given to the agent as a top-level goal, or the system can select subgoals to achieve which it thinks will allow it to achieve its top-level goal.

5.3.2 Scenario Two: Sword Crafter

Goal Achievement: How did you achieve the goal where you are holding an wooden sword in your hand?
(question holding_me_wooden-sword-item1 type goal how t)

- `I did (EQUIP WOODEN_SWORD_ITEM1) to achieve (HOLDING ME WOODEN_SWORD_ITEM1).`

The agent responds that it was holding the wooden sword because it decided to equip it. The agent's explanation reveals its understanding of the causal relationship between its behavior, and the goals they can achieve.

Goal Achievement: How did you achieve the goal where you are carrying an wooden sword?
(question carrying-wooden-sword_me_wooden-sword-item1 type goal how t)

- `I did (CRAFT-WOODEN-SWORD STICK_ITEM1 PLANKS_ITEM1) to achieve`

(CARRYING-WOODEN-SWORD ME WOODEN_SWORD_ITEM1).

Skill Execution: What are the subskills or actions to make an wooden sword?

(question craft-wooden-sword_stick-item1_planks-item1 type intention how t)

- I executed (CRAFT-WOODEN-SWORD STICK_ITEM1 PLANKS_ITEM1) by performing motor functions: ((*MAKE wooden_sword))

In this response, the system reveals that making a wooden sword is a primitive skill. In Minecraft, an agent can craft items using simple craft commands so long as it is carrying all of the item's constituent parts.

Skill Selection: Why did you choose to make an wooden sword?

(question craft-wooden-sword_stick-item1_planks_item1 type intention why t)

- I executed (CRAFT-WOODEN-SWORD STICK_ITEM1 PLANKS_ITEM1) because I wanted to achieve: ((CARRYING-WOODEN-SWORD SELF1 ?2)). All I had to to do make to it executable was to have: ((CARRYING-SWORD-MATERIALS SELF1 STICK_ITEM1 PLANKS_ITEM1))

Before the system crafts a wooden sword it needs to possess all of the sword materials, namely a stick and a some planks. Even though crafting a wooden sword is a primitive skill, the agent still needs to make and successfully execute a plan to satisfy its precondition.

Goal Achievement: How did you achieve the goal where you are carrying materials for a sword?

(question carrying-sword-materials_me_stick-item1_planks-item1 type goal how t)

- I did (GATHER-SWORD-MATERIALS PLANKS2 ?PLANK2 ?PLANK3 ?PLANK4) to achieve (CARRYING-SWORD-MATERIALS ME STICK-ITEM1 PLANKS-ITEM1).

In this response, the system reveals that it plans on acquiring the sword materials by going to collect four planks. At the moment when the agent was planning, it had only committed itself to gather planks2. Later on, it would decide which other planks to pick up.

Skill Execution: What are the subskills or actions to gather materials for a sword?

(question gather-sword-materials_planks2_?planks2_?planks3_?planks4 type intention how t)

- `I executed (GATHER-SWORD-MATERIALS PLANKS2 ?PLANK2 ?PLANK3 ?PLANK4) by following steps:`

  `((GATHER-STICK-MATERIALS PLANKS2 ?PLANK2)`

  `(GATHER-SWORD-BLADE-MATERIALS ?PLANK3 ?PLANK4)`

  `(CRAFT-STICK ?24))`

Here, the system reveals that gathering the sword materials requires gathering the stick materials, and the sword blade materials, then crafting a stick. Again, ICARUS was not sure which planks to gather and also does not know which stick it will make. For now, it suffices that ICARUS knows that it will make *a* stick.

Skill Execution: What are the subskills or actions to gather materials for a stick?

(question gather-stick-materials_planks2_?plank2 type intention how t)

- `I executed (GATHER-STICK-MATERIALS PLANKS2 PLANKS1) by following steps:`

  `((GATHER-RESOURCE PLANKS2) (GATHER-RESOURCE PLANKS1))`

The system chose to ground the plank variables to actual plank objects in the world when it started executing (GATHER-STICK-MATERIALS ?O1 ?O2).

Skill Execution: What are the subskills or actions to gather a resource?

(question gather-resource_?plank type intention how t)

- `I executed (GATHER-RESOURCE PLANKS1) by following steps: ((WALK-TO PLANKS1))`

  `I executed (GATHER-RESOURCE PLANKS2) by following steps: ((WALK-TO PLANKS2))`

The agent reveals that it was able to gather PLANKS1 and PLANKS2 by walking to them. Since Minecraft is a continuous domain, (WALK-TO ?O1) involves a combination of moving forward, backwards, turning left, and turning right, but for brevity, we leave out these details from the explanations.

The debriefing could go on further in a similar fashion. The ICARUS agent is able to describe in detail information about its decision-making process that allows a user to predict its behavior and make expectations about what tasks the system can accomplish. Note that a system without episodic memory capabilities will not be able to answer questions about its behavior, not to mention provide appropriate information in this level of detail. This demonstrates that researching the role of episodic memory and its related processes is a promising direction for creating explainable collaborative agents.

Chapter 6

Related Work

The novel contributions of the current work are inspired by many previous work in several different directions. We begin by reviewing work on memory, and then discuss explainable autonomy.

6.1 Memory

Some previous work in cognitive systems discussed episodic memory in computational agents. Most notably, Soar (Laird, 2012b) has an episodic memory that contains sequentially ordered snapshots of the agent's working memory (Nuxoll & Laird, 2007). The authors present a series of design decisions that are generically supported by psychological evidence. The episodic memory in ICARUS is also psychologically inspired, but the architecture has a commitment to hierarchical organization of episodes in memory that Soar does not have. The episodic memory in Soar stores episodes in a flat list and it searches the entire memory during retrieval using the specified cue, where as our system can find an episode that is representative of its children in the hierarchy.

More recently, a previous version of ICARUS had episodic memory-like capabilities. Stracuzzi et al. (2009) extended the architecture to reason over time by adding timestamps to concept instances. In order to utilize the timestamps, the system posits new beliefs into the belief memory without first clearing previous ones. With this extension, ICARUS was able to reason about the temporal relationship that exists in the world. In comparison, the current extension gives ICARUS a dedicated episodic memory that stores episodes, which allows the system to remember what happened. The previous work represents the passage of time only by changing numeric attributes of beliefs, and this does not qualify as an episodic memory with which the system can create and

store episodes. Consequently, that system has no explicit notion of experience. Our system also has the ability to explain its behavior by recalling its past experiences which the previous work did not cover.

In addition to these cognitive systems, work in incremental concept formation is also relevant to episodic memory because such systems build a hierarchy of experiences from individual instances. One of the most famous examples is COBWEB (Fisher, 1987). Like many of its predecessors, COBWEB induces a concept hierarchy from input data in an online fashion. Training instances are represented as lists of attribute value pairs with associated probabilities and are sorted in a tree from top to bottom. Although COBWEB made improvements on its predecessors, it could only represent states as lists of attribute value pairs, and attributes could only take discrete values. Subsequent systems based on COBWEB tried to address some of its limitations. Gennari et al. (1989) developed a system that handles numeric attributes by modeling the occurrence of an attribute value as a normal random variable. Most recently, TRESTLE (MacLellan et al., 2015) succeeds in performing incremental concept formation with both symbolic and numeric values.

ICARUS' episodic memory differs from these systems in three important ways. First our system does not represent probabilistic states. Instead, ICARUS stores probabilities in the concept frequency forest, and its generalization hierarchy is induced by variablizing individual differences amongst lower-level episodes. Second, episodes in ICARUS' episodic memory have a temporal dimension and are not snapshots of single states. The only system, to our knowledge, that can represent episodes like this is TRESTLE. Structures called *components* can form an episode by specifying its component states. In order for this to work, however, each state in a component must be represented as a list of attribute-value pairs. So, TRESTLE cannot represent propositional states using components only. Third, the episodic memory in ICARUS is part of the system's cognitive architecture. Because of this, its episodic memory can store and remember the agent's plans. The other memory systems cannot do this since they are not part of a larger cognitive system.

In addition to incremental models of memory, Hochreiter and Schmidhuber (1997) devised a Long Short-Term Memory (LSTM) in recurrent neural networks that can be remember state

information for arbitrarily long time intervals. Mahasseni et al. (2017) show that this technique can be used to summarize experiences. It takes a video as input and returns a sequence of frames that represent the video. The ability to summarize video into a series of key frames bares resemblance to the summarization capability in ICARUS' episodic memory, however, there are many distinctions between ICARUS' episodic memory and this work. The first is that ICARUS' episodic memory is a tree structure built incrementally over time. The LSTM's structure, on the other hand, is set before training occurs. Another key difference is that ICARUS can summarize events at multiple levels of details, while the LSTM can only summarizes video at one level of detail. Furthermore, ICARUS explanations are human-readable, while the LSTM only presents key frames for humans to interpret.

6.2 Explainable Autonomy

Another branch of related work involves explanation generation. Systems that explain their internal motivations and justify their actions are not new (Doyle et al., 2003; Sørmo et al., 2005). The case-based reasoning community has a rich history of building such systems. One early work described a system that provides external explanations of how it designed physical devices (Goel & Murdock, 1996). This work used meta-cases that store a trace of the cognitive processing done during problem solving. These meta-cases could be retrieved to explain to a user the reasoning behind the agent's design choices. There are several differences between these systems and the extended ICARUS. First, our work describes a domain independent episodic memory that is not limited to handle only the agent's design choices for physical devices. Second, our work integrates the episodic memory inside a cognitive architecture. Third, the plans the system generated, which can be viewed as internal explanations, are repurposed into external, text-based explanations that people can consume. Fourth, explanatory ability is encapsulated inside skills, which are more similar to goal-task network formalisms (Alford et al., 2016).

Recently, there have been efforts to explain black box algorithms. Turner (2016) created a system that generates Boolean statements to explain black box classifiers. Given an input vector,

the system would output a statement defined on the input feature space that explained the class assignment of the input. The system takes a Monte Carlo approach to understand the characteristics of the underlying classifier. Unlike decision trees, it only explains the model working in individual input cases, but not how the system works in general. The work is limited to classifiers and does not provide any information on what features the input should have to obtain different classification result. In contrast, we focus on building a system that that explains extended behavior, not just the properties of the input that led to certain outputs.

Furthermore, there has been a push towards building explainable machine learning systems (Aha et al., 2017). Although there is a heavy focus in machine learning, there are a few model-based systems that address the issue of explainable artificial intelligence. Fox et al. (2017) discuss why model-based planning systems need to have explainable components. They cite people's growing need to collaborate with intelligent agents, and describe properties of explanations, such as causality, that help human users understand why actions were taken. Although this work is philosophically in line with ours, it lacks theoretical postulates for explainability, and the authors do not provide an implementation of a planning system with the explainable qualities they discuss. Furthermore, it seems that the authors assume that the end user asks questions about the planner's current plan, but not those plans that were executed in the past. In contrast, our theory of explainability relies heavily on episodic memory, and ICARUS agents are able to explain behaviors that occurred in their personal past.

In robotics, researchers developed a robotic system that can explain its decisions when designing new tools (Wicaksono & Sheh, 2017). The system uses relational models to describe the tools and can learn how to use tools via inductive logic programming. The robot is capable of explaining why or why not the system makes certain decisions. States are represented in an abstract, relational manner similar to the concepts in ICARUS, and actions are represented as STRIPS operators. Additionally, the authors gave the system an ontology of tools, which is used as a case base for modifying and creating new tools. The system demonstrates explainable behavior by storing its plans for achieving goals in what is essentially an episodic memory of their own for storing

the agent's plans. It, however, seems to be specialized to tool creation and use, and the system only supports answering questions about its external behavior, not its choice of specific goals. Furthermore, the agent's actions are not discussed in relation to the goals they achieve, so it will be difficult for human users to understand the agent behavior without prior understanding of the application domain.

Johnson's (1994) Debrief system is perhaps most similar to our work. The authors present Debrief, a system that explains its behavior to a user in an after-action interview. The system includes an episodic memory and integrates it within the Soar cognitive architecture. The agent stores the plans it creates, and is able to recall the plans to generate an explanation. The system can answer questions about what it did, as well as provide details about what it could have done. Nonetheless, the work makes few theoretical commitments to episodic memory representations and processes, providing little details about when and how episodes are constructed, or how they are organized in memory. Furthermore, Debrief requires a specification of what state elements are relevant for explanations. ICARUS' episodic memory does not need such specifications and is integrated into the architecture's cognitive cycle, so that it automatically stores its experiences in memory without interruptions.

Chapter 7

Future Work

Now that we have presented our theoretical position on the role of episodic memory in explainable autonomy and shown how the theory can be implemented, we propose a research agenda that can guide our efforts for further progress. First, we would like to enable the system to summarize multiple episodes into one explanation. We showed that the system can abstract away low-level details when describing how it killed a zombie. So, to some extent, our agent can already explain its behavior in an abstract manner, but we want our system to have the ability to synthesize its history into summaries. Much like the way humans tell stories, we want our systems to be able to tell users stories about the events they experience. For example, when a user tasks an agent to complete a task, the user may initially be interested in "what happened". In such scenarios, the system should be able to retrieve a series of episodes from the episodic memory and weave their components together to explain large segments of time. In other words, it should extend the current system by integrating multiple agent intentions and plans into one explanation.

Second, we would like to build multiple, cooperating agents that share their experiences through one episodic memory. Since one agent's experiences would affect all agents sharing that memory, we wonder how learning and performance in multi-agent domains would be affected by shared episodic memories. In one scenario, we envision a team of agents acting to achieve a joint goal. We also would like to observe how the quality of learned knowledge is affected when multiple agents start learning asynchronously.

Third, in this work we showed that our system is able to explain its rationale for why it takes action. In addition to this, we would like our system to adapt to user preferences by adhering to feedback. In general this is a difficult endeavor, since an agent will likely need meta-cognitive

ability to control how it thinks about its own behavior. One plausible next step, however, is to model inverse trust as done by Floyd et al. (2014). In this way, the system can estimate if users believe it is a trustworthy agent.

Fourth, although our system can infer beliefs about the mental state of others, through perceptual questions, we did not talk very deeply about shared mental models (Mathieu et al., 2000; Jonker et al., 2011). In the future, we would like our system to be able to use shared mental models for demonstrating explainable behavior. Constructing a good explanation not only depends on having methods for responding to questions, but also on the human-agent dyad's understanding of the question being asked. This means that an agent should be able to reason about what information would be most relevant to satisfying the user's goals.

Fifth, we would like to partner with cognitive scientists and psychologists to evaluate these agents in human-robot teams. We would like to understand how humans perceive the quality of the explanations of the system, and also we would like to understand how humans try to structure the agent's experiences so it learns desirable behavior. These human subject tests are valuable because even in communities that have historically focused on generating explanations for humans, very few of them have actually verified the quality of their explanations with humans (Doyle et al., 2003; Sørmo et al., 2005).

Sixth, we desire systems that can remember and talk about their experiences at different levels of detail. Those interested in creating episodic memory systems ought to consider the scalability of their approaches. Creating systems that scale usually is not the most important consideration in research, but in the case of episodic memory, scalability is an important issue because only episodic memory systems that scale can attempt to solve problems that require the agent to exist for extended periods of time. To achieve scalable systems, we want to consider innovative episodic memory organizational schemes as well as examine established strategies, such as the use of hierarchy.

Finally, We also consider ethical aspects of explainable autonomy. If current trends continue, AI will fundamentally pervade our personal and societal lives. We argue that existing in a society

with other intelligent agents that humans can interact with and understand is far more desirable than living with artificial systems that are shallow reproductions of human behavior. This issue is even more complicated for collaborative agents built from machine learning technology. These agents will encourage people to attribute to them goals, intentions, and beliefs, when in actuality they possess none of these things. They portray an image of themselves that does not reflect reality. They can come across as open and inviting, but are no less impenetrable than other machine learning-based systems. So, our final thought is to attempt to actively represent the cognitive systems paradigm in explainable agency to avoid this ethical dilemma.

Chapter 8

Conclusions

There has been a widespread acceptance of artificial intelligence techniques in many different applications, but the continued growth of the field will be limited if as humans cannot understand, trust, and properly manage autonomous systems. The state of the art in artificial intelligence has produced systems with incredible performative ability, but often by inducing opaque, non-intuitive decision functions that humans cannot interpret. This makes these systems black boxes, for which humans cannot predict when they will make mistakes or successfully achieve its end. Episodic memory provides key insights into this problem by providing a window into the internal decision-making processes of intelligent agents. In this thesis, we discussed how episodic memory in cognitive systems plays an important role in constructing intelligent agents that can explain their behavior to their human counterparts.

The cognitive systems approach has received less attention in the context of explainable autonomy, but we believe that the paradigm makes it possible to build integrated systems that humans can understand and collaborate with. In this thesis we showed evidence that suggests that reasoning over episodic memory contents allows an agent to explain its behavior using constructs, like goals and intentions that humans readily understand.

Our work also described a series of behavioral aspects, namely the abilities to summarize events, answer questions, and engage in hypothetical reasoning that enable agents to demonstrate explainable behavior using episodic memory. We also implemented this theory in the context of the ICARUS cognitive architecture and built an agent that demonstrates those capabilities in the game of Minecraft. Our results suggest that systems with episodic memory can answer a variety of questions about their decision-making process.

We also provided our thoughts on promising directions that could yield new insights into explainability and sought to begin a conversation about the challenges facing work on explainable agents in both technical and ethical perspectives. In closing, we found that episodic memory enables computational agents to explain their internal and external experiences. As we enter a new age of artificial intelligence, explainability will be essential for fostering a harmonious integration of intelligent agents in our society. Episodic memory is a fundamental component of human cognitive ability, and our extended architecture serves as an important basis for future research.

References

Aha, D. W., Darrell, T., Pazzani, M., Reid, D., Sammut, C., & Stone, P., Eds. (2017). *IJCAI 2017 Workshop on Explainable Artificial Intelligence*, Melbourne, Australia.

Alford, R., Shivashankar, V., Roberts, M., Frank, J., & Aha, D. W. (2016). Hierarchical planning: Relating task and goal decomposition with task sharing. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence* (pp. 3022–3029).

Anderson, J. R. & Lebiere, C. (1998). *The Atomic Components of Thought*. Mahwah, NJ: Lawrence Erlbaum Associates.

Atance, C. M. (2015). Young children's thinking about the future. *Child Development Perspectives*, 9(3), 178–182.

Baddeley, A. (2000). The episodic buffer: A new component of working memory? *Trends in Cognitive Sciences*, 4(11), 417–423.

Bölöni, L. (2011). An investigation into the utility of episodic memory for cognitive architectures. In *AAAI 2011 Fall Symposium: Advances in Cognitive Systems* (pp. 42–49).

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

Doyle, D., Tsymbal, A., & Cunningham, P. (2003). *A Review of Explanation and Explanation in Case-Based Reasoning*. Technical report, Trinity College Dublin, Department of Computer Science.

Faltersack, Z., Burns, B., Nuxoll, A., & Crenshaw, T. L. (2011). Ziggurat: Steps toward a general episodic memory. In *AAAI 2011 Fall Symposium: Advances in Cognitive Systems* (pp. 106–111).

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

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

Floyd, M. W., Drinkwater, M., & Aha, D. W. (2014). Case-based behavior adaptation using an inverse trust metric. In *AI and Robotics: Papers from the AAAI-14 Works* (pp. 16–21).

Fox, M., Long, D., & Magazzeni, D. (2017). Explainable planning. In *arXiv Preprint arXiv:1709.10256*.

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

Goel, A. K. & Murdock, J. W. (1996). Meta-cases: Explaining case-based reasoning. In *European Workshop on Advances in Case-Based Reasoning* (pp. 150–163).: Springer.

Gunning, D. (2017). *Explainable artificial intelligence*. Technical report, Defense Advanced Research Projects Agency.

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

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.

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

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).

Johnson, W. L. (1994). Agents that learn to explain themselves. In *Proceedings of the Twelfth National Conference on Artificial Intelligence* (pp. 1257–1263).

Jonker, C. M., Van Riemsdijk, M. B., & Vermeulen, B. (2011). Shared mental models. In *Coordination, Organizations, Institutions, and Norms in Agent Systems VI* (pp. 132–151). Springer.

Kurby, C. A. & Zacks, J. M. (2008). Segmentation in the perception and memory of events. *Trends in Cognitive Sciences*, 12(2), 72–79.

Laird, J. E. (2012a). *The Soar Cognitive Architecture*. Cambridge, MA: MIT Press.

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

Langley, P. & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence* (pp. 1469–1474).

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*.

Mahasseni, B., Lam, M., & Todorovic, S. (2017). Unsupervised video summarization with adversarial LSTM networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

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

Mathieu, J. E., Heffner, T. S., Goodwin, G. F., Salas, E., & Cannon-Bowers, J. A. (2000). The influence of shared mental models on team process and performance. *Journal of Applied Psychology*, 85(2), 273–283.

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*.

Nuxoll, A. M. & Laird, J. E. (2007). Extending cognitive architecture with episodic memory. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence* (pp. 1560–1565).

Schiller, D., Eichenbaum, H., Buffalo, E. A., Davachi, L., Foster, D. J., Leutgeb, S., & Ranganath, C. (2015). Memory and space: Towards an understanding of the cognitive map. *The Journal of Neuroscience*, 35(41), 13904–13911.

Sørmo, F., Cassens, J., & Aamodt, A. (2005). Explanation in case-based reasoning–perspectives and goals. *Artificial Intelligence Review*, 24(2), 109–143.

Stracuzzi, D. J., Li, N., Cleveland, G., & Langley, P. (2009). Representing and reasoning over time in a unified cognitive architecture. In *Proceedings of the 31st Annual Meeting of the Cognitive Science Society* (pp. 2986–2991).

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

Tulving, E. (1985). Memory and consciousness. *Canadian Psychology/Psychologie Canadienne*, 26(1), 1.

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

Turner, R. (2016). A model explanation system. In *Twenty-Sixth International IEEE Workshop on Machine Learning for Signal Processing* (pp. 1–6).: Insitute of Electrical and Electronics Engineers.

Wicaksono, H. & Sheh, C. S. R. (2017). Towards explainable tool creation by a robot. In D. W. Aha, T. Darrell, M. Pazzani, D. Reid, C. Sammut, & P. Stone (Eds.), *IJCAI 2017 Workshop on Explainable Artificial Intelligence*.