ACS Poster Collection (2018) 219–237 Published 12/2018; Paper 14

Episodic Memory: Foundation of Explainable Autonomy

David H. Ménager DHMENAGER@KU.EDU
Electrical Engineering and Computer Science, University of Kansas, Lawrence, KS 66045 USA

Dongkyu Choi DONGKYUC@KU.EDU
Aerospace Engineering, University of Kansas, Lawrence, KS 66045 USA

Abstract

In recent years, the inner workings of many intelligent agents have become opaque to users who wish to manage and collaborate with them. This lack of transparency makes it difficult for human users to understand and predict the behavior of such agents. In this paper, we present a theory of episodic memory that enables intelligent agents to remember their personal experience and make known their internal decision making process to the users. We describe an implementation of this theory in a cognitive architecture and present some initial results. We also discuss related work in episodic memory and explainable autonomy and propose directions for future research.

1. Introduction

Recently, machine learning systems have produced results that have garnered an attentive audience, especially amongst those in industry and government. Despite the often impressive performance of such systems, their opaque decision functions are severely deficient in interpretability and explainability. Most of these systems operate like black boxes, and humans can neither understand, nor predict their internal decision-making processes. Users who rely on the decisions, recommendations, or actions from such artificial intelligence systems naturally want to understand and predict their behavior. Otherwise, these individuals will not use, let alone trust them. This is especially true for autonomous agents existing over extended periods of time, because they will likely operate under intermittent supervision, and their behavior, in retrospect, may seem unintuitive to the users at first glance. In addition to understanding and predicting behavior, users will also want to provide feedback on the agent's decision-making process itself. If our goal is to build autonomous agents that collaborate with humans in a natural manner, artificial agents that can explain themselves to their human counterparts will be essential to achieve this goal.

Agents with *explainable autonomy* will help remedy this issue, providing information and justifications for its behavior and allowing users to understand and predict its decision-making processes (Gunning, 2017). We argue that explainability is a necessary feature for agents that engage in high-level reasoning, or agents that exist over extended periods of time because their rationale may not be well understood by users who wish to manage or collaborate with them. To demonstrate explainability, an agent should, upon request, be able to:

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

We believe that cognitive systems provide an ideal basis for building such agents. This paper builds on our previous work (Ménager & Choi, 2016) where we developed episodic memory for a cognitive architecture, ICARUS (Langley & Choi, 2006), and characterizes the role of this memory in facilitating explainable autonomy. Since meaningful explanations between humans often involve verbal communication, we explore the feasibility and challenges in making an explainable agent that demonstrates question-answering capability at appropriate levels of abstraction. In the remainder of this paper, we first review ICARUS and describe the episodic memory extension we have previously made in the context of the architecture. Then we present our take on how episodic memory enables explainable behavior, followed by an example ICARUS agent that uses episodic memory and explains its internal decision-making processes to users in a popular game domain, Minecraft (Johnson et al., 2016). Finally, we present related and future work before we conclude.

2. ICARUS Review

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

2.1 Representation and Memories

ICARUS distinguishes two main types of knowledge, *concepts* and *skills*, which represent semantic and procedural knowledge, respectively. The architecture stores the definitions of these in their respective long-term memories. 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 sub-relations. More formally, a *primitive concept* is defined over a finite set of first-order propositions, P, as $c_i = \langle \lambda, \epsilon \rangle$ where $\lambda \in P$ is the concept head and ϵ denotes elements to pattern match against the world state,

$S \subset P$. If we let C_p be the set of such primitive concepts, then a *non-primitive concept* is defined over $P \cup C_p$ as $c_j = \langle \lambda, \epsilon, \gamma \rangle$ where γ denotes the subrelations. We can further define more non-primitive concepts over $P \cup C_p \cup C_n$, where C_n is the set of non-primitive concepts.

Table 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 itself and the inventory, *hotbar*. The second concept, *front-of*, is a non-primitive one that describes the situation where an entity is in front of the agent. It requires a perceptual match for the agent itself, *?self*, a belief of an entity *?o1*, and test conditions against the positions of the two. The last definition, *on-horizontal-axis*, is another non-primitive concept that uses both of the earlier definitions as subrelations 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 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's 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. Given the finite set of actions, $\mathcal{A}$, a *primitive skill* is defined as $k_i = \langle \epsilon, \gamma, \alpha, \eta \rangle$ over $C \cup S$ where C is the set of concepts and S is the propositional world state. Here, $\epsilon \subseteq S$ denote one or more pattern match conditions, $\gamma \subseteq \{\lambda | \langle \lambda, \cdot \rangle \in C\}$ are preconditions, $\alpha \subseteq \mathcal{A}$ are actions, and $\eta \subseteq \{\lambda | \langle \lambda, \cdot \rangle \in C\}$ are the inteded effects. If we let K_p be the set of primitive skills, a *non-primitive skill* is $k_j = \langle \epsilon, \gamma, \alpha, \sigma, \eta \rangle$, where the additional component σ are the sub-skills from K_p and other non-primitive skills.

Table 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. Then the agent will be carrying the resource as an effect.

Table 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 stores the instances of its concepts and skills in separate short-term memories. The concept instances, or *beliefs*, are in the form of a tuple, $\langle c_k, \beta \rangle$, where c_k is a concept definition and β is a set of bindings. Similarly, the skill instances, or *intentions*, are in the form of a tuple, $\langle k_l, \beta \rangle$, where k_l is a skill definition and β is a set of bindings. Often, there are also certain bookkeeping information added to beliefs and intentions for the architecture's internal use.

2.2 Cognitive Processes

ICARUS operates in cycles as shown in Figure 1. At the beginning of each cycle, the architecture receives perceptual information from the world as a list of objects with their attribute-value pairs and deposits them into the perceptual buffer. The system then elaborates a belief state by invoking the *conceptual inference* process, during which the system computes the instances of its concepts that are true based on the perceptual information received. 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, which is one of the two short-term memories in the architecture.

Once the system finds the state of the world by inferring all possible beliefs, it performs the optional goal reasoning and proceeds to *skill execution* to select a skill that achieves one or more of

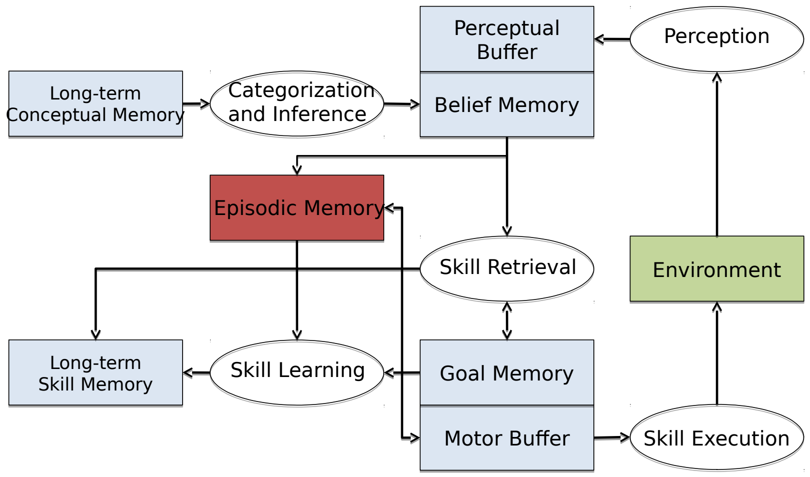

Figure 1. ICARUS cycle diagram with the episodic memory extension.

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 instance that will achieve one or more of its goals, ICARUS sets it as its intention and stores it in its goal memory, another short-term memory in the architecture. The system will then execute the intention either by invoking the direct actions if it is a primitive, or by following one of the sub-skills and instantiating it recursively.

If, however, the architecture is unable to find a skill instance that works, it invokes its problem solver to find a solution to the problem. The default method is a 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 concepts and skills upon the successful incremental execution of the solution. 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 readers to our previous work (Langley & Choi, 2006; Choi & Langley, 2018) for more detailed discussions on these processes.

3. Episodic Memory in ICARUS

Through a recent extension (Ménager & Choi, 2016), ICARUS has a long-term, cue-based memory that the architecture can use to encode and retrieve episodes. As shown in Figure 2, the architecture organizes its episodic memory $E = \langle \rho, \mathcal{F}, \mathcal{T} \rangle$ in a compound structure composed of a state-intention cache ρ, a concept frequency forest $\mathcal{F}$, and an episodic generalization tree $\mathcal{T}$. In this section, we describe these components and the processes that work over them.

The first component, a *state-intention cache* (ρ), is an ordered sequence of belief state and intention pairs. This cache stores a complete, detailed history of what the agent has observed and

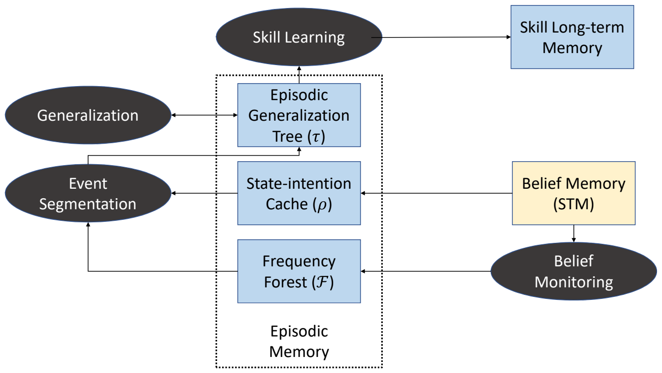

Figure 2. Block diagram depicting ICARUS's episodic memory components and information flow starting from the belief memory.

what the agent has done in the world. In this regard, the state-intention cache is a reminiscent of the episodic buffer proposed by Baddeley (2000). The second component of ICARUS's episodic memory is a *concept frequency forest* ($\mathcal{F}$). This enables the agent to maintain the statistics of what happens in the world and allows it to detect any interesting events that occurs. There can be many different ways to do this, but our current implementation uses the agent's locations as the indices for this forest structure and generates expectations of the agent at particular locations. The last component in the episodic memory is an *episodic generalization tree* ($\mathcal{T}$). Episodes are stored in this tree according to their similarities. The root node of this tree is the most generic episode that can instantiate any of its childrens. Each parent episode is a partially variablized version of its children, representing a generalization of the children episodes.

3.1 Creating Episodes

Using these memory elements, ICARUS stores its experience for long-term use. When humans observe the world, they maintain expectations about the near-term future. Kurby & Zacks (2008) argue that humans perceive the beginning of a new episode, when their expectations are violated because it then seems that some new event is taking place. In our current implementation, we have a location-indexed concept frequency forest that generates expectations. It maintains conditional probabilities of state elements given the location that the agent is in. The agent sets two thresholds: one for positive expectations and one for negated expectations. Any belief with a conditional probability, given the location, that is greater than the positive threshold is said to be expected to happen at that location. Any belief with a conditional probability, given the location, that is less than the negated threshold is expected not to be in the belief state. A belief that violates an expectation is considered interesting to the agent, which prompts the system to create an episode using the process shown in Table 3.

An episode created in this manner is a tuple, $\langle B_s, B_e, \Sigma, \psi \rangle$, where B_s is the start state of the episode, B_e is the end state of the episode, Σ is the set of interesting beliefs in B_e, and ψ is a count for the number of times the episode has occurred. Note that an episode can happen multiple times, and B_s and B_e are stored as lists of pointers, which locates the states in the state-intention cache where this episode occurred. Theoretically, the start and end states can be arbitrarily far apart,

Table 3. Pseudocode for creating a new episode.

```
1: $\rho$ is beliefs-intention cache
2: $loc$ is current location
3: $B_c$ is current belief state
4: $\iota \leftarrow$ agent's executed intention
5: $B_{prev} \leftarrow$ last state in $\rho$
6: $\rho \leftarrow \rho.add(B_c, \iota)$
7: $sigs \leftarrow$ GETINTERESTINGBELIEFS($B_c$, $loc$)
8: if not NULL($sigs$) then
9:     $\varepsilon \leftarrow$ MAKEEPISODE($sigs$, $B_c$, $B_{prev}$)
10:    $\mathcal{T} \leftarrow$ INSERT($\varepsilon$, $\mathcal{T}$)
```

but they are assumed to be consecutive in the current implementation. The interesting beliefs field contains all the positive and negated beliefs that violated the agent's expectations.

3.2 Inserting Episodes

Once an episode is created through the event segmentation process, it is inserted into the episodic generalization tree at a proper location. Table 4 traces how episodes are inserted into the episodic generalization tree. Suppose the generalization tree contains several episodes. Γ is a list of sibling episodes under parent $\varrho \in \mathcal{T}$ If $\forall \varepsilon_i \in \Gamma, (\varepsilon_i, \varepsilon) \notin E$ then $(\varrho, \varepsilon) \in E$. That is ε becomes a child of ϱ. A new episode has successfully been encoded into the episodic memory. If $\exists \varepsilon_j \ni \varepsilon_j = \varepsilon$, then the counter for ε_j increments by one and ε is not inserted.

On every cycle, ICARUS records the belief state and executed intentions into the episodic cache and updates $\mathcal{F}$. When the agent infers one or more interesting beliefs, it encodes a new episode. The root node of the generalization tree is the most general episode and is allowed to have an arbitrary number of children. Under the root, episodes are grouped according to their structural similarities. Two episodes e_1, e_2 are structurally similar if their interesting beliefs unify, which implies that there exists a binding set that transforms the interesting beliefs of e_1 to those of e_2 and vise versa. Each child is a *k*-ary tree where $k \in \mathbb{N}$. Episodes become more specific at each decreasing level of the tree and there are fully instantiated episodes at the leaf nodes. We describe the generalization process in more details below.

3.3 Generalizing Episodes

ICARUS supports generalization of the episodes during insertion of an episode, ε_i. As discussed before, an episodic generalization tree is organized by structural similarity. Table 5 shows how the system performs this task. Observe that on line 4, two sibling episodes $\varepsilon_i, \varepsilon_j$ generalize if and only if $\exists$ episode ε_g such that $(\varepsilon_g, \varepsilon_i) \in E$ and $(\varepsilon_g, \varepsilon_j) \in E$, but $(\varepsilon_i, \varepsilon_g) \notin E$ and $(\varepsilon_j, \varepsilon_g) \notin E$. This means that the variablize function returns a variable binding set for ε_g, which is tested for validity

Table 4. Pseudocode for inserting a new episode.

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

on line 5, such that ε_g unifies with its children, $\varepsilon_i, \varepsilon_j$, but its children cannot unify with it because they contain more specified bindings. If ε_g exists, ICARUS tests to see if it is still more specific than the parent of ε_i, ε_p. If so, then ε_g becomes a child of ε_p, and ε_g's children become $\varepsilon_i, \varepsilon_j$. The count for a generalized episode is the summation of the count of its children.

Based on our discussion of the episodic memory and its processes, we explain our understanding of explainable autonomy and the reasoning behind using episodic memory to enable agents with such capability in the next section.

4. Explainable Autonomy using Episodic Memory

In this section, we provide more detail about the episodic phenomena we wish to model, and present some theoretical postulates to explain them. We provide insight on feasible cognitive structures for achieving explainable autonomy using episodic memory, and lastly we present preliminary evidence showing that agents with episodic memory capabilities can in fact be explainable agents. The behaviors we think are most suited for explainable autonomy are:

- *Summarization*: People summarize their past experiences at appropriate levels of abstraction. They discuss information at appropriate levels of detail, rather than spew low-level informa-

Table 5. Pseudocode for generalizing episodes.

```
1: if NOT(NULL($\varepsilon_p$)) then
2:     children ← GETSUBEPISODES($\varepsilon_p$)
3:     $\varepsilon_g \leftarrow \emptyset$
4:     for $\varepsilon_j \in$ children do
5:         $\varepsilon_g \leftarrow$ VARIABLIZE($\varepsilon_j$, $\varepsilon_i$)
6:         if VALIDGENERALIZATION($\varepsilon_g$, $\varepsilon_p$, $\varepsilon_j$, $\varepsilon_i$) then
7:             $\varepsilon_g$.SETSUBEPISODE($\varepsilon_i$)
8:             $\varepsilon_g$.SETSUBEPISODE($\varepsilon_j$)
9:             $\varepsilon_p$.SETSUBEPISODE($\varepsilon_g$)
10:            break
```

tion about quotidian happenings that may overwhelm their counterparts. People can provide more detailed summaries whenever someone asks, and people are usually prepared to naturally answer follow-up questions about their thoughts and activities.

- *Question answering*: People are also able to answer a variety of questions about their internal decision-making processes and behaviors. Amongst other things, people can explain:
  - How and why goals were achieved;
  - How and why actions and behaviors were executed; and
  - Any alternative methods for achieving goals and acting

  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*: Humans 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.

Our goal is to create agents that demonstrate this range of behaviors. To do this, we supplement the behavioral aspects related to explainable autonomy with theoretical postulates that attempt to explain them:

- *Explainability is facilitated by episodic memory 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.

- *These contents describe an agent's internal and external environments.* Episodes contain descriptions of an agent's beliefs about the world as well as descriptions of the agent's goals and intentions for achieving its ends.

- *Episodic memory structures are domain independent.* Cognitive structures such as goals, beliefs, intentions, and episodes are abstract symbolic structures that eventually make contact with grounded predicates.

The first postulate in the theory emphasizes the role of knowledge and information processing in explainable autonomy. In contrast to popular machine learning techniques, the cognitive systems approach recognizes 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. The second and third postulates expand the first by specifying some representational properties of the mental structures reasoned over by episodic memory processes. Based on this understanding of explainable autonomy, we designed an ICARUS agent that has the capacity in the context of Minecraft.

5. ICARUS Agent with Explainable Autonomy

In this section, we present our findings on how episodic memory lends itself to answer questions about the personal past in actual scenarios. The episodic memory in ICARUS supports cue-based retrieval to allow the agent to recall its personal past. Retrieval operates on the generalization tree and is triggered deliberately. Deliberate retrieval is goal-directed, meaning that the system has skills for remembering. For instance, consider the skill for answering how a goal was accomplished:

Table 6. 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))))
```

This skill allows an agent to a question about how it achieved a goal. Given that the questioner is wants to know how the goal was achieved, if the system explains how, then the questioner will no longer be uninformed about the goal. The action **explain-goal-achievement*, outlined in Table 7 involves the agent retrieving episodes from its memory, processing them, and synthesizing text that gets output to the screen. Line 1 invokes the episodic memory process shown in Table 8 to remember all relevant experiences to the retrieval cue, $goal$. Once the agent's stored past plans are collected, the system fills out a text template that answers the question, as shown in Line 3. Note that all the information needed to answer a question is already stored in the agent's intention, so answering a question is only a matter of identifying the necessary information contained therein.

Using these facilities, we designed and tested our explainable agent in the popular video game Minecraft (Johnson et al., 2016). Minecraft is an open-world, sandbox game. Players can create and experience new worlds by collecting resources, building structures, discovering new lands, and so on. We chose Minecraft for a number of reasons. The first is that the agent lives in a dynamic

Table 7. Pseudocode for explaining goal achievements.

```
1: **for** intention in REMEMBER($\rho$, $\tau$, $goal$) **do**
2:     **if** $goal \in$ intention.effects **then**
3:         SAY("I did ?x to achieve ?y", intention.head, goal)
```

Table 8. Pseudocode for remembering the agent's past through episodic retrievals.

```
1: remembered $\leftarrow$ RETRIEVE($\tau$, goal)
2: idx $\leftarrow \emptyset$
3: ints $\leftarrow \emptyset$
4: **for all** episode in remembered **do**
5:     collect episode.cache-pointers into idxs
6: **for all** idx in idxs **do**
7:     executed-intention $\leftarrow$ $\rho$.intentions[idx]
8:     **if** not NULL(executed-intention) **then**
9:         collect executed-intention into ints
10: return ints
```

environment. Events happen outside of the agents control, and the agent can interact with non-human and human players. Second, the dynamic nature of the game forces the agent to construct robust plans and rely on its ability to adapt to change. Lastly, the Minecraft world is a continuous space, so the agent has to understand and describe continuous events in discrete terms.

Table 9 shows some sample definitions of concepts for this domain. It reveals that the concept hierarchy can describe 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. On the other hand, the last concept states that carrying planks and sticks are all part of carrying materials for making a sword. As Figure 3 shows, we placed our agent in a room with a zombie and told the agent to defend itself against the threat. After the agent kills the zombie, we asked the agent a series of questions about what happened. The following is a transcript of the interaction, where we present English translations of the questions in bold, the perceptual representation of the question next to the English, and then the agent's response in bullet points:

- How did you achieve a situation 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)).`

Table 9. 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)))
```

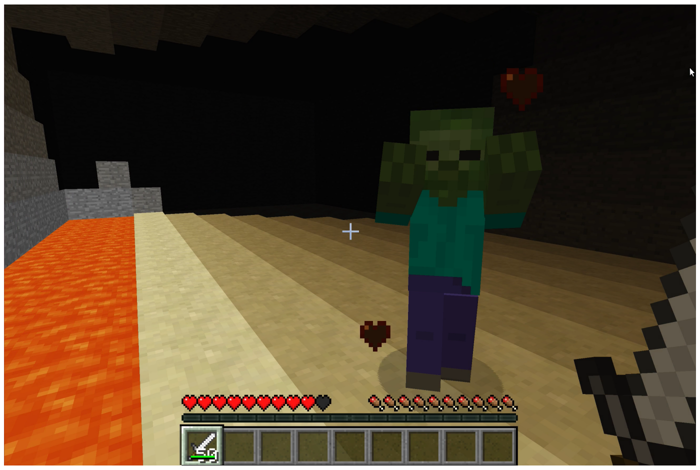

Figure 3. ICARUS agent fighting a zombie in Minecraft.

In this response, the system exposes that it did two different procedures for clearing the room of the zombie. Next we discover why two different procedures were followed.

- Why did you 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))
```

- Why did you 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 how the system reveals that there are two different ways to rid the room of a zombie. Notice also that the agent makes known alternatives that it could have chosen and explains why they were not selected. Next, we find out about the structure of (KILL ?O1) and (ATTACK ?O1).

- How did you actually attack zombie1?
  (question attack_zombie1 type intention how t)

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

  The attack skill specifies low-level details on what motor functions to perform to complete the task.

- How did you actually 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 from a room than attack, because (ATTACK ?O1) is contained as a subskill of (KILL ?O1). In Minecraft, zombies move around or move to the player they are trying to eat so, at times the more generic (KILL ?o1) procedure is applicable, and other times, the zombie is in a situation, where the system can directly execute primitive actions to achieve its ends.

- Why did you nominate being armed and dangerous as a goal to achieve?
  (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 question, the agent is explain 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 nominate goals to achieve which it thinks will allow it to achieve its top-level goal.

The debriefing could go on in more detail, but we end it here. Our ICARUS agent is able to describe in detail its decision-making process, and this allows the user to predict its behavior and make expectations about what tasks the system can accomplish. There is still more work to complete, but we believe this is a promising direction. Further note that a system without episodic memory capabilities will not be able to answer questions about its behavior in this level of detail. So, it seems that researching the role of episodic memory and its related processes is a promising direction for creating explainable collaborative agents.

6. Related Work

Without a doubt, our work is influenced by many previous work. Here we discuss two main groups of them, one for episodic memory and another for explanations. A few previous work in cognitive systems dealt with episodic memory systems. For example, the Soar cognitive architecture has an episodic memory that contains sequentially ordered snapshots of the agent's working memory (Laird, 2008; Nuxoll & Laird, 2007). Like our previous work (Ménager & Choi, 2016), they present a series of design decisions that are generically supported by psychological evidence.

In addition to these cognitive architectures, work in incremental concept formation is also relevant to our work. One of the most famous examples of such a system is COBWEB (Fisher, 1987). Like most of its predecesors, COBWEB induces a concept hierarchy from training instances in an online fashion. Training instances are represented as lists of attribute value pairs with associated probabilities and are sorted through the tree in a top-down manner using the notion of category utility. One of its limitations was that it could only represent symbolic values, and thus was unable to reason about numeric states. In response to this, Gennari et al. (1989) developed CLASSIT based on COBWEB that handles numeric attributes by modelling the occurrence of an attribute value as a normal random variable. But CLASSIT, unfortunately, was unable to model symbolic attributes because of this. Most recently, TRESTLE (MacLellan et al., 2015) succeeds in performing incremental concept formation with both symbolic and numeric values. The system is built on top of COBWEB.

Some of the main differences between these systems and ICARUS episodic memory is that our system does not represent probabilistic states. Rather, we store probabilities in the concept frequency forest, and our hierarchy is induced by variablizing individual differences amongst lower-level episodes. Secondly, with the exception of TRESTLE, these systems are not actually episodic memories because they can only sort single states at a time into memory. TRESTLE (MacLellan et al., 2015) overcomes this using what it calls *components*, so each state, theoretically be represented as a component. In order for this to work, however, each state must be represented as a list of attribute-value pairs. So, TRESTLE cannot represent propositional states using components. Thirdly, none of the systems are integrated into a cognitive architecture. So, these systems do not have any ability to store the agent's plans, because no plans are created.

Finally, a long short-term memory (Hochreiter & Schmidhuber, 1997) in recurrent neural networks can remember values for arbitrarily long time intervals. Mahasseni et al. (2017) show that this technology can be used to summarize experiences. It takes as input a video, then using a series of these memories, it returns a sequence of frames that represent the video. The ability to summarize video into a series of key frames bares resemblance to individual experiences contained in episodic memory. One key shortcoming with the work is that it only summarizes video at one level of detail, and the system does not generalize to new domains.

There are also many related work in the context of explanation generation. The case-based reasoning community has a rich history of building systems that explain their internal motivations and justify their actions (Doyle et al., 2003; Sørmo et al., 2005). One early work in case-based reasoning attempted to explain how a system designed physical devices to users (Goel & Murdock, 1996). This work used meta-cases represented using the Task-Method-Knowledge (TMK) model to 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 an agent's design choices. our work, on the other hand, describes a domain independent episodic memory integrated inside a cognitive architecture, while their system is tied to explanations of design choices for physical devices. The system's plans, which can be viewed as internal explanations are repurposed into external, text-based explanations that people can consume. In addition, explanatory ability is encapsulated inside skills, which are more similar to goal-task network formalisms (Alford et al., 2016).

Also, the automated planning community often uses the planning domain definition language (McDermott et al., 1998) to represent and solve planning problems. It represents actions using STRIPS operators (Fikes & Nilsson, 1971) and writes state elements as grounded predicates. This language has been widely used, leading to several popular planning systems such as the SHOP (Nau et al., 1999) family of planners and the fast downward planning system (Helmert, 2006). Although the STRIPS operators these systems rely on are internally interpretable to the planning system and to domain engineers, the resultant plans are simply a list of actions the agent will take to achieve a goal. The rationale for why these actions are taken is generally lost once the plan is created.

There are also efforts attempting to make integrated agents more predictable to human users. Floyd et al. (2014) investigated how a robot can adapt its behavior based on estimates of its trustworthiness. This work does not consider how a system can explain its behavior to others. Rather, it attempts to change its behavior to the preferences of the operator. This ability is in fact a desirable trait in robots, but systems like these would be more valuable if they could also engage their operators about their decision-making process.

More recently, there has been a push towards explainable machine learning systems (Aha et al., 2017). Fox et al. (2017) discuss why model-based planning systems need to have explainable components. Although the paper takes a similar philosophical position to us, the paper is less theoretically strict on how explanations are generated in the first place, and provides no implementation of a planning system with these explainable qualities. Furthermore, it seems that the authors assume that the end user asks questions about the agent's current plan, but not any other plan that was executed in the past. Therefore, it makes unrealistic assumptions about explainable AI, since no explanations of past behavior are required.

In robotics, researchers developed a robotic system that can use and design new tools based on past experiences (Wicaksono et al., 2017). The system is able to learn action models for tools via demonstration. Similar to our work, they demonstrate explainable behavior by storing the agent's plans for achieving goals. Note that this work essentially creates an episodic memory of its own for storing the agent's plans. Next, the system seems to designed only for tool creation and use. Lastly, the system only supports answering questions about its behavior, but cannot answer questions about why it chose to pursue specific goals. Furthermore, the agent's actions aren't discussed in relation to the goals they achieve. So, it assumes that the human will have understanding of the rationale for why the agent wants to behave the way it does.

Sheh (2017) build an explainable robot system for navigating rough terrain. The system uses behavioral cloning and logical rules stored in decision trees to be able to explain to users its internal decision-making process. Because of the use of the decision trees, the explanation the system gives are attribute-centric, whereas the explanations ICARUS provides are based around the operators with respect to the goals they achieve.

Perhaps, one of the most similar work is Johnson (1994). They presented Debrief, a system that can explain 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. Unfortunately, the work makes few theoretical commitments to episodic memory processes and representations, 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 explanation. Our episodic memory does not need such specifications and works automatically.

7. Directions for Future Research

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 towards meaningful progress. In the coming years, we would like to: enable the system to summarize multiple episodes into one explanation; build agents that share one episodic memory across multiple agents; build agents that accept feedback from users about explanations; incorporate shared mental models into our theory; and conduct human subject tests to validate the quality of explanations and learned behavior of the system.

In this section, we walk through each future work item in turn, providing details and ideas on what our goals are and how to accomplish them. First, our agent can already explain its behavior in an abstract manner to some extent, but we want our system to have the ability to synthesize its history into summaries. We want our systems to be able to tell users stories about the events they experience. For example, whenever a user wants to know“what happened”, the agent should be able to retrieve a series of episodes from the episodic memory and weave their components together to explain large segments of time.

Second, we would like to investigate the implications of sharing episodic memory across multiple agents acting in an environment. 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.

Third, in addition to explaining its rational for why an agent takes action, we would like our system to adapt to user preferences by adhering to feedback. In general this is a difficult endeavor, but 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, we would like our system to be able to use shared mental models (Mathieu et al., 2000; Jonker et al., 2011) 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 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.

Finally, 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. 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.

8. Conclusions

Continued application of artificial intelligence systems in the field will be limited as long as humans cannot understand, trust, and manage autonomous systems. Currently, humans are unable to predict when a system will make mistakes or successfully achieve its end. Episodic memory provides key insights into this problem by providing a window into artificial intelligence systems' internal decision-making processes. In this paper, we discussed how episodic memory in a cognitive system can play an important role in constructing intelligent agents that can explain their behavior.

We described a series of behavioral aspects, namely, the abilities to summarize events, answer questions, and engage in hypothetical reasoning, that enable agents to demonstrate explainability. Using a psychologically inspired theory that supports these behaviors, we built an ICARUS agent to demonstrate such capabilities in Minecraft for demonstration. Our work suggests that thinking over episodic memory contents allows an agent to explain its behavior using constructs (i.e. goals and intentions) that humans readily understand.

We find that episodic memory can help extend an agent's ability to explain its internal decision-making process. These abilities are essential for fostering a harmonious integration of intelligent agents in society. We believe episodic memory is a fundamental component of human cognitive ability, and the 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. *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.

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

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

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.

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*, 139–172.

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

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

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

Goel, A. K., & Murdock, J. W. (1996). Meta-cases: Explaining case-based reasoning. *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.

Helmert, M. (2006). The fast downward planning system. *Journal of Artificial Intelligence Research*, *26*, 191–246.

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

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

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

Johnson, W. L. (1994). Agents that learn to explain themselves. *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 M. D. Vos, N. Fornara, J. V. Pitt, & G. Vouros (Eds.), *Coordination, organizations, institutions, and norms in agent systems VI*, 132–151. Springer.

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

Laird, J. E. (2008). Extending the Soar cognitive architecture. *Frontiers in Artificial Intelligence and Applications*, *171*, 224.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. *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. *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. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

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*, 273–283.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). *PDDL-the planning domain definition language*. Technical report, Yale Center for Computational Vision and Control, Yale University, New Haven, CT.

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

Nau, D., Cao, Y., Lotem, A., & Munoz-Avila, H. (1999). SHOP: Simple hierarchical ordered planner. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (pp. 968–973). Morgan Kaufmann.

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

Sheh, R. (2017). "Why did you do that?" Explainable intelligent robots. *2017 AAAI Workshop on Human-Aware Artificial Intelligence* (pp. 628–634).

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

Wicaksono, H., Sammut, C., & Sheh, R. (2017). Towards explainable tool creation by a robot. *Proceedings of the 2017 IJCAI Workshop on Explainable AI* (pp. 63–67). Melbourne, Australia.