The Goal Re-activation Problem in Cognitive Architectures

by

Ning Hui Li

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2016

Doctoral Committee:

Professor John E. Laird, Chair
Professor Satinder Singh Baveja
Professor Edmund H. Durfee
Research Scientist Christian Lebiere, Carnegie Mellon
Professor Richard L. Lewis

2016

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF APPENDICES

ABSTRACT

The Goal Re-activation Problem in Cognitive Architectures

by

Ning Hui Li

Chair: John E. Laird

Intelligent agents in the real world have to manage multiple goals. However, the pursuit of some goals may only be possible under specific conditions which, if not met, requires the agent to suspend the goals for future re-activation. In cognitive architectures, suspended goals are stored in long-term memory; however, this prevents agents from automatically recognizing future opportunities to complete those goals.

This thesis characterizes this previously-unidentified problem as an instance of a more general circular dependency between the retrieval and use of knowledge in cognitive architectures: the agent must recognize that a goal is relevant to retrieve it, but cannot recognize it as such without retrieving it in the first place. We apply this characterization to develop preemptive and spontaneous retrieval strategies for goal re-activation in the Soar cognitive architecture. Evaluation of these strategies in an abstract domain shows that the spontaneous retrieval strategy dominates the other strategies, achieving higher goal completion rates in fewer operations, although it is also not without failure cases. Both types of strategies not only provide solutions to the goal re-activation problem, but also pave the way for further exploration of how intelligent agents can access the right knowledge at the right time.

CHAPTER 1

Introduction

1.1 Goal Re-activation in Cognitive Architectures

This thesis examines the question of how artificial agents manage multiple, concurrent goals when a subset of the goals cannot be immediately pursued. In particular, it looks at the problem of *goal re-activation*: if a goal was not originally relevant and was stored, how does the agent detect that the situation has changed so that the goal is now relevant and should be acted upon? Consider the following scenarios:

- An agent is tasked with conveying a message to someone, but it doesn't know their schedule. While performing some other task, the recipient of the message coincidentally appears.
- An agent is tasked with buying milk from the grocery store, but is currently on its way to a work shift patrolling a building. During its patrol, the robot happens to go by a grocery store that sells milk.

In these scenarios, the agent is given a goal (also known as the *prospective goal*) that should only be considered for pursuit under some specific conditions, called the *target*. For example, a target for the goal of delivering a message may be when the recipient is in sight; the target is *satisfied* when the described situation occurs. The example goals above are also one-time — while the target may again be satisfied after the goal is completed (that is, the agent may run into the recipient again), the agent should not pursue the goal again (the agent should not deliver the message twice). Crucially, when the agent is first given the goal, the target is *not* satisfied, and therefore the goal cannot be immediately pursued. Furthermore, the agent is unable to predict when the target will be satisfied. Finally, the prospective goal is not important enough for the agent to attempt to immediately satisfy the target — the agent does not go looking for the recipient of the message, nor immediately go to the grocery store. Thus, in the mean time, the agent pursues other goals (*foreground goals*), during which the target of the original prospective goal may be satisfied, giving

the agent the option of pursuing the prospective goal. Such scenarios can occur for an agent that has multiple, non-hierarchical goals, and whether a goal can be pursued is out of the control of the agent, and the opportunities for completion are unpredictable.

To summarize, the scenario with which this thesis is concerned has these characteristics, described with terminology from literature [7]:

A1 The agent has a one-time goal. This corresponds to achieve and perform goals in a classification of goals, where the agent must take some action once, either to *achieve* some world state, or simply for the *performance* of the action itself. After the goal is completed, the same circumstance may require a different response from the agent. Achieve and perform goals contrast with maintenance goals, where the agent must ensure that some environmental state remains true — and more importantly, that the agent must act whenever the state does *not* hold. Although an identical achievement goal could be instantiated after the goal is first complete, we consider these separate goals and do not attempt to take advantage of this possibility.

A2 The goal cannot be immediately pursued due to the absence of its target. That is, while the agent *adopts* the goal, it does not become an *option*, and is instead *suspended*. We treat this scenario as a special case of task resumption — where an agent, after beginning some goal, is unable to complete it for some reason [59] — except in this case, the agent has not made any progress at all. The techniques discussed in this thesis apply to both scenarios.

A3 The agent does not know when the target will be satisfied (such that the goal can be pursued), and does not have a model of the environment.

A4 The agent has multiple goals, and these other goals are pursued while waiting for the target of the prospective goal to be satisfied. This assumption rules out the possibility of devoting all of the agent's resources to the prospective goal.

The goal re-activation problem is the problem of how the agent detects a change in the situation that makes it possible for the goal to be pursued. The focus of this thesis is on developing computational strategies for recognizing that a goal can be pursued and is a possible choice for the agent. Whether the agent chooses to pursue the goal, or whether the agent's plans allow it to complete the goal are outside the scope of this thesis.

Although goal re-activation can be tackled on an agent-by-agent basis using decision theory and planning, this thesis considers the problem in the context of cognitive architectures, which is an *agent framework*. Agent frameworks, such as cognitive architectures and the Belief-Desire-Intention (BDI) formalization, aim to provide developers with tools to quickly create intelligent agents that

operate in diverse domains [29]. The goal re-activation problem is relevant to the agent framework community for two reasons. First, since the ability to pursue and complete goals is an important part of many agents, it may be beneficial for agent frameworks to standardize the mechanisms for agents to reason about goals. Second, the structure of agent frameworks may determine how the goal re-activation problem can be solved, since the organization of their components determines how knowledge — including goals — are stored and processed.

Cognitive architectures such as Soar and ACT-R share multiple structure similarities. They store knowledge in separate *memories*, and many architectures organize declarative knowledge into *memory hierarchies*, analogous to the memory hierarchies in computer architectures. A depiction of this hierarchy is shown in Figure 2.5 and this organization is further discussed in Section 2.3. Most importantly for the goal re-activation problem, only knowledge stored in the memory at the top of the hierarchy (*working memory*) can directly affect reasoning and behavior. At the same time, knowledge in working memory is automatically removed (or *forgotten*) over time, although it can be recovered (or *retrieved*) from the next level in the hierarchy (known as *long-term memory*). This structure mirrors the relationship between registers and the cache, or the cache and RAM, in computer architectures.

This division of knowledge stores and the processes that transfer knowledge between them present additional challenges for goal re-activation. Consider the message-delivering scenario above. When the agent first receives the message it must deliver, the goal of delivering the message is stored in working memory. Since the recipient is absent, however, the goal target is not satisfied and the agent cannot immediately pursue the goal. Over time, the goal is removed from working memory and stored in long-term memory. When the recipient appears, however, the change in the situation means that the goal can now be pursued. But while the agent may see the recipient, it cannot connect its perception of the recipient to its representation of the recipient as the target of the message delivery goal, since the goal and its target remain in long-term memory and are unavailable for use in reasoning. That is, while in theory the target of the goal is satisfied, *the agent does not recognize it as such*, and therefore the agent fails to pursue the goal.

In sum, this thesis is concerned with architectures where the following statements hold. These statements are a mix of assumptions that define a class of cognitive architectures, and conventions within the cognitive architecture community. The exact epistemic status of these statements is discussed in Section 2.4.

A5 Suspended goals are represented symbolically and declaratively, and are subject to the remaining assumptions describing the processing of declarative knowledge by the architecture. In other words, there are no architectural mechanisms that process goals specially.

A6 Only declarative knowledge in working memory can be used for immediate decision making;

this implies that the goal must be in working memory when it is used for decision making. Specifically, the goal must be in working memory during the encoding, initiation, and completion stages of the goal's life-cycle (see Section 2.1.1).

A7 Declarative knowledge in working memory is automatically removed over time. Thus, although goals are initially created in working memory (A6), they are not automatically maintained in working memory indefinitely.

A8 Knowledge that is automatically removed from working memory can be retrieved from long-term memory. This implies that the forgetting of goals in and of itself does not mean the goal cannot be completed, only that it must be retrieved into working memory when the goal is required for decision making — for example, when the agent must decide whether to pursue the goal.

A9 The only mechanism for long-term memory retrieval is deliberate cued retrieval.

A10 Rules automatically match on and modify declarative knowledge in working memory.

A11 The agent cannot directly modify procedural memory, including the modification and deletion of existing rules. The one exception to this is the ability to create rules.

At a high level, the main research question of this thesis is therefore the following: Given the structure of memory in cognitive architectures, what are strategies for ensuring that a prospective goal is in working memory when its target is satisfied, such that the agent can pursue the goal?

1.2 General Approach

The field of cognitive architectures aims to discover general mechanisms that are sufficient for generally intelligent agents. A more detailed account of the methodology of cognitive architecture research is given in Section 2.4, but a brief introduction is required to understand the layout of this thesis. Research often cycles between developing agents that performs a novel task, and the development of the architecture to support more sophisticated agents — challenges in the former drives advancements in the latter. A new task or a new *cognitive capability* is analyzed theoretically, which involves identifying the architectural components that can be involved in the solution. For example, if a task requires the agent to acquire knowledge online, this may imply that the long-term memory of the agent must be used; a deeper analysis may also consider how this knowledge interacts with other architectural components. With this understanding, the existing mechanisms of the architecture are then used to attempt to support the capability, and the result evaluated. The goal of this attempt is to determine, either positively or negatively, whether the architecture in its

current state is sufficient (in terms of effectiveness and efficiency) to support the desired capability. Finally, if the architecture is found wanting, new mechanisms may be added to the architecture. While the inability of the architecture to provide the capability is sufficient to justify the new mechanism, it would be ideal if the mechanism was also beneficial to other tasks beyond its role as a specialized module for the desired capability. Often, inspiration is taken from other fields of artificial intelligence and from disciplines such as psychology and neuroscience, to provide evidence that the mechanism is general. If this is the case, the architecture with its new mechanism is then evaluated for the cognitive capability again. Over time, these experimental mechanisms are adopted into the architecture proper, and a new cognitive capability is found to drive the next round of development.

This research and the layout of this thesis follow this methodology. Since the goals and restrictions of cognitive architectures constrain and define much of this work, Chapter 2 first provides background on the goal re-activation problem, then goes into detail about existing cognitive architectures. Most of the work in this thesis is done in the Soar cognitive architecture, but its many similarities with other architectures, and with the ACT-R cognitive architecture in particular, suggests that the results are transferable and are relevant to a wider audience. After the major components and processes of Soar are defined, Chapter 3 begins with a theoretical analysis of goal re-activation. Despite the importance of re-activating goals, this has not been a problem that has received much attention in the artificial intelligence community. As a result, inspiration is taken from psychology, where human goal re-activation is studied as *prospective memory*. Work in that field, especially the taxonomic organization of human strategies, paved the way for the first major result of this thesis: that goal re-activation is difficult due to a circular knowledge dependency between the retrieval of goals and the recognition of their pursuability. Although this result may seem obvious, the community has yet to tackle problems where this dependency causes significant problems; the novelty of this characterization means that there has been no attempt to adapt existing solutions to the memories of cognitive architectures.

The strategies developed in this thesis roughly resemble a different circular dependency problem, namely, that of detecting input/output events in operating systems via the use of polling and interrupts. Chapter 4 describes *preemptive strategies* that aim to retrieve goals into working memory before they are needed, by actively ensuring that goals are frequently in working memory. This was the first strategy we explored, and its initial success led to the development of an abstract domain that allow more controlled variations in environmental properties, such that the limits of the strategies can be tested. Chapter 5 describes the *spontaneous strategy* that uses a new architectural mechanism to retrieve goals at the time they are needed; since it is long-term memory that signals the agent with a relevant goal, it has parallels with the use of interrupts in computer systems. Chapter 6 then describes how these two strategies can be combined, and compares all strategies evaluated to

form a meta-strategy for when each strategy should be used.

Finally, to conclude this thesis, Chapter 7 reviews the contributions of this thesis. Goal re-activation is a new problem for cognitive architectures, and much work remains to be done, particularly in evaluating the strategies developed in this thesis on real-world domains. Additionally, in tackling the goal re-activation problem, we discovered that it is in fact an instance of a larger problem, that of retrieving the right knowledge at the right time, when the agent may not have the correct search knowledge, in terms of both the when to search and what to search for. For goal re-activation, the agent must retrieve the goal (which is a piece of knowledge) when the goal is pursuable; at the same time, without the goal in working memory, the agent cannot know the right time to retrieve it. The strategies from this thesis serve as a good starting point for future cognitive architecture research on how to have the right knowledge available at the right time.

1.3 Contributions

The major contributions of this thesis are:

- This thesis defines the goal re-activation problem for cognitive architectures, and a computational analysis of its source. Namely, that the current memory mechanisms of cognitive architectures lead to a circular dependency problem if the agent lacks memory search knowledge. We demonstrate this with goals, but the same problem exists for general knowledge, and may also exist for agents with only remote access to a knowledge base. As such, any developer of real-time, knowledge-rich agents may be interested in this result, as would cognitive modelers of human memory phenomena.

- This thesis generalizes existing, and develops new, memory strategies for goal re-activation. Although solutions for circular knowledge dependencies exist, this thesis is the first to adapt them to the use of knowledge bases by artificial agents. Broadly, they correspond to the polling and interruption mechanisms seen in computer systems, the latter of which does not current exist for architectural memories.

- This thesis characterizes the benefits of spontaneous retrieval. While spontaneous retrieval, which roughly corresponds to interrupts, has been previously implemented in a subset of cognitive architectures, it has not received widespread use due to its inherent unpredictability. Using the goal re-activation problem as an example, we show that spontaneous retrieval can be beneficial when the agent does not know when to search memory, or does not know how to perform such a search. Aside from the direct applications to architecture developers and cognitive modelers, this characterization may be of interest to the developers of agents with

partially observable internal state, as it may violate the assumption that it is sufficient for the agent to control which internal state to observe.

- Implementation and evaluation of strategies with respect to environmental and task properties. Although they adapt existing solutions, the implementation details of the adaption to long-term memories — particularly the meaning of an interrupt — is non-trivial. This thesis provides such details for future developers, as well as an evaluation of the strategies and the environmental characteristics in which the strategies perform well.

CHAPTER 2

Background

This chapter describes the precise problem that goal re-activation problem poses for cognitive architectures. Section 2.1 breaks down the life-cycle of a goal, a description that is surprisingly consistent between the agent-design community and the psychological study of human prospective memory. Section 2.2 then takes a slight detour by describing the Soar cognitive architecture, in which the work in this thesis is done, before integrating it with the goal re-activation problem in Section 2.3 to examine why it is a problem for cognitive architectures. Since Soar is found to be inadequate at goal re-activation, we discuss the criteria for modifying Soar in Section 2.4. Finally, Section 2.5 looks at existing work in goal management, as well as relevant ideas from outside artificial intelligence and psychology.

2.1 The Goal Re-Activation Problem

This section more precisely defines the goal re-activation problem by reviewing the existing literature on goals in both artificial intelligence and psychology. The difference between the goal re-activation problem and other related problems is also described.

2.1.1 The Life-Cycle of Goals

One of the basic distinctions between different types of agents is whether the agent is goal oriented [51]. As a result, a theoretical understanding of goals and their management has been of some interest to the artificial intelligence community. Simultaneously, the human ability to pursue goals after some delay has also led to the growing field of prospective memory in psychology [40], although the phenomenon has no concise definition short of a "fuzzy set" of intuitions around "remembering to *do something* at a particular *moment (or time period) in the future*" (emphasis in original) [41]. We draw from both fields to aid in understanding the problem.

First, we must acknowledge that arriving at a standard definition of goals is difficult, as it must accommodate the representations and processes of different agent architectures, in addition

to providing a description of the types of activities that researchers would like agents to perform. This difficulty is also recognized elsewhere [6]. Since this thesis is primarily concerned with the management of goals, and not with their fulfillment, we keep our definition of goals general by under-specifying its representation. The goal may be associated with a sequence of actions that must be performed, or may describe some desired state of the world independent of specific actions; this representation has no effect on this thesis. Instead, we require that the goal contain a *target*, which describes the conditions under which the goal can be pursued. We use "pursue" here in the general sense. One possible target for the message delivery agent is being in the same room as the recipient, in which case the target represents a precondition that must be met before the agent can take action. Alternately, the target could also be an unexpected opportunity that can lead to the completion of the goal if taken advantage of (for example, unexpected learning the telephone number of the recipient). In this case, the agent may not have planned for receiving the recipient's phone number, but this knowledge may cause the agent to replan and thus make the goal relevant. The target therefore represents a set of features which, when present, indicate that the agent should reason about a goal.

To use the first scenario from Section 1.1 as a concrete example, the goal of delivering a message may have multiple targets. One target may be being in the same room as the recipient, but another may be learning the recipient's telephone number, which would now allow a different action for achieving the goal (that of calling them instead of meeting them physically). Note that the two targets are different in nature: the first is a precondition for an existing plan, the other is a criterion for replanning. These targets are not mutually exclusive — multiple targets could exist for one goal, each of which could require the agent act different when satisfied. The only common element between targets is that they require the agent to deliberately consider the goal, whether that means the agent should take action or should re-plan.

Under this definition, both the psychology and artificial intelligence literatures have proposed similar models of the life-cycle of a goal [18, 49]:

Encoding In this stage, the agent is either given the goal externally or generates it from some internal process. Whatever the source of the goal, it must be stored in the agent's memory. In the message-delivery example, the goal to deliver the message is conveyed to the robot.

Retention Between the creation of the goal and when its target is satisfied and the goal can be pursued, the goal is retained in the agent's memory, where it is subject to the dynamics of the memory system. During the *retention interval*, the agent is working on other immediately-pursuable tasks (*foreground tasks*), and its computational resources are allocated for that purpose. In the running example, this stage corresponds to the time while the recipient of the message is not present, the goal is maintained in memory and the agent is otherwise occupied.

Initiation The initiation stage is the crux of the goal re-activation problem, and this stage begins when the target is satisfied. There needs to be a careful distinction, however, between whether the target of a goal is *objectively* satisfied in a situation, and whether the agent recognizes that the target is satisfied in the situation (that it is *subjectively* satisfied). To use the message-delivery example again, the recipient could be in the same room as the agent, meaning that the target is objectively satisfied. For a bounded agent, however, it is possible that it fails to perceive the recipient, or alternately fails to recognize that it has a goal of delivering the message. In either case, the agent would not recognize the goal as pursuable in that situation. It is the recognition that the target of a goal is subjectively satisfied, *not* objectively satisfied, that is the goal for the initiation stage.

Performance Once the agent recognizes that the target of a goal is satisfied, the agent may choose to pursue the goal. In the example, the agent catches the attention of the recipient and gives them the message.

Completion After the goal is achieved, the agent must update its memory so that it will not pursue the goal in the future. As a concrete example, the message-delivery agent should not try to get the target's attention (for the purpose of delivering the same message) the next time they are near each other.

Goals that agents have *adopted* are often classified as either *active*, as an *option*, or as *suspended*. Active goals are those currently pursued by the agent; options are goals that the agent could pursue if it so chooses (that is, when its target is subjectively satisfied); and suspended goals are ones whose targets are not satisfied. This thesis is concerned with the transition of a goal from being suspended to being an option, which corresponds to the initiation stage above. While agents could deliberately decide to pursue suspended goals — for example, by deciding to call the recipient of a message instead of waiting until they are nearby — that case is not within the scope of this thesis. However, that use case must be kept in mind, as it implies that goals must remain declaratively available for the agent's goal management reasoning.

2.1.2 Other Concerns

First, the goal re-activation problem is not concerned with the goal being completed, only that it is an option for the agent to pursue. Whether the agent chooses to pursue the goal is well-suited to be tackled by decision theory; that the goal was first suspended requires no special consideration. Since choosing which goal to pursue (or whether to pursue it) is not the topic of interest, this thesis treats all goals as having the same utility, and agents will simply select randomly between multiple prospective goals if they are simultaneously options for pursuit. Additionally, this thesis ignores the

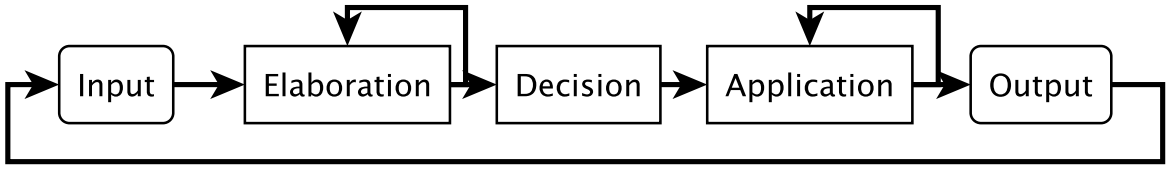

Figure 2.1: The five phases in Soar's decision cycle

foreground task and only considers the amount of computation required for detecting the satisfaction of the targets of prospective goals. It is possible that a particular goal re-activation strategy could be impacted by the foreground task — a strategy may generally perform well but degrade quickly when interrupted — but as a starting point, this thesis focuses only on the cost of the goal re-activation strategy.

The second and third questions of whether the action is optimal and whether the action achieves the goal can be solved with a multitude of other artificial intelligence techniques. To keep the focus on goal re-activation — that is, whether the agent recognizes that the target of a goal is satisfied — the problems in this thesis assume that the agent always completes the goal.

It should be noted that these are not assumptions that restrict the scope of the thesis, but considerations that are irrelevant to the goal re-activation problem. They are described here to further delineate the boundaries of the problem, and to acknowledge that we selected solutions to these issues in the evaluation; they are not required for the results of the evaluations to apply.

2.2 The Soar Cognitive Architecture

Since we implemented the work of this thesis in the Soar cognitive architecture, this section describes the mechanisms and processes of the architecture. The Soar cognitive architecture was chosen because its structure is representative of many other cognitive architectures, and is additionally well-documented for use and for architectural experimentation.

2.2.1 Overview

Soar is a cognitive architecture focused on knowledge-rich agents that operate in real-time. It has been successfully deployed not just on robots, but also as intelligent opponents in computer games. The architecture was first developed as a realization of the problem space computational model, which describes how agents represent knowledge, propose different ways of processing that data, and decide which of these methods to perform. A *decision cycle* in Soar therefore comprises five

phases (also shown in Figure 2.1):

Input The architecture updates the agent's state with perception from the environment.

Elaboration The agent applies knowledge to enrich its understanding of its current state. This knowledge includes the proposal of multiple actions, which could be both internal (e.g. a knowledge retrieval) or external (e.g. moving an actuator)

Decision The agent selects one of the actions, taking into account its goals, the environmental state, and other knowledge.

Application The agent executes the selected internal or external action.

Output The architecture communicates any actuator movement to the control system.

These phases are all involve changes to Soar's *working memory* — a rooted, edge-labeled directed connected graph that represents Soar's current state. This representation is explicitly relational — the edge labels describe relationships between entities, which are represented by the nodes. A <node, edge, node> tuple in working memory is called a *working memory element*. Special regions of this graph are *buffers*, and serve as communication channels for other modules of the agent. For example, perception from the environment arrives through the *input link*, while actuator movement is created in the *output link*. Additionally, there are also buffers to the *long-term memory* of Soar, which are described in Section 2.2.2. An overview of the Soar architecture is depicted in Figure 2.2. Although multiple symbolic long-term memories are shown in the figure, this thesis is only concerned with semantic memory as a declarative long-term memory. All usage of "long-term memory" therefore refers to semantic memory.

Elements are added to and removed from working memory through *if-then rules* that are stored in *procedural memory*; they constitute the *procedural knowledge* of the agent. Unlike the graph structure of working memory, the agent does not have declarative access to rules, nor can it directly modify or remove them. The *condition* and *action* parts of a rule both describe graph structures. A rule *applies* or *fires* if the conditions of the rule matches the structures in working memory, at which point working memory is modified according to the actions of the rule. Rules also propose the different actions (or in Soar terms, *operators*) that the agent can take, while other rules create *preferences* for these operators so that a single best operator is selected to be pursued; the results of these operators are considered the *deliberate* actions of the agent. Since working memory is a graph, matching the conditions of a rule to the structures of working memory requires solving a subgraph isomorphism problem, which is known to be NP-complete. Soar therefore uses the Rete algorithm [22] to help mitigate some of these costs in the typical case, although it remains the case that rule matching can take time exponential to the size of working memory.

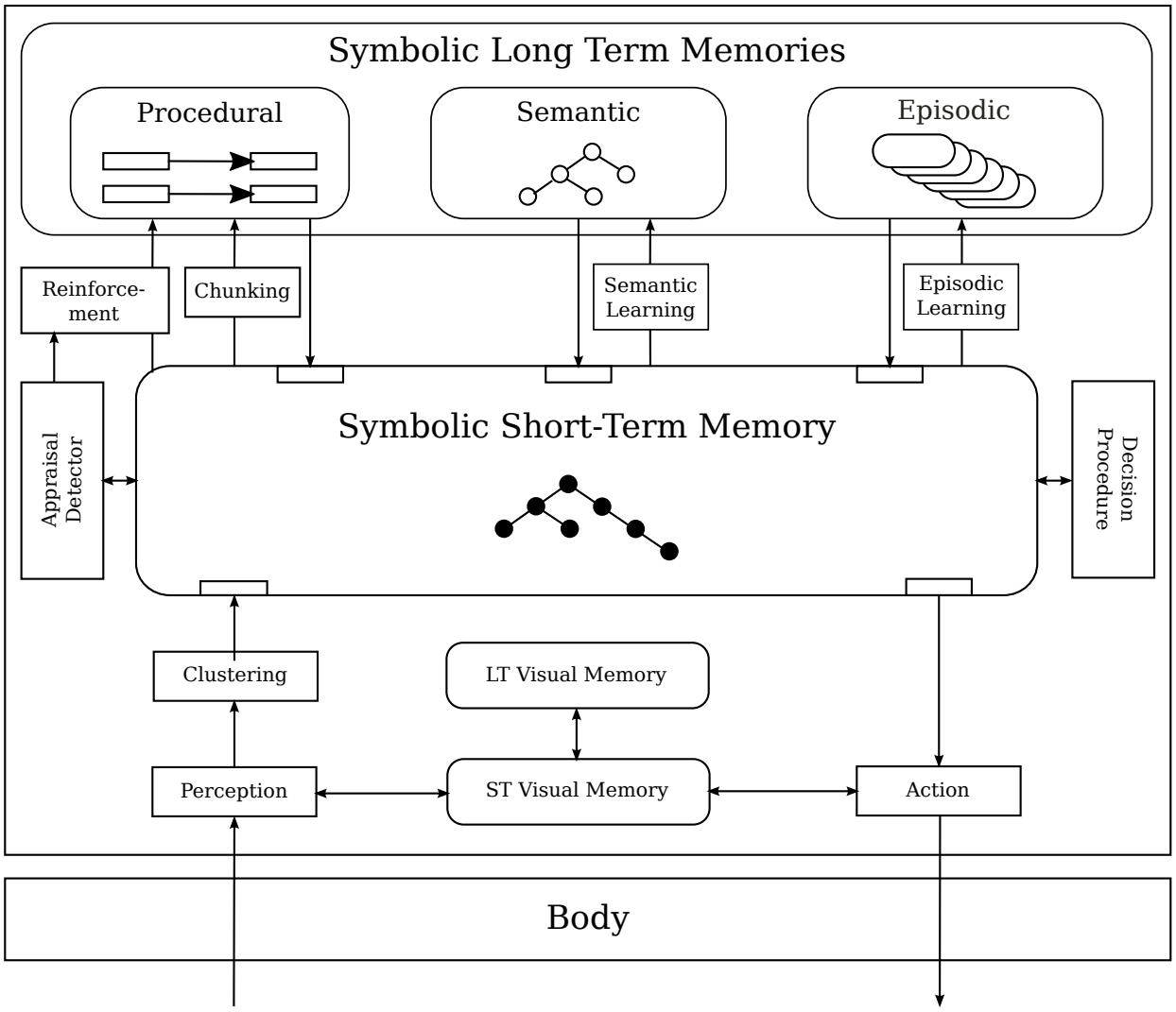

Figure 2.2: The Soar cognitive architecture

As an architecture to be used in real-time agents, Soar needs to be reactive to changes in the real world. This is often translated into a requirement that a decision cycle takes at most 50 milliseconds, a threshold that has been found to be sufficient in human tasks [9]. This requirement, however, can be in conflict with the goal of agent competency, which in general requires larges amount of knowledge. Storing all knowledge in working memory drives up rule-matching costs, which in turn would reduce the reactivity of the agent. In general, the paradoxical reduction of agent performance despite (or rather, because of) the addition of knowledge is known as the *utility problem* [44].

In Soar, a heuristic solution to this problem is *forgetting*, the automatic removal of knowledge from working memory. Associated with each working memory element is a number, called the element's *activation value* or simply *activation*. The activation of a working memory element increases (is *boosted*) when it is tested by the conditions of a rule that matches; semantically, this means that the piece of knowledge was used in the agent's decision process, and is therefore more likely to be useful in the future. Over time, if the element is not used in rule matching, its activation decreases or *decays* logarithmically to signify its falling importance; the *decay rate* is an agent parameter often tuned to the particular domain. If the activation drops below a *forgetting threshold*, the element is removed from working memory. An example trajectory of two working elements is shown in Figure 2.3, which shows the activation of the elements on the y-axis and time on the x-axis. One of the elements (designated by a 'X') is tested by the conditions in several rules that fire, leading to the discontinuous positive jumps in its activation; the other element (designated by a '+') is only tested once by a single rule that fires, and so its activation decays over time, until it passes below the forgetting threshold. This model of activation boosting and decay is known as *base-level activation*, which comes from the cognitive modeling literature and has been found to predict the likelihood that features in the real-world reoccur [4]. Base-level activation is mathematically described by the equation shown in Figure 2.4, where n is the number of activation boosts, t_i is the time since the i^{th} boost occurred, and d is a free decay-rate parameter.

The inclusion of forgetting means that it is possible for the agent to lose knowledge. To prevent this, forgetting only applies to knowledge can be recovered from long-term memory, which is described in the next section.

2.2.2 Long-Term Memory Mechanisms

Soar's long-term memory serves several purposes. First, philosophically, working memory is only for knowledge that is immediately relevant to the agent, and not for other general knowledge that the agent may need in other situations; long-term memory is the architectural answer to the question of where such knowledge would be stored. Second, as mentioned in Section 2.2.1, the capacity of working memory must be balanced against the speed of rule matching, and the introduction of a

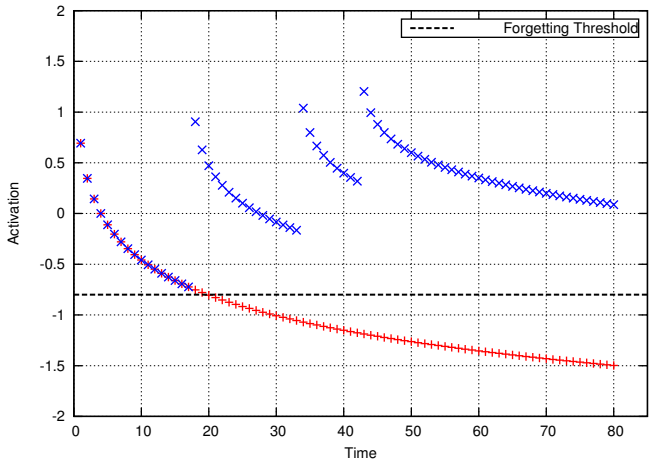

Figure 2.3: An example base-level activation time plot

$$activation = \ln(\sum_{i=1}^{n} t_i^{-d})$$

Figure 2.4: The base-level activation equation

forgetting mechanism in working memory suggests that the ability to recover forgotten knowledge is desirable. Long-term memory therefore serves as a "backup" repository of knowledge, from which the agent can retrieve knowledge into working memory at some point in the future.

Finally, long-term memory is a mechanism to allow more flexible indexing of knowledge. Like working memory, long-term memory represents knowledge as an edge-labeled directed graph; this shared representation allows seamless transfer of knowledge between the two memories. Long-term memory currently has two sources of knowledge. First, knowledge can be pre-loaded into an agent; this often occurs with large knowledge bases such as WordNet [43] or DBpedia [5]. Alternately, the agent can deliberately store knowledge via the long-term memory buffer in working memory. The same buffer is also used for *retrieving* knowledge from long-term memory. To do this, the agent creates a *cue* — a description of the features in the desired piece of knowledge. Long-term memory then selects one element that matches this cue, and recreates it in working memory, allowing the agent to reason with this knowledge, incorporating it into its decision making process. The retrieved knowledge can also be modified while in working memory, and then stored into long-term memory again; by this process, the contents of long-term memory can be updated. Although deleting long-term memory elements is not technically possible, the same effect can be achieved by modifying an element such that it has no features; in essence, this prevents it from being retrieved, and could therefore have no further effect on the behavior of the agent.

Similar to working memory, knowledge in long-term memory has associated activation. Every

long-term memory element has an activation level that follows the base-level activation equation. Whereas in working memory the activation is boosted by rule matches, the activation of long-term memory elements is boosted whenever it is stored or retrieved, since these are the events that indicate that an element is useful. Unlike working memory, however, activation is not used to remove knowledge from long-term memory; rather, it is used as a *bias* for retrieval. For example, consider a retrieval in which the agent-created cue matches multiple long-term memory elements. In this situation, which element should be retrieved? To break the tie, long-term memory selects the most highly activated element, which can be intuitively understood as retrieving the element with the highest prior probability of being useful. The connection between base-level activation and a Bayesian interpretation of memory retrieval is beyond the scope of this thesis, but has always been part of the original rational analysis work on long-term memory [4].

In addition to the bias mechanism, the agent can also iterate through all long-term memory elements matched by the cue by *prohibiting* previously retrieved elements from being retrieved again. That is, after the first element is retrieved, the agent can search long-term memory again with the same cue, but this time with the first result prohibited. Long-term memory then returns the second-highest activated element that matches the cue. The agent can repeat this process as long as necessary, until no new elements are returned, in order to iterate through matching elements; for example, this is a way an agent can retrieve all suspended goals.

2.2.3 Rule Learning

Thus far, the description of Soar has focused on how the architecture manipulates declarative knowledge in working memory and in long-term memory. As the beginning of Section 2.2.1 mentioned, this process is described by rules, which themselves form the procedural knowledge of the agent. Although any agent needs both procedural and declarative knowledge, these two types of knowledge are considered distinct, and the architecture treats them differently. The most important of these differences is that procedural knowledge is not considered part of the data available for the agent for reasoning, and as such, as a fixed, limited set of mechanisms for access. The matching of rule conditions with working memory elements occurs automatically, and this match must be complete — satisfying a subset of the conditions of a rule is not sufficient to cause the actions to apply. Furthermore, since rules are not considered knowledge for reasoning, Soar has no declarative access to the conditions or actions of rules, nor to reason over the structure of those rules; this also means that the agent cannot delete rules, since it cannot specify which rule to remove. In contrast, long-term memory allows the agent to search for knowledge using only a subset of its features, and also allows the agent to freely create new knowledge by linking and re-combining existing knowledge. These restrictions on procedural knowledge exist primarily because rule matching is one

of the inner loops in the architecture, and as such must be efficiently computable; the Rete algorithm achieves its speed partially by exploiting these constraints. In turn, the long-term memories were developed to allow a more flexible method of knowledge access that rules could not provide.

While Soar agents do not have declarative access to its rules, there does exist a single mechanism, *chunking*, for the agent to learn new rules. Chunking, also known as *production compilation* in ACT-R, is a learning mechanism that learns new rules by combining existing rules, thus eliminating the need for those rules to match and apply individually. Despite this mechanism, however, there are several disadvantages to using procedural memory as a store for declarative knowledge. First, if knowledge is stored in rules, the only mechanism for retrieving that knowledge is for the rule to match. Thus, if a piece of knowledge is needed in multiple contexts, the agent must create a rule for each of these contexts. Not only is this process inefficient, but it also requires that the agent be able to predict all the contexts when a piece of knowledge is needed at the time rules are learned. This is a difficult task, if not impossible for many domains. Second and more problematic is that agents do not have declarative access to the knowledge in rules. This means that unless a rule fires, the agent has no way to reason about the stored knowledge — nor does it have a way to create the conditions under which the rule would apply. The lack of declarative access also implies that once learned, rules cannot be modified or removed, as the agent has no mechanism to refer to a particular rule to be deleted, nor to indicate which part of it should be modified. Together, this means that chunking should only be used for knowledge that is unlikely to change over time, and whose use is highly predictable. All other knowledge is better stored in long-term memory.

2.3 Goal Re-activation in Cognitive Architectures

The memory mechanisms discussed above may not be the most suitable for goal re-activation, as they were designed for more generally uses of memory. This section addresses two concerns. First, are these mechanisms specific to Soar? Second, how might the memory mechanisms described be used for goal re-activation? We first point out the similarities between Soar and other agent architectures, focusing particularly on those that impact goal re-activation, before considering their application in goal re-activation.

2.3.1 Declarative Memory Hierarchies

The relationship between working memory and long-term memory can be generalized into a *memory hierarchy*, an organization of knowledge that is shared by many architectures, including ACT-R, Clarion, LIDA, and others. Much like their analogues in computer architectures, the memories in the hierarchy are arranged by the stability, the influence on behavior, and the amount of knowledge

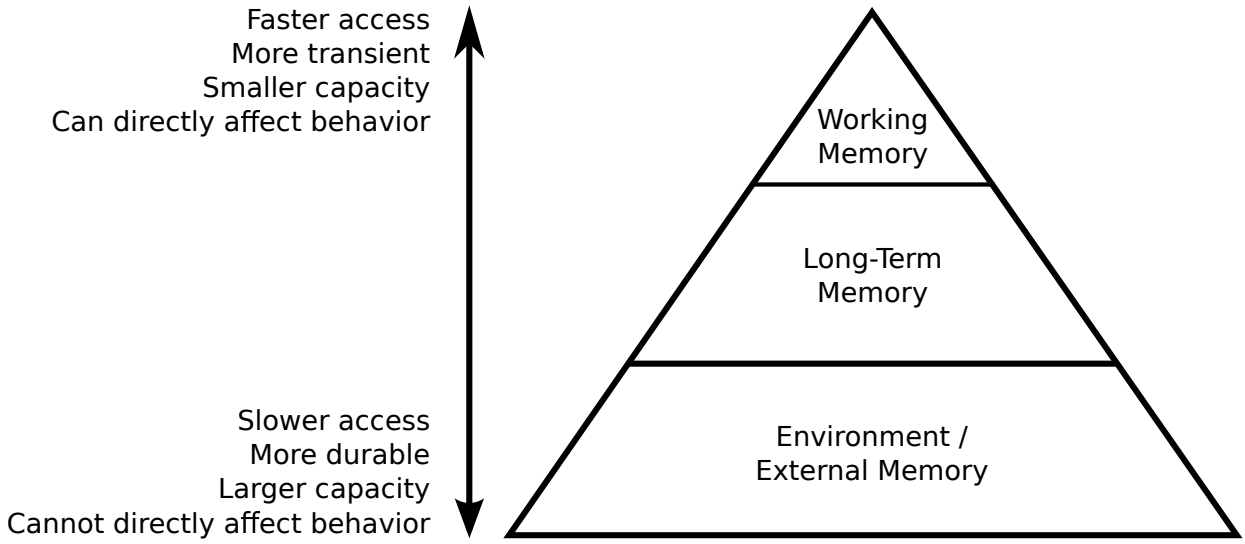

Figure 2.5: The agent memory hierarchy

contained. A depiction of this hierarchy is shown in Figure 2.5.

At the top of the hierarchy is short-term or working memory. This memory contains the immediate perceptions of the agent and knowledge that are relevant to the current situation. The knowledge in short-term memory directly determines the behavior of the agent, as it is the only knowledge against which procedural knowledge is matched. Since the size of short-term memory is a major factor in the cost of rule-matching [22] it is usually kept small. For this reason, some architectures place an architectural limit on the size of working memory (as in ACT-R), while others have an architectural process that removes memory elements over time (as in Soar). Both of these mechanisms limits the data that is available to direct behavior, and may be considered forms of forgetting.

At the next level of the memory hierarchy is one or more long-term memories. This level of the hierarchy contains knowledge that may be useful to the agent over its lifetime, but not necessarily at the present; examples include facts about the domain and the agent's previous experiences. Due to its potential size, procedural knowledge cannot directly be conditioned on knowledge in long-term memory. Instead, long-term knowledge is accessed through deliberate cued-retrievals, where the results are deposited in specialized buffers in working memory. Although knowledge in long-term memory does not directly impact rule-matching costs, it may still be forgotten (depending on the architecture); whether a particular memory element is lost is often a function of the agent's previous access to that knowledge.

The last level in the "memory" hierarchy is the environment, which is included for completeness. It is considered a level in the hierarchy because knowledge that is lost from long-term memory may be recoverable from the environment. Since the environment is external to the agent, access to

knowledge is extremely slow as compared to other memories.

Although not all cognitive architectures implement the entire hierarchy, this thesis mainly focuses on the interaction between the first two layers, on working memory and long-term memory, which many architectures do contain. In particular, the architecture must have a working memory of limited size (e.g. CHREST, Companions, Polyscheme) or that is subject to forgetting (e.g. CLARION, LIDA), and must support directed memory retrievals (all of the above) [10, 20, 23, 56, 57]; a specific example of the memory hierarchy of ACT-R [3] is given in Appendix A. As described in Chapter 3, these features are sufficient for goal re-activation to be problematic, and for the analysis and solutions of this thesis to apply.

Agents that contain this memory hierarchy present additional challenges for goal re-activation due to the assumption that goals cannot be maintained in working memory indefinitely. The problem is that, for any goal with a sufficiently lengthy retention interval, the goal is removed from working memory, only existing in long-term memory. Since the contents of long-term memory cannot directly affect behavior, this means that even if the target is objectively satisfied, the agent would not recognize that the goal is an option for pursuit. In cognitive architectures, where procedural knowledge automatically matches against the contents of working memory, the detection of goal target satisfaction is automatic, as long as the description of the target is also in working memory. Thus, the crux of goal re-activation for cognitive architectures is the retrieval of the goal from long-term memory at the right time. This is the retrieval problem of goal re-activation.

It should be noted that the cognitive architecture community has thus far not addressed the goal re-activation problem. Most research in the field assumes that the agent can be designed with knowledge of when exactly to retrieve its goals, as well as how to do so. This may be true of the domains in which cognitive architectures have been applied, where the targets of goals are known at the time of agent design and their satisfaction could be predicted. This is, however, not true more generally, which suggests that the community has yet to consider domains where the goal targets are more varied and where the environment may be less predictable. As a result, while models of human prospective memory exist, they have not tackled the processes in the initiation stage, which is the crux of the goal re-activation problem.

2.3.2 Procedural Memory

There is one type of long-term memory that the previous section did not address: procedural memory. This section considers the type of memory mechanisms that are necessary to support the storage and retrieval of suspended goals, and demonstrates why procedural memory is ill-suited for goal re-activation.

Consider the case where the agent unexpectedly finds itself with some idle time, and decides

to organize its goals and perhaps pick another goal to pursue. A simple procedure for doing so may involve the agent iterating over its goals, then for each goal determine whether it is still achievable (and delete it if not), or if details about the goals need to be updated (if, for example, a message-passing agent has learned of a new way to contact the recipient). The agent could then use the content of the goals to prioritize them, before selecting a particular goal for pursuit. A more sophisticated agent may take the environment into account to do a more targeted search for applicable goals; for example, if the agent is in a part of the environment that it does not often visit, it may search for goals that need to be performed in the area, to avoid the need to return in the future.

This simple vignette hints at how an agent interacts with its goals, and therefore the memory mechanisms that are necessary to support these interactions. Although not all of the processes that the agent goes through in this vignette are addressed in this thesis, as mentioned in Section 2.1.2, they must nonetheless be taken into account in a complete goal management system. Broadly speaking, this example illustrates the need for goals to be *declarative*, in that the agent must reason about its goals in a number of ways. First, the agent must be capable of retrieving and iterating through its goals, in order to consider all the possible goals it could pursue. Furthermore, this consideration process requires the goal to be represented declaratively, since it is the content of the goals that allows the agent to prioritize different goals. While simple iteration and filtering them may be sufficient, a more efficient method of finding pursuable goals is to directly search for goals with specific features, such as the location in which the goal should be performed. In addition to these access requirements, the example also suggests that goals need to be modifiable, not only so that they can be marked as completed, but also because alternate methods for their completion may arise, or their specification may be changed after they are initially created. Finally, if a goal becomes unachievable, the agent may decide to remove it from memory entirely, which suggests that the deletion of goals is also necessary.

We can now consider whether different memory mechanisms in cognitive architectures meet these requirements. We first consider procedural memory, that is, the use of rules. At a glance, the non-declarative nature of procedural memory means that it likely does not meet many of these requirements. For completeness, however, we can examine how procedural memory fails specifically. First, once a rule is learned and the goal is forgotten from working memory, the agent has no way of retrieving the goal back into working memory without the rule firing. This means that the agent cannot iterate over goals that are stored in procedural memory, never mind performing any kind of search. In fact, the agent cannot determine whether it has any goals at all, outside of when a rule matches the goal target, since unless a rule that represents that goal fires, the agent would have no method of interacting with the goal. Even if the agent could create the conditions for a rule representing a goal to match, other problems remain. In the scenario where the goal must be

updated, the agent has no way of modifying the original rule that had brought the goal into working memory. Similarly, the agent cannot simply replace the rule with a new one, procedural memory does not provide a mechanism for deleting rules. While new rules can be learned, the now-obsolete goal would remain in memory for the remainder of the agent's lifetime. To be more specific, both replacing and deleting rules would require a mechanism for the agent to specify which rule to replace or delete. No such mechanism currently exist in cognitive architectures, and its development would also require significant changes to the architecture. In summary then, the limited access methods to rules in procedural memory means that it is not a strong candidate for supporting the storage and retrieval of goals.

Declarative long-term memory, in contrast, was designed to overcome these problems with procedural memory, and is a more suitable mechanism for goal management. The deliberate retrieval mechanism of long-term memory explicitly allows for filtering goals by their substructure, thus allowing the agent to efficiently narrow the number of goals it has to consider. The same mechanism allows the agent to easily iterate through its goals by not specifying any substructure at all. Once a goal is retrieved, the agent could modify the goal in working memory, then use the storage mechanism to update the contents of that goal in long-term memory; alternately, the agent can delete the goal if it is unachievable or no longer relevant. Long-term memory therefore already supports the various manipulations of goals that an agent requires, and is a natural candidate for further exploration in goal re-activation.

Although procedural memory only offers a limited access mechanism and does not do well in this comparison, it does not mean that it has no role in goal management. The automatic, architectural matching of rules to the contents of working memory means that the agent can use rules to detect specific targets. While the inability to delete rules means this cannot be directly used for one-time goals (assumption A1), it *is* well suited for habitual behaviors that should occur repeatedly, such as checking for cars before crossing the street. This is, however, not the focus of this thesis, and is not examined further.

2.4 Criteria for Architectural Modifications

The previous section described the state of Soar and of other cognitive architectures before this thesis. In the course of this work, however, we found that the existing mechanisms of Soar were insufficient to adequately solve the goal re-activation problem, and that modifying the architecture was necessary. Specifically, the mechanism of *spontaneous retrieval* was developed; this mechanism is described in detail in Chapter 5. This section justifies this particular choice of modification.

First, it is important to understand the space of architectural modifications. The development methodology of the cognitive architecture community is outside the scope of this thesis, but in

general the constraints of an architecture can be classified as either *core* or *peripheral* hypotheses [12]. Core hypotheses, known as *assumptions* in this thesis, are constraints that researchers do not consider for removal or relaxation in the development of an architecture. Peripheral hypotheses, here *conventions*, on the other hand, are constraints that the community has adapted, but whose epistemic status is still subject to change. Thus, if researchers find tasks at which the architecture cannot perform adequately under the current set of assumptions and conventions, the conventions may be relaxed or removed. As other conventions hold over time and over multitude of tasks, they begin to be regarded as assumptions by the community.

This thesis concerns the architectural assumptions and conventions A5 to A11 as listed in Section 1.1. A5, A9, and A11 are conventions, while the remaining four (A6, A7, A8, and A10) are core assumptions. We list them again below for convenience:

A5 Suspended goals are represented symbolically and declaratively, and are subject to the remaining assumptions describing the processing of declarative knowledge by the architecture.

A9 The only mechanism for long-term memory retrieval is deliberate cued retrieval.

A11 The agent cannot deliberately modify or delete rules.

Although the relaxation or removal of any one of the three conventions may allow the Soar architecture to adequately solve the goal re-activation problem, we have chosen to focus on the removal of A9. There are four major reasons for this choice:

- *Removing A9 takes the conservative approach to architectural development.* The analysis from Section 2.3.2 suggests that long-term memory is a natural mechanism for the storage of goals, and it is therefore also natural to first consider how the system could be extended. While Soar agents can only access long-term memory through deliberate retrievals, researchers have created alternate retrieval mechanisms in other architectures. In both ACT-R and Companions, the architecture automatically provides agents with contextually relevant knowledge from long-term memory, without agent deliberation. Although this mechanism is rarely used in practice, the community recognizes that there is little cost to the mechanism, even if its benefits are thus far unknown. In contrast, no cognitive architecture has explored the consequences of allowing agents to have deliberate access to its rules, and such a modification is likely to have an impact beyond the scope of this thesis. Similarly, the special treatment of suspended goals in long-term memory has not been explored. Although the consequences of removing A5 are likely to be smaller than that of removing A11, there are other reasons for selecting A9 over it; see below.

- *Removing A9 provides a general solution.* Because cognitive architectures are tools for creating agents, and are not themselves agents created for specific tasks, they need to address the more general problem of knowledge retrieval from long-term memory when the agent does not know that such a retrieval is needed; this is an obvious extension of the goal re-activation problem. Since this larger problem is not limited to goals, removing A5 would not provide a satisfactory general solution; this is further discussed in Section 3.1 As for the removal of A11, because it is a generally unexplored mechanism, it is unclear how it would support general knowledge retrieval, and furthermore deviates from the current architectural processing of semantic knowledge.

- *Removing A9 avoids incurring ongoing costs.* One of the key motivations behind the development of long-term memory is so that working memory does not grow too large, as it could otherwise lead to decreased reactivity. Removing A5 would mean that prospective goals are stored in working memory, possibly indefinitely, leading to a growing working memory over the agent's lifetime if the goals are never achieved. This in turn would increase the time necessary for rule matching, and therefore permanently decrease the reactivity of the agent.

- *Removing A9 allows this thesis to contribute to the cognitive modeling community.* Although this thesis explores a topic in computer science and artificial intelligence, it should not be forgotten that the study of cognitive architectures came out of the need to integrate disparate psychological and cognitive science theories. The description of architectural processes in Section 2.2 also apply to other architectures, including ACT-R (see Section 2.3), the main research goal of which is to model human behavioral data. As human psychological data shows that people are not limited to deliberate memory retrievals, discarding A9 would allow researchers to tackle a larger set of psychological phenomenon. The same cannot be said of the two other conventions. Since rules correspond to ingrained behavior in people (either through biology or thorough habit), removal of A11 would be the equivalent of humans having the ability to modify and remove automatic responses at will, something that we cannot do. Similarly, it is clear that humans do forget goals, much as we forget other knowledge, a fact not reflected by the removal of A5. Selecting A9 for relaxation therefore broadens the contribution of this thesis.

2.5 Related Work

This section looks at existing work on goal management in both cognitive architectures and non-cognitive agents. We also discuss relevant ideas from computer science outside of artificial intelligence.

2.5.1 Other Cognitive Architectures

As described in Section 2.3, goal re-activation is a problem that is shared by a number of cognitive architectures. The majority of prior work related to goal re-activation was done in ACT-R, due to the architecture's widespread use in modeling all aspects of human cognition. These models, however, have not focused on how agents detect that the target of a goal in long-term memory is satisfied; instead, the majority of models assume that the goal target is satisfied, then focus on matching data of human performance on prospective memory tasks. One such study looked at two strategies [17]. A "goal monitoring" strategy first retrieves uncompleted goals, then directs the agent to test specific aspects of the environment to determine if its target is satisfied. In contrast, a "goal cueing" strategy first elaborates on the agent's current perceptions, then relies on spreading activation to bias which goal gets retrieved. In both of these strategies, however, the agent is given knowledge as to when it is appropriate to retrieve a goal. Indeed, the only challenge for the agent is to determine which goal is the correct one to pursue, with the assumption that all the goals are pursuable (that is, that their targets are subjectively satisfied). A different study on the *intention superiority effect*, where uncompleted goals are more easily retrieved than completed goals, has a similar issue [37].

The most complete account of prospective memory in ACT-R looked at how supergoals could be retrieved and resumed after a subgoal has been completed in the Tower of Hanoi puzzle [1]. To ensure that the supergoal remains retrievable from declarative memory, the agent boosts the goal's activation sufficiently before beginning the subgoal. By ensuring that the supergoal is the most activated element in long-term memory, the agent will retrieve it after the supergoal has been completed. However, this solution does not completely solve the goal re-activation problem. In order for the supergoal to be the most activated element, the agent must know the length of the retention interval so it can sufficiently boost the activation of the goal; otherwise, it is possible for some other long-term memory element to have a higher activation and be retrieved instead. This not only requires knowledge of exactly when a goal should be pursued — which is only possible due to the highly structured domain of Tower of Hanoi — but also knowledge of the properties of how base-level activation behaves. The goal re-activation scenarios listed in Section 1.1, in contrast, all have variable retention lengths (A3), and are also relevant during an unrelated task (A4) — in contrast to the strict supergoal and subgoal relationship in Tower of Hanoi. In other words, this strategy assumes relationships between goals and knowledge about goal retention intervals that may not be available in other tasks.

There are other studies related to goal re-activation that do not directly address how the agent detects that a goal is pursuable. One viewpoint on goal re-activation is to frame it as the agent attempting to complete two goals — that is, the agent is multitasking. While most studies on multitasking focus on resource allocation between tasks, and not on detecting whether a task should be pursued [52], it may be possible to consider the detection itself as a task that the agent is pursuing

concurrently with others. The majority of tasks modeled under the multitasking paradigm require different resources sequentially; for example, a task may first use memory to retrieve the goal, then require motor resources to perform an action. Detecting the pursuability of prospective goals, however, constantly requires the same resources: the retrieval of the goal from long-term memory, and the procedural knowledge to compare it with perception. Since the resource demand is constant, any use of memory by other tasks would directly reduce the ability for the agent to recognize prospective goals. Instead of creating other tasks that could impact goal-reactivation performance, in this thesis we instead directly manipulate the resources available for goal relevance detection, in order to study its effects on agent performance.

2.5.2 Non-Cognitive Agent Frameworks

While the representation and pursuit of goals has been the focus of much research, the generic management of suspended goals has not received much attention. Work that does exist tends to assume that agents have full access to suspended goals; that is, agent knowledge is fully observable, unlike the partial observability of long-term memory in cognitive architectures. As a result, while there is some overlap, most strategies do not apply.

In general, the detection of whether a suspended goal should become an option is reduced to an issue of timing. That is, periodically, the agent iterates through all its suspended goals to check their targets; if their targets are now satisfied, then the agent reasons about the newly pursuable goals and decides on a new choice of action. [59] A more sophisticated strategy is to check the goal targets whenever certain environmental percepts are detected [6]. While both strategies are intuitive, they under-represent the space of possible solutions, and neither their performance nor their efficiency has been well explored.

Other work exists which does not depend on using timers to check for goal target satisfaction, but instead creates *monitors* for changes to the relevant world states [62]. Given a model of the world, these systems examine the goal target and how it determines what changes to the current state would result in new plans being possible. Since this work is based in the planning community, it assumes that the agent has a model of the environment, which we do not assume in this thesis (A3). At the same time, using environmental features to trigger goal-related actions avoids the efficiency problems of using a countdown, and we draw inspiration from this approach in Chapter 4.

Other work has drawn on similarities between goal re-activation and re-planning. The Goal-Driven Autonomy framework, for example [30], uses the violation of expectations to direct the search for goals, a form of plan monitoring. For each step of the plan that the agent is currently executing, an expectation of the environment is generated. If the actual perceived state differs from expectation, an attempt is made to explain the discrepancy; it is the explanation that drives

the checking of goals. Since this thesis also applies to agents that do not have a model of the environment for creating expectations (A3), this approach would fail to generate expectations and thus fail to detect that a goal is pursuable.

Goal re-activation has received a minimum of attention in case-based learning and planning. The planning community has looked at opportunistic planning, where the challenge is for agents to exploit unexpected opportunities. Goals and suspended plans in opportunistic planning literature are also treated separately from other knowledge and there is research on how the representation of the suspended plan affects its later recognition [53], an issue this thesis does not tackle. Finally, where details are provided as to how satisfied goal targets are detected, they are reduced to a periodic check of the goal [55, 63].

Perhaps the area closest to the retrieval of knowledge from memory is the control of active sensors in robotics, which must contend with partial observability of the external environmental state, as a parallel to the partial observability of the internal agent state that this thesis addresses. A major difference between these two problems is that for active sensing in robots, there is a clear objective function to maximize: the reduction of uncertainty in the robot's belief state [32], as compared to the ground truth of the environment. This means that while the optimal policy may be unknown, the features that are needed to make the optimal decision is present. In contrast, the feature that suggests that knowledge needs to be retrieved from memory itself depends on the retrieved knowledge — or rather, it requires both the feature and the retrieved knowledge itself to indicate that a retrieval is necessary. Of course, this is not true of all active sensing, but as far as we know, there has not been research on how to use active sensing when its success depends on the result of that sensing.

2.5.3 Non-Artificial-Intelligence Research

Finally, although this thesis is in artificial intelligence, the use of memory as a knowledge base suggests similarities to ideas from both the software and hardware aspects of computer science. Given that the memory hierarchy is inspired by the memory hierarchy in computer architecture, it is not surprising that ideas from computer systems are also relevant. In particular, the retrieval and retention of knowledge in working memory parallel the processes of pre-fetching and caching. In computer architecture, pre-fetching refers to the ability of the architecture to read data that is near data that is requested, thus exploiting the spatial locality of the data. Similarly, between local and remote servers (for example, in the case of an internet browser), links in the current page may be fetched before the user clicks on it, since it is likely that they will navigate to one of those pages subsequently (known as link pre-fetching). Notably, these techniques rely on the underlying structure of the data, that spatially-adjacent data are likely to be accessed in turn, whether the

distance is measured by spins of a disk or by graph distance on the internet. The spontaneous retrieval strategy explored in Chapter 5 uses some of the same idea, by assuming that using the graph distance in long-term memory as a locality measure. Similar to link pre-fetching (but unlike standard memory pre-fetching), however, the calculation of what to pre-fetch, as well as the fetching itself, does not come freely with the fetch of the requested data, and in the case of long-term memory in particular, there may be bottlenecks to the amount of knowledge that could be pre-fetched.

Caching, on the other hand, is the opposite of pre-fetching — the local retention of data after it has been used. One of the key research questions in caching is what old data should be removed to create space for new data. These cache replacement algorithms often use measurements of frequency and recency as features for their decisions, which parallel how base-level activation is correlated with these features. This provides additional support for the design of base-level activation, although the connection with caching does not directly inform solutions to the goal re-activation problem.

A second related concept from computer science is that of polling and interruption when a processing waits for an external device, such as getting data from disk or ensuring that a printer is ready. The commonality with the goal re-activation problem lies in that the processing cannot proceed without additional information, but it is unclear how long it will take for that additional information to arrive. As a result, computer architecture designers have looked at two methods for recognizing that the information has arrived. The first, polling, requires the processing to periodically ask for the status of the external device; the second, interruption, requires the device to signal the process when it is ready. These two techniques roughly correspond to the preemptive strategies (described in Chapter 4) and spontaneous retrieval (described in Chapter 5).

CHAPTER 3

Problem Definition and Solution Overview

This chapter presents the first result of this thesis: a computational characterization of the difficulties of goal re-activation in a cognitive architecture. The development of this characterization is described in Section 3.1, together with several problems related to goal re-activation that exhibit the same characteristic. Section 3.2 then describes how this characterization leads to the strategies that are explored and evaluated in the rest of this thesis. Finally, Section 3.3 relates these strategies to existing work.

3.1 A Computational Definition of Goal Re-Activation

To summarize the problem that this thesis is solving: given the separation of working and long-term memory, and the storage of suspended goals in the latter, how does the agent retrieve goals into working memory when their targets are (objectively) satisfied, such that the goals can be pursued? Without loss of generality, it can be assumed that retrieving the goal also retrieves the target, and that if only the target is retrieved, its associated goal can be easily retrieved after the agent recognizes that the target is satisfied. Since we are primarily concerned with recognizing that a goal is relevant, we equate the retrieval of a goal with the retrieval of the representation of its target (and vice versa).

Since the only mechanism for retrieving knowledge from long-term memory is through a rule creating a retrieval cue, it appears that a rule must be written to retrieve the goal as well. But what should be the conditions for the rule — that is, when should the goal be retrieved? Naively, it should be retrieved when its target is subjectively satisfied. This means the retrieval rule must first compare the stored target of the goal to the agent's current percepts, to ensure that it is satisfied. This comparison, however, itself requires that the target of the goal is in working memory, which is the reason we need the rule in the first place; this implies that any additional rules that attempt to retrieve the target would face a similar problem, each requiring additional rules for retrieval before the comparison, leading to an infinite regress. In fact, the conditions of the retrieval rule require that the actions of that rule to have already occurred; the rule requires itself to have fired before it could

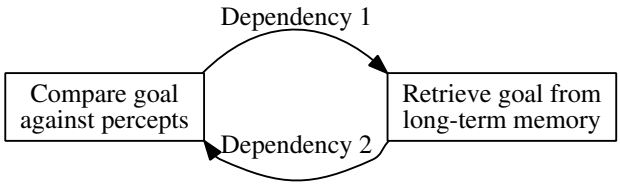

Figure 3.1: The knowledge dependency cycle of goal re-activation

fire.

The fundamental computational difficulty of the retrieval problem of goal re-activation is therefore that the necessary actions have *circular knowledge dependencies* (Figure 3.1). In order to retrieve a goal's target into working memory, the agent must use procedural knowledge. The rules that make up this procedural knowledge, however, must have a condition — in this case, the condition that the target of a goal is satisfied by the current percepts. But for this comparison of target and percepts to occur, the target must be in working memory in the first place. As a concrete example, consider the task of conveying a message, with the target of seeing the recipient. Recognizing that someone is the recipient of a message requires the agent to recall that it has the prospective task of delivering a message (dependency 2 in Figure 3.1). And yet, the agent would not retrieve the goal and its target unless it believes that the task was relevant — such as that the recipient is present (dependency 1 in Figure 3.1). This dependency cycle means that it is impossible to write a rule that only retrieves a goal when its target is subjectively satisfied. Either the goal is already in working memory, rendering moot the point of the retrieval, or the goal is not in working memory, and the rule for retrieval never matches.

Although the core problem of goal re-activation is a dependency cycle for retrieving the goal from long-term memory, a similar dependency cycle exists for perceiving the goal target in a partially observable domain. As a concrete example, consider a scenario where both the message-delivery agent and the recipient of the message are in a crowded room, but are unaware of each other. For the agent to detect the recipient, it must actively search through the crowd; but without knowing that the recipient is present, the agent has no reason to do so. There are less contrived scenarios where this occurs; for example, if the target of a goal is a specific time and the agent must rely on an external clock for the time. The agent could only initiate a sensing action through rules, but the rules themselves would be conditioned on recognizing that the target is (subjectively) satisfied; this is a second knowledge dependency cycle for interacting with external domains. Of course, these two cycles are not mutually exclusive — such is the case for forgetful humans with a time-based target for the goal of attending a meeting, where a clock may not be in sight, and in any case the human has forgotten that they have a meeting, so they don't remember to check the clock.

Another related failure is a lack of knowledge needed by the agent to recognize that the target is

satisfied. Consider an agent with the goal of buying milk, with a reasonable target of being near a store that sells milk. If the agent is standing outside an ethic Irish grocery store, the agent may not recognize the target as satisfied, even if it perceives the grocery store and the goal of buying milk is in working memory. What it is lacking is the knowledge that Ireland leads the world in milk consumption per capita, and the store is therefore likely to carry milk as well. Just as there is circular dependency between retrieving the target and recognizing that it is satisfied, this thought experiment suggests there is a similar circular dependency between retrieving knowledge that is relevant, and recognizing that it would be relevant so that it could be retrieved.

At a glance, this circular dependency leads to an absurd conclusion: if such a dependency exists for all percepts, agent would have trouble recognizing any object. For example, if the agent sees a face, this theory would predict that the agent would not know to retrieve the person to whom the face belongs. There are two mistakes with this extension of the idea. First, there is a difference between recognizing that the percepts form a face, and recognizing that the percepts form the target of a goal. In Soar, automatically recognizing a face may mean that there is additional, sub-symbolic processing involved, with only the symbolic output of that process entering working memory; this parallels existing work on object recognition [45]. This symbol may then be labeled as a face via elaboration rules. The combination of sub-symbolic processing and rules is appropriate for this task, since facial features rarely change over time, since declarative access is not necessary, and since there is rarely an explicit need to delete such knowledge from memory. (How such rules are learned is a topic of ongoing research and is not part of this thesis.) Goal targets, however, do not share these three attributes, which is why rules should not be used (as outlined in Section 2.3.2). Even if goal targets are automatically recognized, there is a second problem: there is a difference between recognizing a set of perceptual features and representing it as a symbol, and retrieving information about that object. In the case of seeing a face, the recognition of it as a face does not automatically identify to whom the face belongs, nor does it retrieve other knowledge about the person, since this information is merely loosely associated with the percept. Similarly, even if a goal target is automatically recognized as a higher-level feature, this does not mean that other information about the goal is retrieved, such as what the goal is or what action needs to be taken. This suggests that recognizing the satisfaction of a goal target is not a specialized capability, but may be an integral part of detecting when knowledge is relevant; this connection is further explored in Chapter 7.

Generalizing from these examples, the goal re-activation, perceptual, and inference circular dependencies are all caused by the partial observability of the agent, either of its goals, of its environment, or of its knowledge store. Note that these issues only arise when the agent cannot simultaneously process the entirety of its environment. For goal re-activation, this corresponds to assumption A4, while for the other circular dependencies it corresponds to the large number of other perceptions and knowledge, respectively. If this was not the case, the agent could always

iterate through all its goals, perceptions, and knowledge, and no dependency cycle would arise.

This analysis shows that the underlying cause of the goal re-activation problem is in fact a broader problem with the mechanisms of long-term memory in cognitive architectures. Although this thesis is focused on goal re-activation, the approaches we develop will attempt to also be applicable to the broader knowledge retrieval problem. The assumptions A1 to A4 also apply to knowledge retrieval. The agent has knowledge that may or may not ever be useful (A1); the agent learns knowledge that is not immediately relevant (A2); the agent does not know when a particular piece of knowledge is useful (A3); and the agent should not devote all resources to knowledge search (A4). While this may mean that the strategies developed may not take advantage of how goals are a specialized type of knowledge, this is in line with convention A5, and does not pose additional constraints.

This thesis assumes that the targets of goals are perceivable without active perception, and that no additional knowledge is needed for the agent to recognize that the goal target is satisfied.

3.2 Strategies for the Retrieval Problem

The knowledge dependency cycle described above is a computational understanding of goal re-activation, caused by the assumptions and conventions of cognitive architectures. This thesis focuses on the relaxation of A9 as a solution to the goal re-activation problem, we also briefly discuss how the other conventions (A5 and A11) could serve as solutions. Relaxing the first convention, A5, suggests the possibility of excluding goals from the forgetting process, allowing goals to be automatically kept in working memory for indefinite periods of time. This would allow rules to directly match on goals without the need for retrieval from long-term memory, thus sidestepping the dependency problem. On the other hand, convention A11 is about the limited interactions the agent has with its rules — in particular, that rules cannot be modified or removed; without this convention, agents could store goals in procedural memory instead of declarative long-term memory, where the rules could directly match the agent's perception of the target. For the reasons described in 2.4, however, these strategies are not pursued in this thesis.

Instead, we focus on relaxing convention A9, which limits the mechanisms through which working memory and long-term memory interact. The dependency cycle remains even without A9, but it provides additional strategies for which to break the cycle. The first strategy — which is already possible — is to retrieve goals without waiting for their targets to be satisfied, with the expectation that they become pursuable before the goals are forgotten; this breaks the first dependency in Figure 3.1. The second strategy — which is only possible by removing A9 — is to somehow compare the target of a goal with the percepts of the agent without first retrieving it from long-term memory; this breaks the second dependency in Figure 3.1.

Perhaps surprisingly, this categorization of strategies can also be applied to the strategies observed in human prospective memory behavior. Within psychology, two major approaches have been described [41]:

- *Monitoring* describes any strategy where attentional resources are periodically expended to check if the targets of goals are satisfied. Since monitoring for the target implies that the goal is already in working memory, this falls under the first class of strategies.

- *Spontaneous retrieval* describes any strategy where the goal simply "pops" into mind; crucially, no attentional resources are used during the retention interval. How the goal is retrieved into working memory is unknown — accounts exist of both retrieving the goal in its entirety, as well as having a vague *feeling of knowing* [31] or some other *metamemory judgment* [46] that the target of a goal is satisfied. In either case, some non-deliberate process is processing the percepts from the environment before the goal is retrieved into working memory; this therefore falls under the second class of strategies.

Traditionally, psychology has distinguished between strategies by whether deliberate cognitive resources are required, as opposed to automatic architectural resources. This is understandable, as procedures for determining cognitive load are well-established. From an architectural standpoint, however, it is more useful to consider the mechanisms required for the strategy to succeed, with a focus on whether the retrieval is deliberate or spontaneous. This mechanism-based classification means that the feeling-of-knowing approach combines the spontaneous metamemory judgment with the deliberate retrieval of the goal. For this reason, the strategy is not fully explored in this thesis, although a plausible mechanism for metamemory judgments is described in Appendix B.

Since it is difficult, if not impossible, to fully enumerate the space of strategy implementations, this thesis explores only a subset of specific implementations. The reasoning behind the choices made in implementing the strategies are explained, but it is possible that alternate implementations of these strategies lead to better performance. The results in the latter chapters should therefore be understood as a characterization of the space explored, and not as a theoretical limit on goal re-activation performance.

Each of the three strategies is briefly described below.

3.2.1 Preemptive Strategies

Instead of using the psychological terminology of *monitoring*, this thesis uses the term *preemptive* to describe a broader category of strategies. In the human strategy of monitoring, people are observed to require attentional resources to check the environment for the targets of goals. These checks

occur either periodically or during context switches [54]; the exact conditions depend on the person and the goal.

Preemptive strategies in artificial agents also incur costs. An agent using a preemptive strategy tries to prevent the forgetting of a goal, either by rehearsing a goal to boost its activation so that it is never forgotten, or retrieving it into working memory periodically to ensure it is almost always there even if it is forgotten temporarily. These actions require deliberate action from the agent during the retention stage, by requiring the agent to perform goal maintenance operations. Since these operations may interrupt the agent's processing of the foreground task — analogous to how the use of attentional resources in human monitoring degrades performance in another task — preemptive strategies are similar to monitoring. Whether the agent rehearses or retrieves goals, these variations on the preemptive strategy ensure that goals are in working memory before the target is satisfied; they are preemptive in that the agent takes deliberate action before the initiation stage, thus removing the first dependency in Figure 3.1. Both of these actions can be performed with Soar's existing mechanisms, and no architectural modification is necessary.

As a concrete example, a message-delivery agent employing a preemptive strategy might prevent the goal from being forgotten from working memory. After the goal is given the agent in the encoding stage, the agent would periodically boost the working memory activation of the goal throughout the retention stage, by deliberating using the goal as a condition in a rule that fires. This prevents the activation of the goal from falling below the forgetting threshold, allowing it to remain in working memory. When the recipient does appear, the agent would detect that the target of the goal has been satisfied, and thus have the option of pursuing the goal. Alternately, instead of preventing the goal from being forgotten, the agent might retrieve it from long-term memory whenever it believes the recipient is likely to appear — such as when the agent is going between meetings. In this case, whenever the agent is transitioning towards situations where large numbers of people are present, it would retrieve the goal of delivering the message, such that the target of that goal is in working memory when the people are in sight, allowing rules to match if the recipient is present. Both variants of this strategy are implemented and evaluated in Chapter 4.

3.2.2 A Spontaneous Retrieval Strategy

The spontaneous retrieval strategy, as the name implies, relies on automatic processes to retrieve goals into working memory. Also known as *remindings* [28], this process is automatic and does not require the use of procedural knowledge. As a concrete example, this is equivalent to the intention of buying milk somehow "popping" into the working memory of the agent as it passes a grocery store. Since no deliberation is required from the agent, this solves the goal re-activation problem by removing the second dependency from Figure 3.1.

Assuming that a spontaneous retrieval mechanism exists, the agent using the spontaneous retrieval strategy only has to ensure that the goal is stored in long-term memory at the encoding stage. In theory, when the target of the goal is (objectively) satisfied, the goal would be automatically retrieved into working memory. Rules would then compare the retrieved goal target to the agent's percepts and determine that it is (subjectively) satisfied, then propose the pursuit of the goal as an option for the agent. Since spontaneous retrieval is a new architectural memory retrieval mechanism, there are a number of open questions regarding how this mechanism works. The implementation and evaluation of the spontaneous retrieval mechanism (by the criteria laid out in Section 2.4), as well as of the spontaneous retrieval strategy for goal re-activation, are detailed in Chapter 5.

3.2.3 A Noticing-Plus-Search Strategy

Noticing-plus-search refers to the strategy that uses a metamemory judgment. In psychology, memory researchers have noted that humans do not only have knowledge about the world, they also have knowledge about their own memory systems; this is known as *metamemory*. For example, one metamemory judgment is whether someone can recall a piece of information at a later date; this *judgment-of-knowing* is used by students to know when they have studied sufficiently for an exam [46]. This phenomenon plays a role in prospective memory when people have a feeling that there is a goal they need to pursue, leading them to perform a deliberate search of their goals. A similar strategy could be applied in cognitive architectures, provided that a source of metamemory judgment is available — such sources would be a new architecture mechanism.

Consider the message-delivery task. After the goal is stored in the agent's long-term memory, at some point the agent runs into the recipient of the message. Unlike spontaneous retrieval, where the goal and its target are automatically retrieved, metamemory judgments return less information. For example, the agent might receive an automatic signal that suggests that long-term memory has additional knowledge about the recipient. Based on this signal, a rule may then search long-term memory for this additional knowledge, and in doing so retrieve the message-delivery goal. A different rule would then propose the pursuit of the goal after the subjective satisfaction of the goal target is detected.

3.2.4 Strategy Summary

The three mechanisms — deliberate retrieval, metamemory judgments, and spontaneous retrieval — can be seen as points in a space of memory mechanisms, which differ in whether agent deliberation is necessary and in the amount of information returned. Both deliberate and spontaneous retrievals result in a piece of knowledge in working memory, while metamemory judgments only result in a piece of metadata *about* a piece of knowledge. The agent must still deliberately search long-term

memory for the source of that metadata before the goal target is retrieved into working memory.

Although noticing-plus-search appears to be a solution to the goal re-activation problem, it was not explored in this thesis. This decision was made due to a preliminary exploration of metamemory judgments, that of recognition [39]. The work is reported in Appendix B, but it was unclear how the agent should select the working memory element with which to search memory. Since metamemory judgments is a novel architectural mechanism — more so than spontaneous retrieval — there is no guidance from prior work as to what judgments would be useful generally or useful for goal re-activation. In addition to this uncertainty, the noticing-plus-search strategies are more complicated than either preemptive and spontaneous retrieval strategies, requiring both an automatic metamemory judgment as well as agent deliberation to retrieve the goal. For these reasons, and because there is prior literature on the mechanisms for preemptive and spontaneous retrieval strategies, they are the focus of this thesis.

3.3 Relation to Other Circular Dependencies

Framing goal re-activation as a problem of circular dependencies allows us to draw solutions from other subfields of computer science. The best-known circular dependency problem comes from how low-level hardware handles input/output, where the processor must wait for a device to be ready, but does not directly have access to the state of the device. Two common solutions to this problem, polling and interrupts, were described in Section 2.5.3.

At first glance, these solutions map roughly onto the solutions presented here. Just as polling requires the processor to periodically probe the device for its state, preemptive strategies require the agent to periodically retrieve goals from long-term memory. There are, however, subtle differences in the use case. First, with input/output from hardware, polling is used to detect a change in the state of the device; with goal re-activation, the state of long-term memory may not change. The simple analogical mapping of long-term memory as the IO device therefore does not hold, as there is no state change that the agent should be notified about. Instead, in order to apply this analogy, we must expand the "device" to include not only long-term memory, but also the external environment of the agent. The agent is then "polling" for when the environment presents percepts that match the target of a goal in long-term memory — and yet, even this expanded definition fails to take into account how long-term memory and the environment can only interact through working memory. There is also no direct parallel for how goals may remain in working memory for some time before being forgotten. Thus, although preemptive strategies resemble polling, there are sufficient additional components in the memory organization of cognitive architectures that warrant exploration of preemptive strategies.

The other form of communication commonly used in IO is the interrupt, where the device sends a

signal to the processor when it is ready. Since the signal originates from the device, this corresponds to both the spontaneous retrieval strategy as well as the noticing-plus-search strategy. Both these strategies require long-term memory to generate some sort of signal — a full piece of knowledge in for spontaneous retrieval, a binary bit indicating noticing for noticing-plus-search. One difference from interrupts, as demonstrated by the two strategies, is that because cognitive architectures are not operating at the hardware level, there is a choice in terms of the amount of information that is transmitted. In general, this may involve a tradeoff between amount of information provided and the cost of acquiring that information in memory. The use of interrupts in long-term memory, however, is a new concept, and poses questions that must be addressed. First, it is unclear what process the interrupt signifies the completion of, and this may itself depend on what information to be contained in the interrupt. Second, since cognitive architectures do not currently have an interrupt mechanism, there is no previous work on how to take advantage of these interrupts, or on how to integrate them with the architecture. Thus, while the concept of interrupts is not new, their implementation details in the memories of cognitive architectures remain a subject of research.

CHAPTER 4

Preemptive Strategies

This chapter presents and evaluates a class of strategies for goal re-activation called *preemptive strategies*, the main unifying feature of which is that they break the dependency cycle in Figure 3.1 by removing the first dependency. Section 4.1 introduces the strategy class and the four variants of the strategy that this thesis explores. Prior work is briefly reviewed in Section 4.2, after which the implementation of the four variants of preemptive strategies are detailed in Section 4.3. As the first strategy to tackle goal re-activation, three different evaluations are necessary; these are laid out in Section 4.4. Section 4.5 explores the upper bound of the number of goals preemptive strategies can manage, while Section 4.6 looks at the strategies in a simulated robot domain. The success of the strategies and the need to perform more controlled experiments drives the development of an abstract prospective memory domain, described in Section 4.7, and the trends in the performance of the strategies in the domain are explained in Section 4.8. Finally, the results from the chapter are summarized in Section 4.9, with takeaways for agent developers looking to use preemptive strategies.

4.1 Overview

Preemptive strategies are strategies that ensure that the goal is in working memory before its target is satisfied; they are *preemptive* in that the agent must take action prior to the satisfaction of the target. To understand preemptive strategies, it is necessary to consider the mechanism that removes goals from working memory: base-level decay of the activation of a goal below the forgetting threshold. This means that for a preemptive strategy to ensure that the goal is in working memory, there are two choices: it must either continually boost the activation of a goal such that it is not forgotten (the *rehearsal* approach), or it must retrieve a forgotten goal from long-term memory before its target is satisfied (the *retrieval* approach). These mechanisms are not mutually exclusive: a strategy could retrieve a goal then boost its activation, such that it would remain in working memory for a longer period of time before it must be retrieved again.

Preemptive strategies can be roughly classified along two dimensions. The first is based on whether they use rehearsal or retrieval for goal maintenance; the former is proactive (to prevent forgetting), while the latter is reactive. Categorizing strategies by this distinction aligns them with the capabilities of the memory systems of cognitive architectures. A proactive approach only applies to Soar's working memory, where the boosting of activation can prevent a goal from being removed. In contrast, ACT-R's working memory is limited in size, and no mechanism exists to prevent a goal from being eventually removed. A reactive approach, on the other hand, applies to the working memories of both Soar and ACT-R, where forgotten knowledge can be recovered from long-term memory.

A second dimension of variation in preemptive strategies is *when* the retrieval or rehearsal occurs; this is called the *trigger* for a preemptive strategy. This thesis explores two such triggers for preemptive strategy, both inspired by human prospective memory behavior. The first trigger is a simple timer: the agent performs preemptive goal maintenance every t time steps, where t is some domain-specific parameter, the ideal value of which depends on both the environment and the agent (specifically, the decay rate of working memory elements). Here, too small a value for t means that the agent would spend more time and computational resources than necessary on maintaining the goal, as would be the case if the activation of a goal is kept far above the forgetting threshold. On the other hand, too large a value of t would mean that the goal is forgotten frequently, and thus is out of working memory for significant periods of time, unavailable for matching against the agent's percepts. At the same time, t should be adjusted based on how frequently the goal target is objectively satisfied; an environment where the target is satisfied often may require a smaller t, since the target may be satisfied even during the short time the goal is in working memory. Although each goal could have a different countdown time t, in this thesis we only look at the case where t is a parameter of the agent and apply to all goals that the agent has. As a concrete example, for the task of passing a message, countdown triggers of five seconds or of two years would both be inappropriate, with the ideal countdown depending on the frequency with which the agent is likely to encounter the recipient of the message.

The second trigger for preemptive strategies relies on perceived features of the environment. For a particular environment, the agent designer determines which features are highly predictive of situations where the goal targets are satisfied, and creates rules conditioned on those features. The action of the rule is to retrieve goals into working memory, such that when the target is objectively satisfied, the agent can detect that that is the case. For example, consider the message-delivery agent in an indoor environment, where the walls of the environment limit the perception of the robot. Since most percepts (including people) only become visible when the robot enters a room, passing through the doorway is highly predictive of new percepts, and therefore of perceiving that the target of a goal is satisfied. A useful strategy for goal re-activation is thus to ensure that goals and their

targets are in working memory as the agent enters a room, so that the targets can be compared to the percepts. In this example, doors are features that indicate a context change, which has been noted to cause the retrieval of goals in humans [54].

Consider the extreme case for a perception trigger, where the agent only has a single goal. In this case, the most "predictive" feature of that goal is the target of the goal itself, and therefore the agent designer could create a rule conditioned on the appearance of the target, that retrieves the goals of the agent (in this case, the single goal). This is different from hard coding the retrieval of the goal into procedural memory, however, in that the action of the rule is not to propose the pursuit of the goal, but to retrieve uncompleted goals. The difference can be seen if the single goal has already been completed; if the goal is hard coded, the agent will propose completing the goal again, while in a preemptive strategy, the agent would search memory for uncompleted goals, fail in that search, and continue with the foreground task. More generally, however, an agent will have many goals with different targets that are unknown in advance, and the agent designer must find percepts that are predictive of the situations in which these targets arise.

In sum, this thesis explores two triggers and two goal maintenance actions for preemptive strategies, each of which can be paired with the other, leading to a total of four preemptive strategy variants:

- the timing-triggered preemptive retrieval strategy
- the perception-triggered preemptive retrieval strategy
- the timing-triggered preemptive rehearsal strategy
- the perception-triggered preemptive rehearsal strategy

4.2 Related Work

As mentioned in Chapter 3, preemptive strategies are inspired by the human prospective memory behavior of *monitoring*. A simple description of this strategy is that people go into a Test-Wait-Test-Exit (TWTE) loop [25], a block diagram for which is shown in Figure 4.1. The person checks their environment to see if the goal target is satisfied; if not, they wait for a while before testing again, until the goal can be pursued, at which point they exit the loop. For example, if the person was tasked with delivering a message, they might employ the TWTE strategy of checking to see if the recipient is nearby every few minutes, until the recipient does appear. Attentional resources are spent whenever the environment is tested for the goal target; in the example, this is when the person checks for the recipient. It should be noted that this use of resources is slightly different from that in the preemptive strategies described here, where it is the maintenance of the goal in memory that

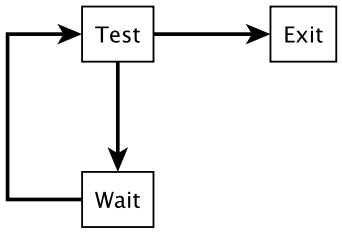

Figure 4.1: The Test-Wait-Test-Exit loop [25]

uses resources, not the checking of the target, which occurs automatically via rule matching. The main characteristic of preemptive strategies, however, remains to be the need for the expenditure of resources during the retention interval [41].

Separate from the monitoring strategy, there is psychological evidence that people tend to recall goals when switching between task contexts, such as when walking down a hallway to a meeting or leaving the house in the morning [54]. Only when a goal is retrieved and an opportunity to act is judged to be imminent does the monitoring begin.

In light of the preemptive strategy classification, it can be seen that previous work with ACT-R also falls under this strategy. A model of "intention monitoring" [17] superficially resembles a preemptive strategy, as goals are first retrieved then tested against the agent's percepts. For the Tower of Hanoi agent mentioned in Section 2.5.1 [1], since the task consists only of super-goal and sub-goals, the generation of a sub-goal is used as the perception trigger for rehearsing the super-goal, before the super-goal is suspended. Simultaneously, the completion of a sub-goal is also used as a perception trigger for retrieving the next goal. For ACT-R, both rehearsal and retrieval are necessary due to the severely limited size of working memory and the possibility of forgetting from long-term memory. That this prior work is an instance of a general preemptive strategy demonstrates the merit of the classification of preemptive strategies.

4.3 Implementation

This section describes the implementation of each variant of preemptive strategies, by describing both timing and perception triggers, followed by both rehearsal and retrieval as goal maintenance actions. In general, both triggers lead to the proposal of the same generic goal maintenance operator. Depending on the settings of the agent, the operator then performs either retrievals or rehearsals.

To implement the timing-trigger, the agent first creates a count of the timing period in working memory. This counter is decremented every decision cycle by whatever operator is selected; this decrement does not require an operator of its own, and therefore does not interfere with other agent

processing. The timing-trigger agent is created with a rule whose condition checks for the counter reaching zero; if this is the case, the rules propose a generic perform-goal-maintenance operator, which is described below. The goal maintenance operator also resets the counter to the timing period, which allows the countdown to begin anew. In contrast, the perception trigger is simpler: it only consists of a single rule that proposes the same goal maintenance operator. The condition of the rule is domain dependent, and its conditions test for whatever features the agent designer believes is predictive of the satisfaction of goal targets.

Although the same goal maintenance operator is proposed by the different triggers, the rules for the operator are specialized to each goal maintenance action, as their conditions check for which action should be taken. For the retrieval action, the rule searches long-term semantic memory using a cue that describes the desired memory element as an uncompleted goal. Additionally, existing goals in working memory are prohibited from being returned. At each decision cycle where the retrieval goal maintenance operator is selected, only a single goal is retrieved from long-term memory. Additionally, even if the retrieval is successful and there could be more unretrieved goals in long-term memory, the operator does not continue in the next decision cycle. While in theory the agent could add the newly retrieved goal to the list of prohibited memory elements and repeat the memory search, thereby asking long-term memory for the next most highly activated goal, in practice this means that the agent could have an unbounded number of retrievals per trigger. Instead, the preemptive retrieval strategy allows only the retrieval of a single goal per trigger, to serve as an upper bound on the number of goal maintenance operations the agent can perform. The limit of a single retrieval is arbitrary; a higher limit trades off reactiveness (from retrieving multiple goals) for the number of goal targets matched in a single decision cycle. Moreover, the bottleneck remains even if more goals are retrieved. As long as there are more goals in long-term memory than can be retrieved (or kept in working memory) simultaneously, this retrieval limit only changes the problem in degree but not in kind.

The rehearsal action also begins with the selection of the generic goal maintenance operator. In this case, however, the operator performs no action if there are no uncompleted goals in working memory. If an uncompleted goal does exist, the rules for the rehearsal action first creates a second counter, which tracks the number of rehearsals yet to be performed. Recall that to boost the activation of a working memory element, the element must be part of the conditions of a rule that fires. For the rehearsal action, this rule is conditioned on both the number of remaining rehearsals (a number that it decrements), as well as on the goal to be rehearsed and its target description. This rule fires repeatedly — within the same decision cycle, since it is considered one application of the same operator — until the rehearsal counter reaches zero. Again, the activation of only one goal is boosted per trigger; as with the number of goals retrieved, this limit does not cause a loss in generality.

Two concrete examples may help with understanding the operations of these preemptive strategies. Consider a message-delivering agent that is using a *perception-triggered* preemptive *retrieval* strategy. When the agent receives the goal to deliver the message, the agent stores the goal in working memory, but no addition action is taken. When the designated feature for the perception trigger is perceived, other goals may be retrieved into working memory. Once retrieved, however, the activations of these goals are allowed to decay over time, until they fall below the forgetting threshold and are automatically removed from working memory. This retrieve-decay-forget cycle may repeat multiple times during the retention stage. Finally, when the target is objectively satisfied, if the goal (and the corresponding target description) is in working memory, a different rule detects that the target is subjectively satisfied, and proposes the pursuit of the goal to the agent. Assuming the agent successfully pursues the goal, the goal is then marked as completed, so that it will not be retrieved again at the next retrieval. The agent thus never considers pursuing the goal again.

To demonstrate the rehearsal action, consider the same message-delivery task performed by an agent using the *timing-triggered* preemptive *rehearsal* strategy. The agent is created with a countdown for (say) 20 decision cycles, together with a rule that matches when the countdown reaches zero. Even before the agent is given the goal to deliver a message, the countdowns still occur and the rule still matches, but the operator performs no action if no uncompleted goals are in working memory. When the timing trigger reaches zero after the agent receives the goal, however, the agent then selects a goal in working memory at random, and boosts its activation by the method stated above. Between triggers, the activation of the goal decays; if the number of rehearsals is insufficient to keep the activation of the goal above the forgetting threshold, the goal is removed from working memory. Since no retrievals are performed as part of this strategy, these forgotten goals will never be retrieved into working memory, and thus will remain uncompleted. On the other hand, if the activation of the goal was sufficiently boosted so that it remains in working memory for the next trigger, the goal may be selected to be boosted again. This decay-boost cycle may repeat, keeping the goal in working memory indefinitely. When the target is objectively satisfied, the same rules as the other agent apply. Once the goal is completed, the goal is marked as completed, and it will not be selected for rehearsal at the next goal maintenance operator. This means the activation of the goal will eventually decay sufficiently for it to be removed from working memory.

Since preemptive strategies only use existing mechanisms common to many cognitive architectures, none of the strategy variants require modification to the architecture. The different organization of other architectures, however, may mean that only a subset of the strategies applies. In ACT-R, for example, working memory is of fixed size, and has limited space for elements that persist over time. As a result, it does not have the capability to keep a large number of goals in working memory, meaning that a preemptive rehearsal strategy would not be effective. On the other hand, since ACT-R's long-term memory elements also decay over time, these strategies can be

modified to prevent goals from being permanently forgotten. A retrieval-based strategy for working memory satisfies both needs, simultaneously bringing the goal into working memory to be checked and boosting its long-term memory activation.

4.4 Evaluation Overview

This evaluation of preemptive strategies is separated into three components. The first is a scaling experiment, to determine an empirical upper limit to the number of goals a retrieval strategy can maintain in working memory. The second is an experiment in a realistic setting, which confirms that preemptive strategies are a viable solution to the goal re-activation problem. This success led to the need to examine the limits of preemptive strategies, which drives the creation of an abstract domain where the environmental parameters are more easily controlled. Finally, the last component of this evaluation is in the abstract domain, where trends in strategy performance can be seen.

All evaluation of goal re-activation strategies are measured by two metrics, which are expected to trade off against each other in different strategies.

Percentage of goals completed This is the main performance metric for prospective memory, and is measured as the percentage of goals that the agent completed, out of the number of goals whose targets were objectively satisfied during the trial. An ideal strategy completes all the goals whose targets were satisfied.

Resources required per completed goal This metric evaluates the efficiency of a strategy by measuring the number of goal maintenance operators per completed goal. Although several other similar metrics could be chosen — for example, measuring the resource usage per encoded goal instead of per completed goal — this metric has the advantage that it can measure the effects of goals that are never relevant (as per P3). As an aid to understanding the trends in this metric, the percentage of decision cycles spent on goal maintenance may also be shown.

4.4.1 Preemptive Strategy Parameters

The parameters for preemptive strategies can be broken down as whether they relate to the timing trigger, the perception trigger, the rehearsal action, or the retrieval action. The timing trigger has one parameter — the length of the countdown, the *trigger period*, which determines how frequently the goal maintenance action is performed. The perception trigger, on the other hand, requires some feature of the environment to serve as an indicator that goal targets will be imminently satisfied.

The rehearsal action also has a parameter — the number of rehearsals to perform, or equivalently, the amount of activation increase. Finally, the retrieval action has no parameters.

4.5 General Scaling Evaluation

The goal of this experiment is to determine, empirically, an upper limit on the number of goals that a preemptive retrieval strategy can maintain. Such a limit exists because only a fixed number of goals can have their activations boosted in a single decision cycle, while the activations of the other goals in working memory decays. Even if a goal maintenance operation is performed every time step, the strategy cannot prevent other goals from decaying and being forgotten if there are a large number of goals in working memory. There is therefore a limit to the maximum number of goals that a strategy can maintain, at the point where the goal maintenance operation enters equilibrium with the activation decay process.

The experiment begins with a large number of goals in the agent's long-term memory. The agent then retrieves a goal into working memory every decision cycle. This continues until the activation of one of the goals in working memory decays sufficiently that it is removed from working memory. After this point, the number of goals in working memory remains at a steady state, since in each time step a new goal is retrieved and a previously-retrieved goal is forgotten.

Since this evaluation is not strictly about preemptive strategies, but about retrieval mechanisms, this experiment looks across multiple decay rates (d in Figure 2.4), which represent how quickly the activation of working memory elements decrease. Varying the decay rate is necessary, as although it is constant for any particular agent, the parameter is sometimes modified in ACT-R to fit human timing data. More generally, it may be useful to increase the decay rate if percepts from the environment change rapidly, such that retrievals are less likely to be influenced by previous percepts. The results of this experiment provide insight as to how goal re-activation may be solved in such domains.

The empirical results are shown in Figure 4.2. The plot shows the number of goals in working memory on the y-axis in a log scale, at the time when the first goal is forgotten, and the decay rate on the x-axis. The plot shows that as the decay rate increases, the limit on the number of goals decreases exponentially (as expected).

This fits the line in the plot, which has the equation $y = 0.0196 * e^{5.61/d} + 28.0$, where d is the decay rate, with a coefficient of determination $R^2 = 0.99$. This relationship arises from the base-level activation equation, and applies to both ACT-R and Soar. After the activation of a goal is initially boosted by its retrieval, it simply decays while other goals are retrieved. Over different decay rates, this results in the exponential curve shown.

It should be noted that the decay rate is not the only parameter that affects the number of time

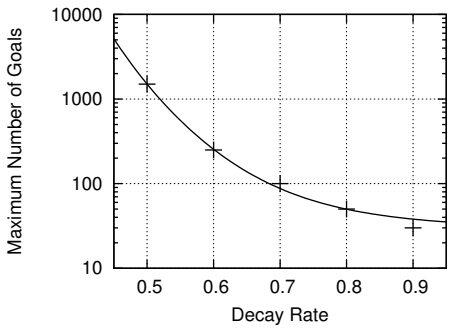

Figure 4.2: Maximum number of goals maintainable in working memory by retrieval wrt. decay rate

steps for which a goal remains in working memory. As described in Section 2.2.1, another relevant parameter is the forgetting threshold, which specifies the activation below which working memory elements are forgotten. Setting the threshold higher means that activations do not have to decay as much before the threshold is reached, and thus the goals would remain in working memory a shorter period of time. Varying the forgetting threshold would therefore scale the y-axis of Figure 4.2, while the trend across decay rates would remain the same.

In literature, while there is a default decay rate of 0.5 for both ACT-R and Soar, the forgetting threshold lacks a similar consensus (ACT-R sets the threshold to 0, while Soar sets the threshold to -2). Changing the forgetting threshold should have no qualitative effect on the results of this thesis. Increasing the threshold in this manner shortens the number of decision cycles before goals are forgotten, but as this applies to all strategies equally, comparisons between strategies still hold. On an absolute scale, it is expected that an agent with a lower forgetting threshold will maintain more goals in working memory than suggested by the results in this thesis and, in general, strategies should perform better when the forgetting threshold is lowered.

The results presented forms an upper bound on the number of goals that the preemptive retrieval strategy can achieve, raising the question of whether a similar baseline for the preemptive rehearsal strategy exists. Unfortunately, such a baseline is neither well-defined nor informative in actual agent performance. First, the performance of the rehearsal strategy depends on more parameters than that of the retrieval strategy: in addition to the decay rate and the forgetting threshold, there is also the number of rehearsals to consider, as well as the order in which goals are rehearsed and the level of activation of the goal prior to the rehearsal. Consider how each of these parameters affects the performance of the strategy. After the decay rate and the forgetting threshold, which have the same effect as they did for the retrieval strategy, the effect of the number of rehearsals is the simplest to understand: the more rehearsals the agent performs, the higher the activation of the element, and therefore the longer the element remains in memory. The last two parameters, however, are

problematic due to the lack of declarative access to the activation of working memory elements. Unlike in the retrieval strategy, where all retrieved goals have the same activation, the activation of the goals to be rehearsed depends on whether the goal has been used in rule matching — that is, the activation values are unknown. Thus, although the ideal order of goal rehearsals is to rehearse each goal in the order of being forgotten, the agent does not have access to this information. In order words, the upper bound on the rehearsal strategy is that all goals can be kept in working memory (if their activations are sufficiently high and are rehearsed in ascending order), while the lower bound is that a goal is forgotten almost immediately (if it has low activation and is not rehearsed). There is therefore little practical benefit in computing a limit on rehearsals, as even an estimate of this would require specifying all the factors listed above.

For agent designers, these results demonstrate that the number of goals maintainable by preemptive strategies may be as high as the thousands. However, this requires that the agent perform no operations other than retrieval, which means that the foreground task would be neglected (A4).

4.6 Mobile Robot Domain Evaluation

This evaluation serves as a proof-of-concept for preemptive strategies, to show that they allow agents to solve the goal re-activation problem. In this domain, the Soar agent controls a simulated SuperDroid robot in an indoor environment, which has been used in previous evaluations of Soar [14, 33]. A bird's-eye-view of the domain is depicted in Figure 4.3. The figure shows the environment is divided into rooms, separated by walls (in black). Rooms in this domain are restricted to be rectangular, meaning that irregularly shaped areas, such as the one at the right of Figure 4.3, are divided into multiple rooms. Some rooms are also separated by doorways, which are gaps in the wall; due to irregularly shaped rooms, the entry into a room is not necessarily preceded by passing through a doorway. Additionally, scattered around the domain are objects with different colors, shapes, and sizes; these are depicted as dots in Figure 4.3. These objects can be picked up, carried, and put down by the agent, and the agent can carry multiple objects at the same time. However, the agent's perception of the objects is limited by the walls: only the objects in front of the robot in the agent's current room are perceivable to the agent. Since this domain is a simulation, the decision cycles of Soar and the simulated time are decoupled, but decision cycles remain under the 50 millisecond reactivity threshold.

A trial in this domain involves the agent picking up and delivering objects to other rooms, while being constrained to a predefined "patrol" route, meaning the agent must visit the rooms in a particular order. The blue line shows the previous locations of the robot in Figure 4.3. For any trial, objects with randomly generated attributes are placed uniformly randomly in the environment. Each trial contains ten deliveries, and the objects that the agent must deliver, as well as the rooms

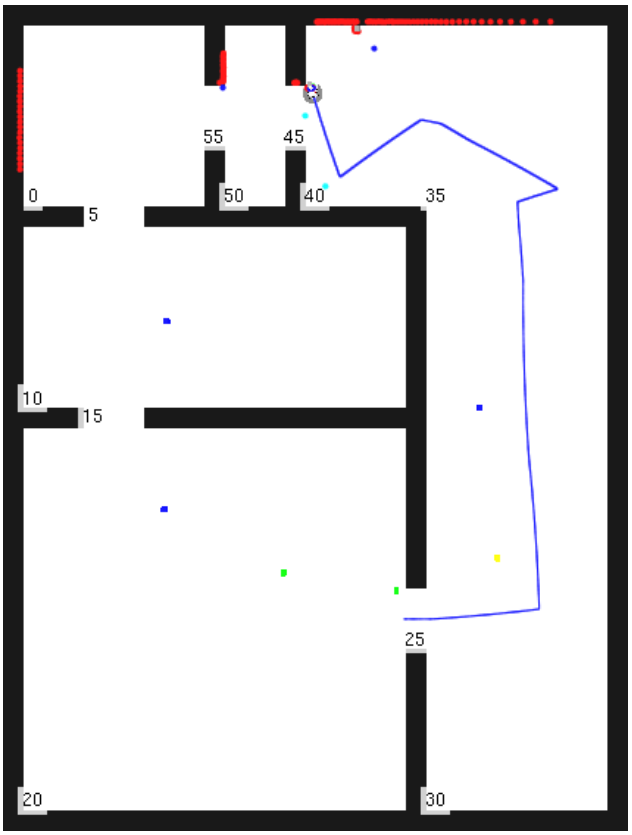

Figure 4.3: The simulated mobile robot domain

that they must be delivered to, are also randomly assigned. These deliveries are determined by the description of the object and the room to which the object should be delivered; in particular, the agent does not know the location of the objects to be picked up. Once the object is delivered to a room, it remains in that location, and the agent does not pick up that object again; therefore, these pick up and deliveries are one-time goals.

Before a trial begins, the deliveries are pre-loaded into the long-term memory of the agent. A successful delivery of an object in this domain requires completing two prospective goals. First, the agent must recognize that an object in the current room is an object to be delivered elsewhere, and that the agent should pick it up. The target of this goal is therefore simply the perception of the object. Second, when the agent enters a room that is the destination of a delivery, it must recognize it as such, and put down the object; the target is the agent being in the destination room while carrying the object to be delivered. The size of the rooms and the distances travelled means that, by default, these goals will be forgotten from memory, and that a prospective memory strategy must be used to ensure their completion.

4.6.1 Agent Description

This section describes the parameters used for each of the strategies, as well as how the strategy is applied to the mobile robot domain.

This evaluation examines all four variations of preemptive strategies, which together require three parameters: the frequency of the timing trigger, the percept used for the perception trigger, and the amount of rehearsals for the rehearsal strategies. For this evaluation, the feature of passing through a doorway is used for the perception trigger, since it is predictive of the subjective satisfaction of the target. Since the agent can only perceive objects in the same room, this means that doorways are highly predictive of the imminent satisfaction of the targets of both goals necessary for a successful delivery. Note that other features could be chosen for this trigger; using the perception of objects as the trigger stands out as an obvious choice. This trigger, however, only applies for the goal of picking up objects, and is not indicative of the satisfaction of the targets of the delivery goals, which are instead based on the location of the agent. For this reason, the doorway is used for the perception trigger instead. After some initial experimentation, the timing trigger is set to occur every 400 decision cycles (a little under 20 seconds of real time), as it is just frequent enough for the agent to perform well, while still being subject to the effects of the other parameters. For the preemptive strategy agent that uses rehearsal, a logarithmic progression of rehearsals (4, 16, 64) is used to obtain a spread of settings.

As a concrete example, consider an agent using a perception-triggered preemptive retrieval strategy. At the beginning of the trial, the agent is created with a rule that matches when the agent

is in a doorway. The delivery goals are then loaded into long-term memory, but these goals do not exist in the agent's working memory. For this example, consider a single goal of picking up a large green object to Room 5. Once the trial begins, the agent moves along its patrol from room to room. When it passes through a doorway, the rule matches, leading the agent to initiate a search its long-term memory for goals. This results in the goal being retrieved into working memory. If the next room that the agent enters contains a large green object, a rule detects that the target of a goal has been satisfied, and proposes the goal as an option for pursuit. Alternately, if such an object does not exist in the room, the agent continues its patrol while the activation of the goal decays, leading to its eventual removal from working memory.

Assuming that the agent successfully picked up the large green object, the next step is for the delivery to occur. The agent's working memory contains a list of objects it is carrying, so it can perceive the object, but may not know the room to which it should be delivered (which is part of the target of the goal). As the agent continues its patrol, passing through a different doorway would once again cause the rule to match, this time retrieving into working memory the goal of delivering the object. As before, if the next room is the correct destination, the agent could then pursue the goal by putting down the object; or if the room is not the destination, the agent could continue its patrol.

4.6.2 Evaluation Parameters and Metric

In addition to the preemptive strategy parameters mentioned in Section 4.4.1, there are two main parameters to this domain. Both of these parameters influence the length of time before the agent perceives a delivery object — that is, they affect the length of the retention interval. The first parameter is the decay rate (d in Figure 2.4), which determines how quickly a memory element is forgotten. By increasing the decay rate, thereby shortening the time that a goal remains in working memory, the retention length is increased relatively. The second parameter is the speed of the robot, which determines how quickly it moves forward in the simulated environment, measured in simulated meters per second. The slower the robot's movement, the longer the goal must remain in working memory and not be forgotten. Again, this attempts to simulate direct changes to the retention interval. Both parameters change the temporal dynamics of the domain, which has an impact on the strategies' performance.

For the robot delivery task, the main measurement of performance is the percentage of deliveries that the agent successfully completes. Unless the agent completes all of its deliveries, it is stopped after the third round of patrol. The efficiency of the agent is also measured, as the ideal agent should only retrieve and rehearse as necessary for the task and no more; this is measured in the number of goal maintenance operators the agents performed. Agents that complete the same number of

Decay Rate	Timing Trigger			Perception Trigger		
	4	16	64	4	16	64
0.34	100%	100%	100%	100%	100%	100%
0.38	100%	100%	100%	76.36%	80.0%	94.54%
0.42	56.36%	60.0%	41.81%	7.27%	5.45%	5.45%
0.46	3.63%	3.63%	7.27%	0.0%	0.0%	0.0%
0.50	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

Table 4.1: Performance of the preemptive rehearsal strategy in the mobile robot domain, wrt. decay rate

deliveries with fewer decision cycles spent on goal maintenance are preferred.

The following two sections present the results of the rehearsal strategy and the retrieval strategy separately.

4.6.3 Preemptive Rehearsal Strategy

Table 4.1 shows the percentage of goals completed for the agent at different decay rates, at a speed of 0.7. The second heading indicates the number of rehearsals the agent performed per trigger. First, comparing across columns where the decay rate is held constant, the number of rehearsals appears to have a noisy but neutral effect. Since goals have different amounts of activation before they are rehearsed, and the order in which they are rehearsed is random, it is not surprising that the results are noisy. The overall neutral effect is surprising, as one may expect more rehearsals to increase the number of completed goals. This may be explained by how the rules implementing the rehearsal actions already causing a large boost in activation, and the further rehearsals as specified by the strategy parameter have no additional effect. Looking down the columns, however, the performance of all agents drops sharply over small increases in the decay rate. This rapid decline is due to differences between varying the decay rate and varying the retention interval. To understand the difference, consider an agent with only one goal, and is using a timing-based preemptive rehearsal strategy sufficient to prevent the goal from being forgotten. If the retention interval is increased, since the goal is continually maintained in working memory, there is no effect on the agent's ability to pursue its goal. If the decay rate is increased, however, *the timing trigger may no longer be sufficient to maintain the goal in working memory*, since it now takes fewer decision cycles for the activation of the goal to fall below the forgetting threshold. This is what causes the sharp decrease in performance in Table 4.1.

To further examine this sharp drop-off in performance, a decay rate of 0.4 was selected, and then the speed of the agent was varied. This decay rate is one where both strategies start exhibiting performance declines, while the speed is varied as it also affects the retention interval. The result of

Speed	Timing Trigger			Perception Trigger		
	4	16	64	4	16	64
0.3	90.9%	90.9%	90.9%	18.1%	18.1%	22.7%
0.5	95.4%	81.8%	95.4%	22.7%	22.7%	13.6%
0.7	92.7%	92.7%	92.7%	56.3%	36.3%	34.5%
0.9	87.2%	92.7%	92.7%	40.0%	36.3%	34.5%
1.1	92.7%	98.1%	90.9%	52.7%	36.3%	23.6%

Table 4.2: Performance of the preemptive rehearsal strategy in the mobile robot domain, wrt. robot speed (in simulated units per second)

varying the agent speed is shown in Table 4.2, again with the number of rehearsals in the second heading. Comparing rows, the performance of the timing trigger agent remains roughly uniform, while performance of the perception trigger agent decreases as the speed of the robot decreases. While changes in robot speed are correlated with changes in the retention interval, they are not exactly equivalent. A slower robot experiences more decision cycles until the goal target is satisfied, *but this does not affect the timing trigger*, which is specified in decision cycles and not real time. Since the activation decay of goals is calculated over decision cycles and not real time, a perception-triggered agent with reduced speed is in effect lengthening the time between goal maintenance actions.

Taken together, these results suggest that for a preemptive rehearsal agent, a perception trigger performs well in a smaller range of environments as compared to the timing trigger, and is further subject to effects from the speed of the agent, or more generally, environmental factors that affect the temporal dynamics of the target.

4.6.4 Preemptive Retrieval Strategy

The previous section compared the timing and perception triggers for a preemptive rehearsal strategy; this section looks at the same triggers for a preemptive retrieval strategy. As a reminder, this evaluation aims to understand the performance of the perception trigger, and how it compares to the timing trigger. Note that unlike the rehearsal strategies, which have a parameter for the number of rehearsals, the retrieval strategies have no such parameter. The results in this section use the same trigger period of 400 time steps as the previous section.

The performance of agents with different decay rates using retrieval strategies are shown in Figure 4.4, with the percentage of deliveries completed on the y-axis and the decay rate on the x-axis. The figure shows that as the decay rate is increased, the performance decreases for both the timing-triggered and perception-triggered preemptive retrieval strategies; the choice of trigger has minimal effect aside from an outlier. However, the efficiency of the strategies differs. Agents

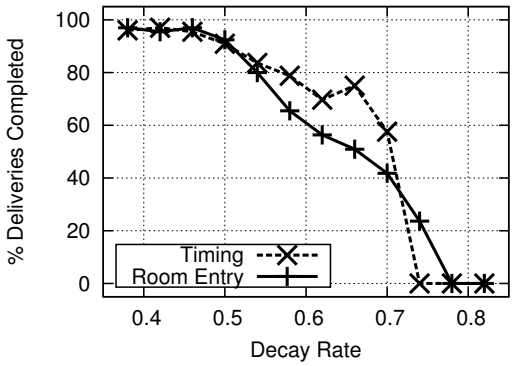

Figure 4.4: Performance of the preemptive retrieval strategy with different triggers

using the timing trigger occurs 50% more frequently than agents using the perceptual trigger, and this ratio grows as the decay rate is increased; this means that the agent may be interrupted from it's current task. The efficiency of the room entry trigger is due to it being highly predictive of opportunity; many of the retrievals due to timing triggers do not lead to action on the part of the agent if, for example, deliveries are retrieved when the agent does not perceive any objects. With the room entry trigger, the embedded domain knowledge makes it more likely that an opportunity to act is present. The high probability of a relevant retrieval allows the agent to reduce the number of goal maintenance operators without comprising its performance.

Compared to the results from the rehearsal strategies, a trend worth exploring is that retrieval strategies appear to perform well across a broader range of decay rates. In fact, this trend has the same underlying cause as the sharp decline seen in rehearsal strategies: that agent performance depends critically on a particular value of the decay rate, above which the agent cannot maintain goals in working memory. If the number of rehearsals is not sufficient to retain the deliveries in memory, all deliveries will be eventually forgotten. Since the rehearsal agents do not retrieve forgotten deliveries in this evaluation, they fail to complete any more deliveries — hence the sharp drop-off in performance for the rehearsal agents. In contrast, the decay rate does not need to be as carefully calibrated for retrieval strategies, as the agent recovers when a delivery is forgotten. Thus, although deliveries may be forgotten earlier due to the lack of rehearsals, this does not affect agent task performance until it reaches extreme levels. For the same reason, agent performance remains stable across changes in movement speed: even if a delivery is forgotten after its retrieval, the agent has a second opportunity to complete the delivery when it enters the room again and triggers a second retrieval.

4.6.5 Mobile Robotics Summary

The mobile robotics domain examined the two triggers and the two goal maintenance actions for preemptive strategies, with the goal of comparing the perception trigger to the timing trigger. Since we could not directly control for variations in the retention interval, we varied the decay rate and the speed of the robot instead, despite these parameters having slightly different effects. While both triggers perform well when the decay rate is low, the perception trigger is more sensitive to changes in the decay rate and in the robot speed. This is because the frequency of the perception trigger has no relationship with when goals are removed from working memory, thus emphasizing the importance of the selection of the perceptual feature. The perception trigger is better suited for a retrieval strategy for this reason, and leads to a strategy that performs well in a wider range of environments.

For agent designers, these results suggest that the selection of parameter values for preemptive strategies should not be based only on the other parameters of the agent, such as the decay rate, but also on the temporal dynamics of the environment. Since there is no easy method of capturing the temporal dynamics of a domain, the simplest solution to finding good strategy parameters may be to perform pilot experiments. If such experiments are not possible, the preemptive retrieval strategy (with either trigger) has a more robust performance than the preemptive rehearsal strategies as long as a predictive perception trigger is used, and may be the better choice given the unknown temporal dynamics of the environment.

4.7 An Abstract Prospective Memory Domain

The results from the mobile robot domain demonstrate that preemptive strategies can solve goal re-activation problems, but the dependence of the results on agent and domain parameters suggest that more experiments are necessary. Unfortunately, there are two major concerns for using the mobile robotics domain for such an evaluation. First, mobile robotics requires parameters that do not translate well into other domains. For example, the decreasing speed of the agent was shown to reduce the performance of the perception-triggered rehearsal strategy, a result that is difficult to generalize to stationary agents (or agents that are not embodied). The second concern, related to the first, is that the simulated physical environment makes it difficult to have fine control over experimental parameters. For example, the length of the retention interval is affected by both the speed of the robot as well as the distribution of the packages, when it would be ideal to have a single parameter that affects retention length.

Instead of using the mobile robotics domain, we instead create a new domain with generalizable results and controllable parameters. Specifically, these parameters should be individually controlled

for:

P1 **Number of goals** The number of prospective goals is an obvious scaling parameter for goal re-activation. An agent may accumulate a large number of goals, some of which may never be pursued. This parameter explores how the total number of goals in the lifetime of the agent affects its performance. An ideal strategy will perform equally well irrespective of the total number of goals it encounters.

In general, it is expected that the performance of strategies will decrease as the number of goals increases. Since forgetting from working memory is determined by base-level activation decay, there should be an upper limit to the number of goals that a preemptive strategy could effectively maintain.

P2 **Length of the retention period** Aside from the number of goals, this is the other obvious scaling parameter. From psychology literature, the length of the retention interval is one of the most predictive conditions of prospective memory behavior — not of performance, but of the choice of strategy. The longer the retention interval, the more likely that a spontaneous strategy will be used over a monitoring/preemptive strategy. Since the preemptive strategy requires computational resources proportional to the length of the retention interval — a longer retention stage means more goal maintenance actions are necessary — this trend is likely to also apply to artificial agents, as the cost of preemptive strategy become higher than that of the spontaneous retrieval strategy. At the same time, a lengthier retention interval would also mean that the long-term memory activation of goals would decrease, which would affect retrieval bias. Since these effects are contradictory, it is difficult to predict how changes to this parameter would impact either strategy.

P3 **Goal encounter rate** Although this thesis is on goal re-activation, some goals given to the agent may never be applicable. A goal whose target is never satisfied has, in effect, an infinitely long retention stage. For example, consider the message-delivery task. If the agent never encountered the recipient of the message, the goal target would never be satisfied. This is likely to have a negative effect on the preemptive strategy, which continually require resources during the retention stage to prevent goals from being forgotten.

As a result of these considerations, we decided that goal re-activation strategies are better evaluated in an abstract domain, where these parameters can be controlled. This domain is not designed to be a single test to classify goal re-activation strategies as either sufficient or inadequate; rather, it is designed to stress test the strategies, to identify locations in the parameter space where the strategies may perform worse than others, and also to allow comparison between strategies in that parameter space. Beyond the parameters of the environment, both preemptive strategies

and spontaneous retrieval strategies have additional parameters that affect their performance and cost, establishing a tradeoff between the two. Instead of specifying a threshold above which strategies must perform, we instead leave it to the agent developer to determine the desired level of performance, and aim only to provide guidance on the strategy parameters that allow that performance.

Generally speaking, the abstract domain is a discrete-time simulation, with each time step in the simulation lasting one decision cycle of the agent. At every time step in the simulation, a set of symbolic *features* is presented to the agent; a set of features forms a *percept* for the agent. Some of these features form the targets of goals, and the agent must correctly recognize that the goal could be pursued when those features are perceived. Since this thesis is not concerned with the execution stage of prospective tasks (see Section 2.1.2), if the agent decides to pursue a goal, the only action the agent must take is to signal the environment that the goal is completed, an action that always succeeds. The focus of the domain is therefore on the agent recognizing that the target of a goal is satisfied.

For any trial in this domain, a number of long-term memory elements are first created as the features. The number of features depends on the number of goals (P1). Each goal has a fixed number of features as its target, but goals may share target features. Additionally, each goal has an identifying number, with which the agent must signal the environment to indicate that it is pursuing the goal. Other features that are not part of the target of any goal are also created, as distractors to the agent. All the features are pre-loaded into the agent's long-term memory before a trial begins.

Once the desired number of goals are created, each goal is assigned an encoding time and, probabilistically, an initiation time; whether a goal will be initiated is determined by the goal encounter rate parameter (P3). The encoding time is when the goal and its target are presented to the agent, while the initiation time is when the target features are observed by the agent. These times are chosen according to the desired retention interval length (P2), which is specified by four numbers: the time step at which the encoding stage begins, the number of time steps until the encoding stage ends, the time step at which the initiation stage begins, and the number of time steps until the initiation stage ends. The "stages" here are periods within the trial during which all goals are encoding and initiated; the encoding and initiation times of a specific goal are drawn uniformly randomly from these periods. By varying the time steps at which each stage begins, the average length of the retention interval can be controlled.

Finally, once all goals have an encoding time and (optionally) an initiation time, the sequence of percepts for the entire trial is generated. The percept for the time steps at which goals are initiated are assigned first, to be the set of features that form the target of that goal. Once this is complete, a random subset of features is selected as the percept of the first time step, and a different random subset selected as the percept for the last time step. The percepts for the remaining time steps are

```
function CREATEPERCEPTS(percepts, start, end)
    mid ← (end − start)/2
    percepts[mid] ← INTERPOLATE(percepts[start], percepts[end])
    CREATEPERCEPTS(percepts, start, mid)
    CREATEPERCEPTS(percepts, mid, end)
```

Algorithm 4.1: The percept generation algorithm

then generated recursively. The algorithm iterates through sections of the percept sequence that are bordered by two determined percepts; this is either between the first time step and the first initiation time, between adjacent initiation times, or between the last initiation time and the last time step. A noisy interpolation of the two endpoints is generated and used as the midpoint for the sequence; the algorithm then recurses on each half to continue the process. Every random percept is checked against every goal, to ensure that it does not contain the target of any goal that should not be satisfied at that time. Pseudocode for this recursive algorithm is shown in Algorithm 4.1.

A deliberate decision in the design of this domain is that there is no regularity in the percepts of the agent, meaning that no perceptual trigger is possible. We consider this acceptable for two reasons. First, as demonstrated in the mobile robotics domain, the choice of the percept has a large effect on the performance of the agent, but is also extremely domain dependent. One possibility for representing the quality of the percept is to parameterize the correlation between the trigger and the target of the goal; while this works for individual goals, it is unrealistic for *all* goals to be correlated to a single percept in this way. Second, it seems obvious that the selection of a predictive percept will lead to high goal re-activation performance, since in the extreme case of a perfectly predictive percept, the agent behaves as though the goal is encoded in rules. Since this thesis is not concerned with selecting predictive features, we decided that focusing on the performance of the timing trigger would be more informative for researchers facing the goal re-activation problem.

Once the entire percept sequence is generated, the trial is ready to begin. The different stages of a prospective task (from Section 2.1.1) can be examined in a trial:

1. The *encoding* stage is when the agent is presented with the goals and their targets. This is presented separately from the percept for that time step. At this point, the agent must store the goal, its target set of features, and the identifying number of the goal into long-term memory.

2. The *retention* stage exists to allow time for the activation of goals to decay; random percepts are presented during this period. At this point, if the agent has not made any errors in the encoding stage, all goals from the environment (specified by P1) are stored in long-term memory. The agent is not given any foreground task, and is programmed to remain idle unless the strategy requires agent deliberation. For preemptive strategies, the timing and perception triggers will occur, causing the agent to perform goal maintenance operations.

3. The *initiation* stage is when the goal targets are objectively satisfied, and the agent must compare the percept to the target of its goals. If the agent recognizes that a goal target is subjectively satisfied, a rule proposes the option of pursuing the goal.

4. The *execution* stage is when the agent has decided to pursue the goal. For this domain, the agent signals the environment with the identifying number of the goal it is pursuing.

5. The *completion* stage occurs after the agent executes a goal. Here, the agent modifies the goal's representation in memory to mark is as completed.

The rest of this evaluation uses the default decay rate of 0.5 and selects a forgetting threshold of 0; the higher threshold is chosen so goals are forgotten more quickly, to reduce the run-times of the experiments. Each trial in the abstract domain is repeated 20 times, with each trial randomizing the encoding time, the initiation time, and the target features of all goals. The encoding stage starts at time step 0 and lasts 300 time steps, which is sufficient for all goals to be encoded while allowing variation between trials. The initiation stage always lasts 375 time steps — again, to allow goal targets to be presented and to allow variation in ordering between trials. The starting time of the initiation stage is varied according to P2, but is always after time step 1500; this allows sufficient time for the activation of goals in working memory to decay and for the goals to be removed from working memory. For trials that last 3000 time steps, the initiation stage begins at time steps 1500, 1875, 2250, and 2625; this corresponds to initiation stages that starts when 50% of the simulation is over, when 62.5% of the simulation is over, and so on, until the end of the last initiation stage is the last time step of the trial.

4.8 Abstract Domain Evaluation

Using the abstract domain described above, this section contains the results for the timing-triggered preemptive rehearsal strategy and the timing-triggered preemptive retrieval strategy. Since goal re-activation is a new problem in cognitive architectures, and the abstract domain a new environment for evaluation, the quality of the results can only be established relative to different agent and environment parameter settings. The goal of these experiments is to identify trends in the performance of the timing-triggered preemptive strategies with respect to the parameters given in Section 4.7, and to identify failure cases for these strategies.

4.8.1 Timing-triggered Preemptive Retrieval Strategy

Figure 4.5 shows the percentage of goals completed by the timing-triggered preemptive retrieval strategy, as a function of the trigger period. Each sub-figure shows the results of an environment

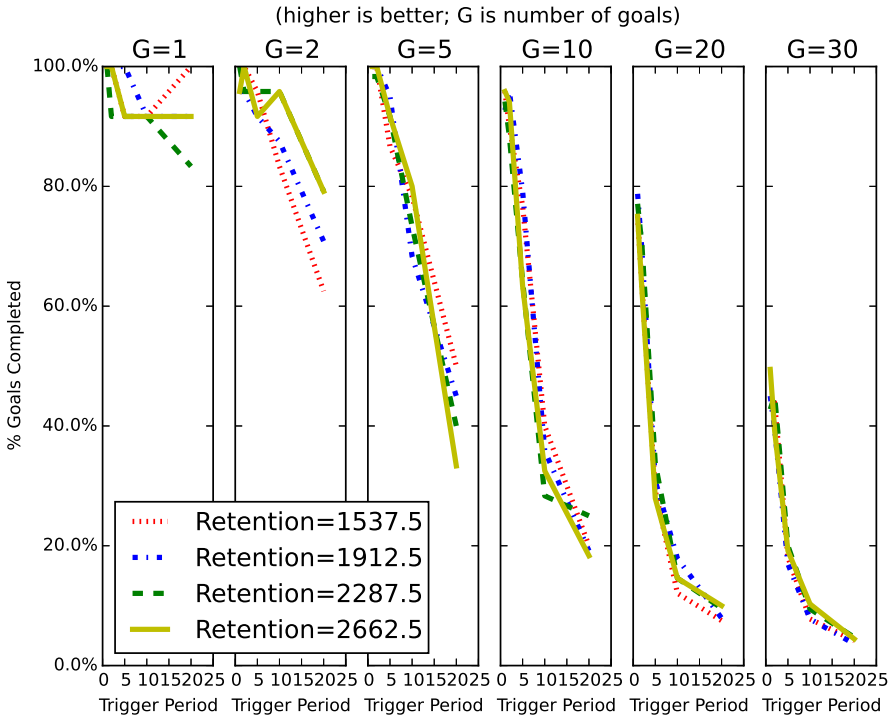

Figure 4.5: Performance of the preemptive retrieval strategy wrt. number of goals

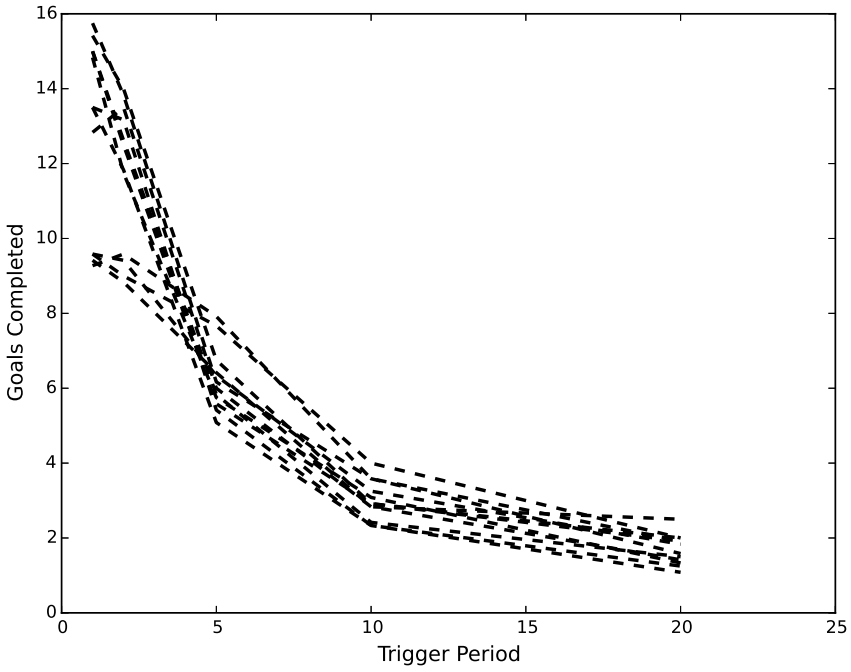

Figure 4.6: Absolute performance of the preemptive retrieval strategy

with a different number of goals (indicated in the title), with the lines in each figure representing the result from different retention interval lengths (that is, the length of time between the encoding and the initiation stages).

Consider the left-most subplot, where agents only have one goal to re-activate. Along the x-axis, even with lengthy trigger periods (longer periods before the goal is retrieved), the majority of strategies can ensure the goal is in working memory when its target is satisfied for the majority of the time (as demonstrated by the high values on the y-dimension). Comparing across subplots, however, as the number of goals increase, shorter trigger periods are needed ensure that the largest number of goals are re-activated. Even with a trigger period of 1, however, the timing-triggered preemptive retrieval strategy can still only re-activate about 50% of its 30 goals in the right-most subplot. To put this in context, this means that if an agent has a moderate number of goals, it can still only achieve half of them *despite spending every time step on goal maintenance*.

On closer inspection of these results reveals the underlying cause: that there is an upper limit, or a *carrying capacity*, based on the trigger period. This can be seen by looking at the absolute number of goals completed in Figure 4.6, which shows all parameters of the environment on a single plot, with the absolute number of goals completed on the y-axis and the trigger period on the x-axis. Notice that although the actual number of goals varies, they roughly follow the same trajectory. This behavior is due to retrievals being the sole source of activation boost for the goals. When a retrieval occurs, the retrieved goal receives an activation boost, but if its target is not satisfied, the activation simply decays over time, until the goal is forgotten again. Since the activation boost from retrieval is constant, this also means that goals spend a constant amount of time in working memory, where the total number of goals in working memory is determined entirely by how often retrievals occur. Thus, even if more goals were encoded by the agent, the number of goals in working memory does not change, leading to a pattern of goal completion that depends only on the trigger period. This also explains why, aside from when the number of goals is small, the length of the retention interval has no effect on goal re-activation performance (comparing lines in the same plot in Figure 4.5).

Figure 4.7 shows the proportion of decision cycles spent on goal maintenance operators, with the proportion on the y-axis and the trigger period on the x-axis. A proportion is used instead of the absolute number to allow these results to generalize across the length of the trial (here they all end after 3000 decision cycles).

Before we discuss the results, we note that the number of goal maintenance operations plotted does not match the timing trigger, as one might believe it should. The reason for this increase in efficiency is that the agent does not directly attempt to retrieve goals into working memory at the trigger. Rather, the agent keeps a count of the number of prospective goals it has yet to achieve, on which the timing trigger is also conditioned. That is, a retrieval is attempted only when the timing countdown reaches zero *and* there are prospective goals not currently in working memory.

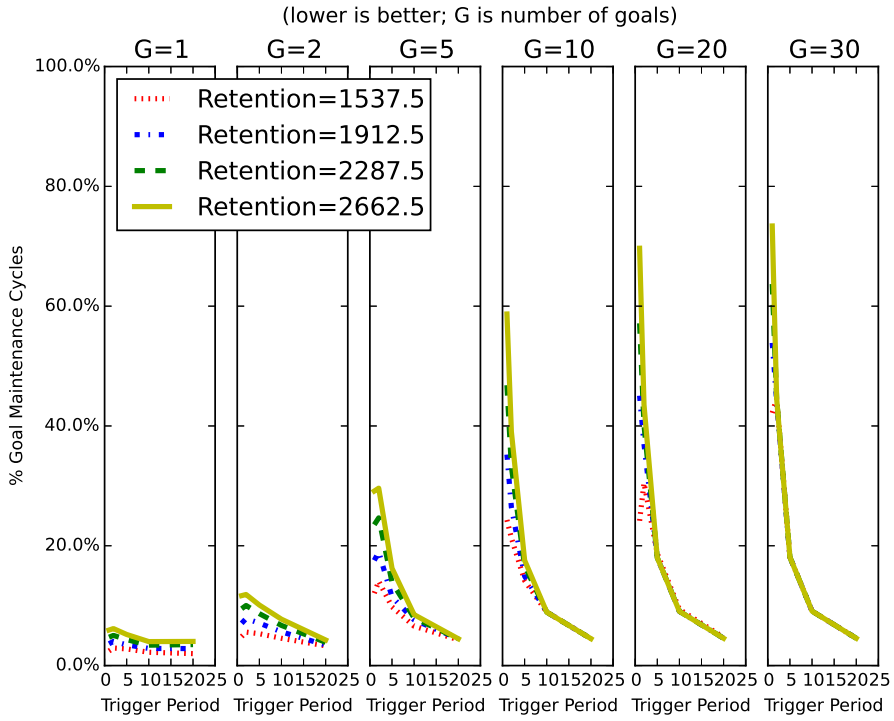

Figure 4.7: Goal maintenance operators of the preemptive retrieval strategy wrt. number of goals

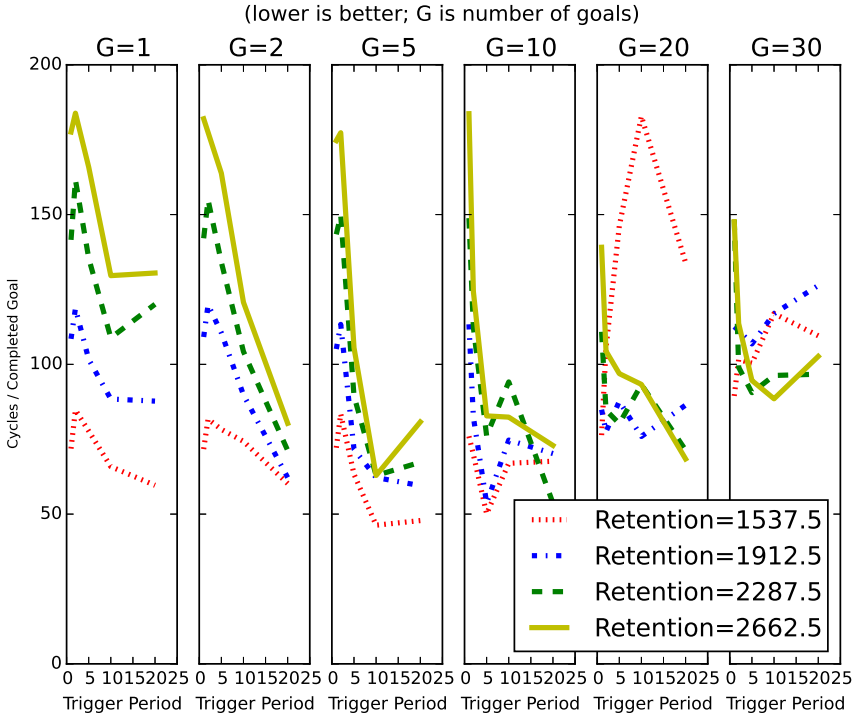

Figure 4.8: Efficiency of the preemptive retrieval strategy wrt. number of goals

Although this count is itself subject to forgetting, it is rarely actually forgotten, as its activation is boosted whenever it is used in a rule firing — that is, whenever the timing trigger occurs and a goal is in fact retrieved. Only when there are no prospective goals in long-term memory does the rule for the timing trigger fail to fire, in which case activation for the count will decay. This indirection allows us to greatly reduce the number of necessary goal maintenance operations.

Inspecting each plot individually, as the trigger period increases, the number of goal maintenance operations decreases; this is as expected. For short trigger periods, however, it can be seen that a shorter retention interval leads to marginally fewer retrievals when the trigger is frequent. At the same time, increasing the number of goals (comparing across plots) leads to the agent performing more retrievals. All of these results make intuitive sense. As more goals that must be retained for a longer retention interval means more retrievals must be used to keep goals in working memory.

Finally, the efficiency of the preemptive retrieval strategy is plotted in Figure 4.8, with the number of goal maintenance operators required per completed goal on the y-axis, and again with the trigger period on the x-axis. This metric of *efficiency* is used to indicate the amount of resources expended per unit of measurable performance. While the plots in Figure 4.8 appear chaotic, there is a trend when the lowest points are compared across subplots. In the two left most subplots, the most efficient trigger period is large; as the number of goals increases (moving to subplots on the right), the most trigger period that is most efficient decreases. This implies that the most efficient trigger period changes depending on the number of goals: with only ten goals, a trigger every five time steps is the most efficient, while a more frequent trigger is more efficient with more goals, and conversely, fewer goals allow for longer trigger periods. With fewer goals, each individual goal has a higher chance of being retrieved, and therefore will remain in working memory for a longer period of time; in turn, this means that the trigger can occur less frequently with no impact on performance. Conversely, once the number of goals is above the carrying capacity, the agent cannot retrieve goals sufficiently quickly to keep all goals in working memory at the same time, and therefore needs a more frequent trigger to keep as many goals in working memory as possible. The remaining parameter, retention interval, shows a clear effect on efficiency when the number of goals is small, but this effect is removed when more goals are present. This can be interpreted as the effect of the number of goals on the number of completed goals being larger than the effect of the retention interval on the number of goal operations, hence the effect of the latter on efficiency is washed out.

It should be noted that an extremely frequent trigger for a retrieval-based preemptive strategy is tantamount to keeping the goal in working memory. Take a strategy that retrieves a forgotten goal every decision cycle — this means that as soon as a goal is forgotten, it is immediately retrieved again, with a corresponding boost in working memory activation. In fact, this strategy works better than a rehearsal-based one, which explicitly attempts to prevent the activation of a goal from

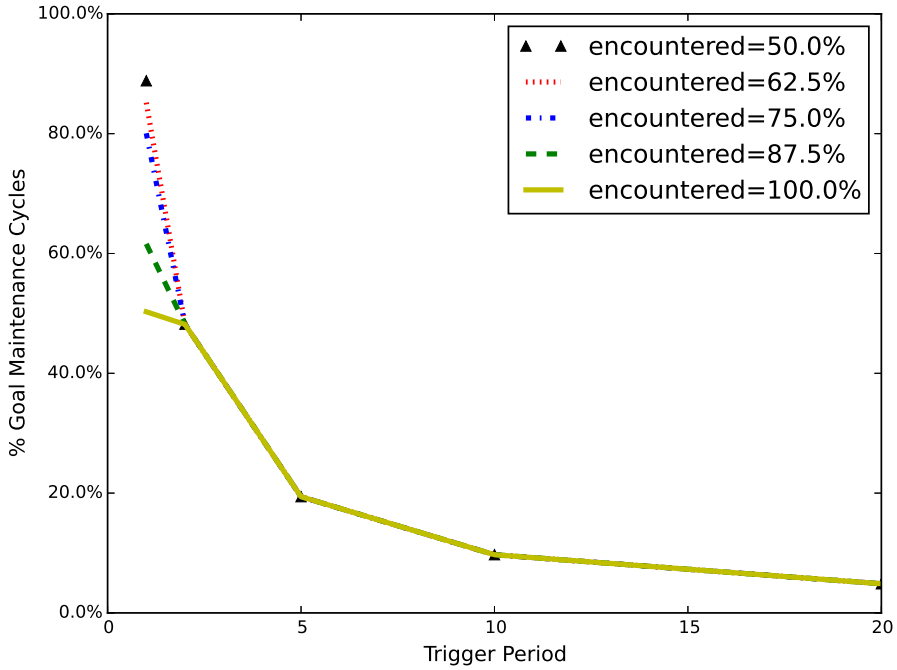

Figure 4.9: Resource usage of the preemptive retrieval strategy wrt. goal encounter rate

dropping below the forgetting threshold. This can be seen in the amount of resources used by the agent, measured as the number of time steps taken in managing goals. Due to the inability for the agent to examine a memory element's activation, a rehearsal strategy must boost the activation of a goal at every step to be safe, lest its activation is about to fall below the threshold. In contrast, a strategy based on retrieval only performs an action when goals are forgotten, and otherwise allows the agent to pursue another task. This lazy approach to maintaining goals in working memory — in the sense of only performing work when necessary — allows savings in resource consumption. As a secondary result, the time steps needed to maintain goals do not scale linearly with how frequently the trigger is used. A trigger for preemptive action every 20 time steps, for a trial of 3000 time steps, gives about 150 triggers total; a retrieval-based agent uses about 135 of those, for a utilization rate of 90%. An agent with a trigger every time step (for 3000 triggers total), on the other hand, only uses about 730 of those, for a much lower utilization rate of 24.3%. This suggests that the agent designer can set the trigger frequency as an upper bound on the number of goal maintenance operations, with the strategy adapting to only perform as many operations as necessary.

The second experiment looks at the effects of different goal encounter rates. Figure 4.9 shows the proportion of decision cycles spent on goal maintenance operators, with the proportion on the y-axis and the trigger period on the x-axis. Note how the results for the different goal encounter

rates diverge with a high trigger frequency (a low trigger period, so towards the left side of the plot); in particular, as the proportion of encountered goals decrease, the agent spends more time steps performing goal management. Since a retrieval is performed at every preemptive trigger *until the goal is completed*, if the latter never occurs, the trigger will continue to lead to action. This means that the cost of preemptive strategies scale not only with the number of goals, but also with the lifetime the agent. This divergence, however, disappears with lengthier trigger periods. This is because, in the environment with these parameters, the agent already fails to complete some of the goals, and already incurs the costs of uncompleted goals; varying the goal encounter rate therefore has little effect. This major problem of wasted operators required for preemptive retrieval strategies has no simple solution.

In summary, the preemptive retrieval strategy performs well with few goals and frequent triggers, but at the cost of low efficiency. Additionally, if there are goals with targets that are never satisfied, the agent will continue incurring the cost for those goals.

For agent designers considering the use of a timing-triggered preemptive retrieval strategy, these results suggest that it is critical to estimate the carrying capacity that the strategy affords. The capacity depends not only on the trigger period, but also on the decay rate and forgetting threshold of the agent. If the number of goals is below this capacity, it is likely that the strategy can ensure that the majority of goals are in working memory when their targets are satisfied, allowing the goals to be pursued. If the number of goals is above this capacity, however, a different strategy may offer better performance.

4.8.2 Timing-triggered Preemptive Rehearsal Strategy

As a reminder, these experiments aim to identify trends in the performance of preemptive strategies as well as expose cases where the strategies fail.

The performance the preemptive rehearsal strategy is shown in Figure 4.10. The figure requires a little explanation. Each plot represents a particular environmental setting, defined by the number of goals and of the retention interval; these are represented in the title of each plot as G and R respectively. Within each plot, the y-axis is the percentage of goals completed by the preemptive rehearsal strategy, with the trigger period on the x-axis. However, the preemptive rehearsal strategy has the additional parameter of the number of rehearsals; this is represented by the individual lines in each plot. Since the results across the number of rehearsals are very similar — whether 10, 20, 50, or 100 rehearsals were performed — they are not individually labeled. This replicates the results from the mobile robot domain in Table 4.1.

From this figure, the performance of the preemptive retrieval strategy shows some similar trends as that of the retrieval strategy. As with the retrieval strategy, the retention length (looking across

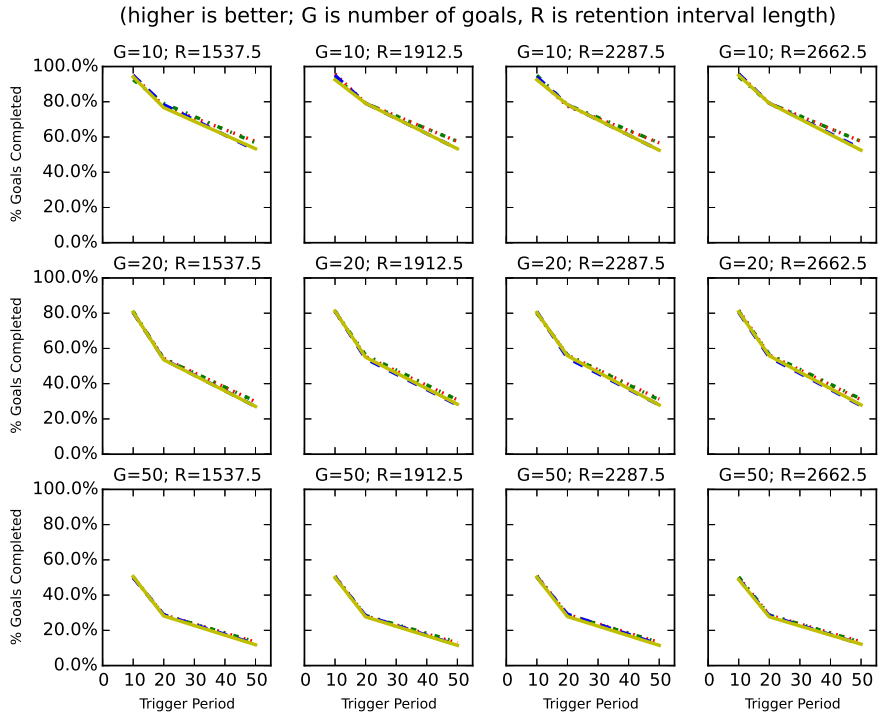

Figure 4.10: Performance of the preemptive rehearsal strategy wrt. number of goals

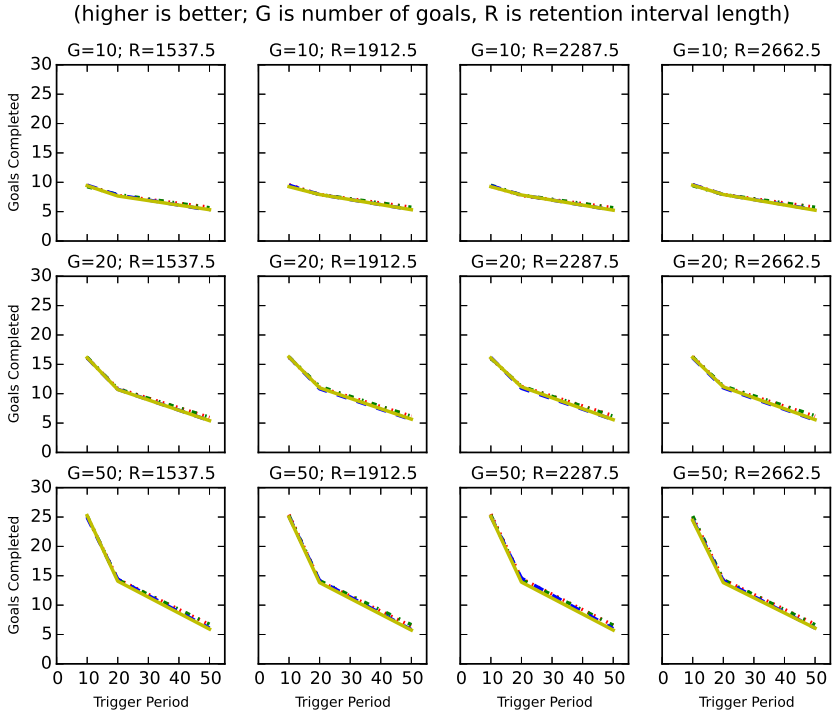

Figure 4.11: Absolute performance of the preemptive rehearsal strategy wrt. number of goals

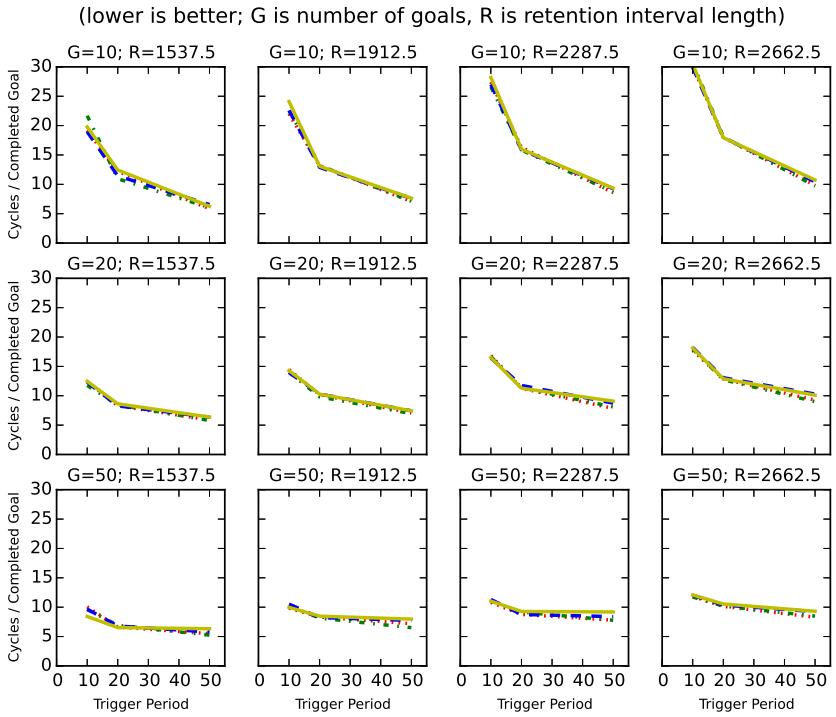

Figure 4.12: Efficiency of the preemptive rehearsal strategy wrt. number of goals

the columns of plots) has little effect, since the carrying capacity has already been reached at the shortest retention length. The effect of the number of goals, however, is slightly different. As the number of goals increase (looking down the rows of plots), the percentage performance of the strategy decreases, but *increase* when the absolute number of completed goals is considered, as shown in Figure 4.11. While it may seem counter-intuitive that the decrease in relative performance is only due to the increase in total number of goals, in fact the same phenomenon was seen in the preemptive retrieval strategy — there is an increase in the number of complete goals between ten and twenty total goals. This can be seen in Figure 4.6 at small trigger periods; the lower cluster of results are from environments with only ten goals, which the higher cluster is from environments with twenty or more goals. This implies that preemptive rehearsal strategies have a higher working memory goal carrying capacity than the retrieval strategy, and are therefore better suited for domains with larger numbers of goals. Returning to the idea of an equilibrium between goal maintenance operations and the decay of activation (see Section 4.5), this is likely because rehearsals lead to a larger increase in the activation of a goal than does a retrieval. As a result, the goal remains in working memory for a longer period, thus increasing the carrying capacity.

The same effect is reflected in the efficiency of the strategy, shown in Figure 4.12. The layout of this figure is the same as Figure 4.10, except with the number of goal maintenance operations

per completed goal on the y-axis. The effects of the carrying capacity can be seen by looking at the results as the number of goals increase (comparing between rows): the efficiency of the agent increases, meaning that at the lower number of goals, the strategy has not yet hit its carrying capacity.

Note that unlike the preemptive retrieval agent, it is not possible to use a separate count of the number of prospective goals to reduce the number of goal maintenance operations — the agent must rehearse its goals on every trigger. This inefficiency is due to the agent's inability to determine whether a goal will be forgotten without rehearsal, as the agent does not have access to the activation levels of its goals, and no additional information from long-term memory can provide that information.

4.8.3 Abstract Domain Summary

In the abstract domain, both retrieval and rehearsal preemptive strategies performed well when the timing trigger is sufficiently frequent. A key metric of both strategies it its carry capacity, which varies by the goal maintenance action and the trigger period; this determines how many goals can be kept in working memory at the same time, and therefore how many goals the agent could detect the target satisfaction of. For the same trigger period, the rehearsal strategy appears to have a higher carrying capacity, only exceeding it with fifty goals, as opposed to the retrieval strategy that exceeded its capacity with twenty goals. While the specific number of goals that each strategy can support depends on the decay rate and the forgetting threshold, the result that the timing-triggered preemptive rehearsal strategy can support more goals than the timing-triggered preemptive retrieval strategy is likely to hold. Both strategies, however, incur costs when either the agent fails to perceive the satisfaction of goal targets, or when goal targets are never objectively satisfied. Agent designers considering timing-triggered preemptive strategies should therefore consider a rehearsal strategy first, assuming that the targets of the prospective goals in the domain are likely to be satisfied.

4.9 Preemptive Strategy Summary

This chapter has presented what may seem to be the obvious solution to goal re-activation: preemptively ensuring goals are in working memory before the target is satisfied. These preemptive strategies are broadly classified by the trigger and by the goal maintenance action taken by the agent. From the mobile robot domain, the perception trigger appears more sensitive to environmental variations, and is better paired with a retrieval strategy — provided that the agent designer selects a predictive feature. From the abstract domain, it is shown that both rehearsal and retrieval strategies can be summarized by a "carrying capacity," which determines how many goals can be kept in

working memory at the same time; under this metric, rehearsal strategies outperform retrieval strategies.

While preemptive strategies perform well across retention intervals, their performance decreases when the trigger occurs less frequently. For the perception trigger, whether this occurs greatly depends on the feature selected as the trigger, which is highly dependent on the domain and on the insight of the agent designer. In theory, both the optimal timing trigger period and the most predictive feature for a perception trigger could be learned, but the latter in particular is a difficult problem. Additionally, agents with these strategies suffer when the targets of goals are never satisfied, as the agent will continue rehearsing or retrieving the goal, potentially for the remainder of the agent's lifetime. This is undesirable, and opens the question of whether a strategy exists that only consumes resources when the goal is relevant.

CHAPTER 5

A Spontaneous Strategy

Chapter 4 showed that preemptive strategies have many limitations: they present a difficult tradeoff between prospective task performance and resource consumption in terms of goal maintenance operators. Preemptive strategies require frequent goal maintenance operations to achieve a high level or performance, a cost that may continue for the remainder of the agent's lifetime if the target of a goal is never satisfied. There exists, however, a different approach to goal re-activation. Recall that preemptive strategies were developed to remove the first dependency in Figure 3.1, by ensuring that goals are in working memory before their targets are satisfied. This chapter presents a class of *spontaneous retrieval* strategies, which instead remove the second dependency in Figure 3.1, by allowing retrievals from long-term memory that do not require agent deliberation. This requires a significant addition to the capabilities of long-term memory, but as this chapter will demonstrate, it is a modification with multiple benefits.

Section 5.1 gives a brief overview of spontaneous retrieval strategy, highlighting the architectural modifications that must be made for the strategy to be implemented. By the criteria listed in Section 2.4, however, any architectural changes should be first evaluated generally. Section 5.2 therefore explains the design space of spontaneous retrieval mechanisms, including prior work. The implementation in Soar is detailed in Section 5.3, and is evaluated for generality in Section 5.4. With these results demonstrating that the architecture change provides additional benefits to agents, spontaneous retrieval is applied to the goal re-activation problem and evaluated in Section 5.5. The chapter closes with remarks in Section 5.6.

5.1 Overview

The second dependency from Figure 3.1 is the dependency of retrieval on deliberate agent action. This dependency is a result of the limitations of the current long-term memory mechanisms of cognitive architectures, which restrict cued retrieval as the sole method for extracting knowledge from long-term memory. Removing this dependency therefore requires a new mechanism: a retrieval

that does not involve agent deliberation, but is instead *architectural*. Just as the input buffer of Soar changes automatically to reflect new features of the environment, an architectural long-term memory retrieval would automatically update a buffer in working memory to reflect changes in long-term memory and working memory. More specifically, for the goal re-activation problem, a spontaneous retrieval mechanism would automatically retrieve a goal when the target of that goal is subjectively satisfied, thus allowing the goal to be pursued.

The crux of the spontaneous retrieval strategy is designing the matching and retrieval mechanism, such that it can not only solve the goal re-activation problem, but ideally also be useful to the agent as a whole. Although the idea of spontaneous retrieval is not new, and it has been implemented in other architectures — the Companions architecture has an "analogical tickler" that retrieves concepts analogical to the contents of the agent's working memory [21], and ACT-R has a spontaneous retrieval mechanism [35] — these mechanisms have not received widespread use. In fact, both mechanisms are often disabled by users of the architecture, as agent designers have found no use for memory retrievals at unpredictable times with unpredictable results. This emphasizes the importance of understanding the role of spontaneous retrieval in a cognitive architecture and the situations in which spontaneous retrieval can be beneficial. The following sections provide this understanding, before the use of spontaneous retrieval for goal re-activation is explored.

5.2 The Space of Spontaneous Retrieval Mechanisms

The goal of this section is to understand the role that spontaneous retrieval can play in a cognitive architecture, and the space of implementations that would still allow it to fulfill this role.

All long-term memory retrievals, deliberate or spontaneous, are aimed towards finding knowledge that is relevant to the agent's current task; in other words, it is a mechanism for *knowledge search* — the retrieval of knowledge that will aid the agent in its decision making [47]. This search is necessary because knowledge in long-term memory may not always be well organized and immediately accessible. This is especially true for a long-lived agent that acquires knowledge during its lifetime and must use that knowledge for multiple goals. From the viewpoint of knowledge search, the cues created by the agent in a deliberate cued retrieval exist to guide the search for the knowledge that would best aid the agent in the current situation. The rules that lead to a retrieval, however, specify not only how that knowledge can be found, but also the specific circumstances under which knowledge search should be initiated. And yet, rules can only match on working memory, which only contain a small subset of the knowledge of the agent; the majority of that knowledge exists in long-term memory. As a result of this incomplete information due to the partial observability of long-term memory, the search-guidance knowledge that rules provide may be suboptimal, or in the worst case, inaccurate or missing. In these situations, it would be beneficial to

have a mechanism that provides a more general heuristic for knowledge search, one that is robust to the agent's lack of search knowledge, even if it is not as precise as is possible with a deliberate cue.

Consider, for example, an agent in the real world, where it perceives an abundance of objects every decision cycle. The agent may have additional knowledge about many of these objects in long-term memory; however, only a small subset of this knowledge is useful for the agent's current decision making. The agent may incorrectly believe that knowledge about one object is more important than the other, however, or it may be unsure which of the many objects is important, if it even knows that the additional knowledge is useful in the first place. Spontaneous retrieval could help in this situation by using architecturally maintained information to heuristically retrieve knowledge into working memory. For example, if an object was previously used, perhaps it is more important; alternately, perhaps objects that are novel are more likely to provide breakthroughs in agent problem solving. The argument for spontaneous retrieval is not that the agent could not otherwise access that knowledge — in the worse case, the agent can iterating through all knowledge in long-term memory — but that this is impractical and inefficient. By supplementing the architecture with a well-designed spontaneous retrieval mechanism, the cost of a lack of memory search knowledge can be mitigated.

The goal of spontaneous retrieval as a mechanism to supplement deliberate retrieval has several implications for the design of the mechanism. These implications are further discussed in the following section, but the most important one is that the mechanism cannot depend on the agent's procedural knowledge to initiate the search nor to provide a cue to guide search, since it must be robust to the agent's lack of knowledge of both when to search and what to search for. The mechanism should therefore be automatic or *spontaneous* and *uncued*. There are at least three kinds of errors or gaps that spontaneous retrieval could provide. Using an example of a conversation about celebrities, spontaneous retrieval could, under the right circumstances, allow an agent to overcome its lack of knowledge of:

C1 What to search for in long-term memory, if the *relation* of the cue to the desired knowledge is unknown. For example, if the agent is asked for information on a celebrity, it may not know which aspects of the celebrity (e.g. their life story, their accomplishments, their habits, etc.) should be searched for in memory.

C2 What to search for in long-term memory, if the *cue* is unknown. In contrast to the example about not known whether to search for a celebrity's history or accomplishments, the lack of knowledge here is on *which* celebrity on which to perform a search.

C3 *When* to search long-term memory. In a conversation about celebrities, many celebrities are mentioned, some of whom the agent may have no additional knowledge about. The ideal agent should not expend resources on these celebrities, and spontaneous retrieval may provide a signal

that no additional knowledge exists in memory. Although this is related to C2 above, it is different in that the correct time to retrieve may not be a function of the agent's perceptions at all, but a function of whether knowledge exists in long-term memory.

The problem of missing or incorrect heuristics for knowledge search presents additional constraints on spontaneous retrieval beyond the need for it to be automatic and uncued. Within these constraints and guided by the theoretical benefits, there is a space for variations in implementation.

5.2.1 How does spontaneous retrieval integrate with a cognitive architecture?

The main consideration in this section is that spontaneous retrieval is not the only retrieval mechanism, and it must work in conjunction with deliberate retrieval. Since spontaneous retrieval is supposed to supplement in deliberate retrieval, one implementation is to emphasize spontaneously-retrieved elements, and allow them to overwrite whatever was deliberately retrieved. On the other hand, since deliberate retrievals involve the use of agent procedural knowledge of some kind, perhaps spontaneous retrievals should only occur when no deliberate retrievals are made. Both of these implementations assume that spontaneous and deliberate retrievals use the same buffer in working memory. Although the architecture designer must shoulder the burden of determining a priority, using the same buffer for both types of retrievals reflects how the mechanisms both serve the same purpose in knowledge search; the agent can be agnostic and apathetic as to the source of the retrieval. The alternative is to create a separate buffer for spontaneous retrieval, which would require that the agent decide when knowledge from each buffer should be used. Since both models of integrating spontaneous retrieval into an architecture have merit, neither dominates the other in this analysis.

5.2.2 When do spontaneous retrievals occur?

Much as preemptive strategies are parameterized by when goals are retrieved, an architecture spontaneous retrieval mechanism has a parameter of when retrievals occur. A simple solution would be that spontaneous retrieval occur every decision cycle, or more generally, every n decision cycles; this corresponds to the timing trigger used for preemptive strategies, a term adopted here for the sake of consistency. An alternate solution would be for spontaneous retrievals to occur when particular features are represented in working memory. While there may not be universal percepts across all domains upon which knowledge should be spontaneous retrieved, it is not unimaginable that spontaneous retrieval occurs only when the agent is in particular metacognitive states, say, when the

agent is confused. This reflects the role of spontaneous retrieval as a heuristic for when deliberate retrievals are inadequate.

5.2.3 Which element is spontaneously retrieved?

A final consideration is how the retrieved element is relevant to the agent's current situation. While spontaneous retrieval could randomly select elements to retrieve into working memory, the likelihood that such an element will be useful is small, considering that the majority of long-term memory is irrelevant at any given time. Luckily, the requirement that a retrieved element is relevant is one that is shared with deliberate retrieval, and thus existing solutions can be re-used. In particular, one popular solution to the problem of retrieval relevance is *spreading activation*, where long-term memory elements related to the elements in working memory get a boost in activation (and are therefore more likely to be retrieved). Although known under other terms, a similar idea lies at the heart of PageRank, where a link from an important page (a relation to elements in working memory) increases the rank of a different site (increases the likelihood that a different element will be retrieved) [26]. More generally, spreading activation is a graph centrality measure that takes into account a set of context nodes, which also suggests the possibility of using other such centrality measures as the retrieval bias [48]. Unfortunately, no standard spreading activation algorithm exists; implementations differ on which elements cause spreading activation to occur, the elements to which activation spreads, and the size of the boost in activation for those elements. Spreading activation also raises concern with the potential for positive feedback loops, where the same elements all spread activation to each other, such that no other elements are spontaneously retrieved [36].

Other mechanisms for biasing retrievals exist. The Companions cognitive architecture, for example, biases deliberate long-term memory retrieval by similarity in graph structure [20]; similar metrics are used in case-based reasoning systems [50]. More generally, spontaneous retrieval can be cast as determining which long-term memory element is the most useful, given the agent state. In fact, there has been much recent work in psychology that casts memory retrieval as a Bayesian process that selects the element with the highest probability of usefulness [27, 58]. The original rational analysis of memory and the invention of base-level activation were partially inspired by this, and assume a power law distribution of perceptual features [2]. To our knowledge, however, there has yet to be a fully Bayesian model of memory that works well with the semantic knowledge common in cognitive architectures.

Even if activation is used as a bias, there remain multiple choices for which element is returned in a retrieval. Drawing inspiration from action selection in reinforcement learning, some possibilities include: to always select the most highly activated element; to be epsilon-greedy, where the most highly activated element is selected except for some small probability of a random selection; or

to use a softmax function where the probability of an element being retrieved is proportional to its activation value. Again, these options also exist for deliberate cued retrieval, except that with spontaneous retrieval, there is no hard constraint for the element to match a cue; in fact, ignoring the cue from deliberate cued retrievals leads naturally to a retrieval bias for spontaneous retrieval.

5.3 Implementation

Given this space of possible spontaneous retrieval implementations, the one used in this thesis focuses on maximizing the similarities between deliberate and spontaneous retrieval, while highlighting the unique aspects of the latter. Thus, like the current deliberate retrieval mechanism in Soar, spontaneous retrieval selects the most highly activated element, and also shares the same buffer in working memory as deliberate retrieval. Spontaneous retrieval is also limited to only occurring when no deliberate retrievals are in progress, but otherwise occurs periodically according to a frequency parameter. To ensure that retrievals are heuristically relevant, spreading activation is used to boost the base-level activation of long-term memory elements. In particular, whenever an element is stored into long-term memory or is retrieved into working memory, it's graph neighbors, and their graph neighbors, and so on up to a parameterized distance d, all receive an activation boost of the same amount. This implementation of spreading activation is a strict superset of the agent's base-level activation mechanism, since if $d = 0$ — that is, if no neighbors are boosted — then only the element that is stored or retrieved receives the boost, which is the current behavior.

Beyond the basic mechanism, there are three constraints on spontaneous retrieval to prevent positive feedback loops. First, spontaneously retrieved elements do not receive a boost in activation, to prevent the same element from continually having the highest activation. Second, spontaneous retrieval only returns elements not currently in working memory. The one exception is where the previously spontaneously retrieved element remains the one with the highest activation, in which case the element is not replaced. Finally, to prevent positive feedback loops through cycles in memory, elements are only boosted once per storage or retrieval.

In terms of computational efficiency, calculating the effects of spreading activation is expensive, and has been an ongoing research problem for the cognitive architecture community [11, 15]. Since this implementation of spreading activation is aimed only to demonstrate the benefits of spontaneous retrieval, it is straightforward and not optimized. Specifically, Soar's long-term memory is implemented as an SQLite database, which uses indices to allow efficient queries; all long-term memory elements are indexed by their activation value for fast deliberate retrieval [13]. For spontaneous retrieval, an element is selected by iterating through the index until an element is found that is not already in working memory. This allows spontaneous retrieval to take advantage of any efficiency improvements for spreading activation, while remaining independent of the

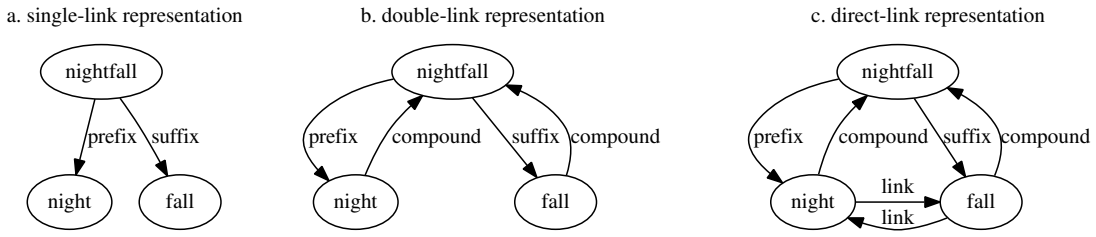

Figure 5.1: Different representations of knowledge in the Missing Link domain

implementation details of spreading activation. The data presented below show the costs of these two processes separately.

5.4 Evaluation for Spontaneous Retrieval Mechanism Generality

The goal of this evaluation is to explore whether spontaneous retrieval is generally beneficial to intelligent agents, by demonstrating whether it allows agents to overcome various types of missing knowledge (C1, C2, and C3). Although such an evaluation of the mechanism is not strictly necessary for its inclusion into a cognitive architecture, it nonetheless highlights some of the strengths and tradeoffs made.

5.4.1 The Missing Link Domain

In order to evaluate the benefits of spontaneous retrieval, we use the Missing Link word puzzle as a domain. In this puzzle, the agent is given three words (*stems*) as clues (for example, `fall`, `fort`, and `time`), and must provide a fourth word (the *link*; in this case, `night`) that forms compound words or phrases with all three stems (`nightfall`, `fortnight`, `nighttime`). The puzzles used in this evaluation are gathered from the Unix word list, using compound words that can be completely divided into two shorter words; a total of 550 compound words formed by 195 stems are used, with each stem being used in an average of 2.8 compound words. In the single-link representation (explained below), the compound words and stems form a connected bipartite graph of diameter 12.

It should be noted that this puzzle is the same as the Remote Associates Test [42], and it was chosen for some of the same reasons. In addition to the large amount of search necessary to find the solution, the puzzle also allows the agent's knowledge and the important features of the environment to be transparently manipulated. Search guidance in this domain depends on the connections between the stems and their compound words, which in turn depends on the source of knowledge. A

generic dictionary may provide only the component stems of a compound word, while a knowledge base optimized for the Missing Link puzzle would have connections directly between the stems. While the existence of the latter connections may make the puzzle trivial, spontaneous retrieval may help when only less specialized representations are available. The first variation of the Missing Link domain models this difference in knowledge with three different long-term memory representations of the dictionary. The three representations, also depicted in Figure 5.1, are:

- The *single*-link representation only has links from the compound words to its prefixes and suffixes.
- The *double*-link representation also has links from the stems to all their compound words.
- The *direct*-link representation also has links between the prefixes and suffixes of any compound word.

Two additional variations of the Missing Link domain are used. In the first variation, Subset Missing Link, the agent is additionally presented with *distractors*, other words that do not form compound words with the missing link. In the above example, the additional words `foot` and `man` might also be presented to the agent. The addition of distractors models how only a subset of features in the environment may be relevant to the agent's goals.

The second variation modifies the Missing Link domain on a multi-puzzle level. In this Probabilistic Missing Link domain, each puzzle has only some probability of being solvable (that is, that a link exists between the three stems). As with the first variation, this models how only a subset of features are relevant, except in the temporal dimension.

These three domain variations correspond to three cases where spontaneous retrieval is likely to be beneficial. In the Missing Link domain with the single-link representation, the agent does not know the relation of the stem to its compound word (that is, whether it is a prefix or a suffix), and must therefore search memory with both (C1). For Subset Missing Link, the agent does not know which portion of its percepts are important and should be used as the cue to search memory (C2), while for Probabilistic Missing Link, the agent does not have knowledge of whether searching at this time will be beneficial, or if it will just consume resources (C3).

Since the agent can always use brute-force search to find the solution, the metric for this evaluation is not the number of puzzles that the agent solves, but the amount of computation needed to do so, in both decision cycles and real time.

5.4.2 Missing Link Agent Design

In each puzzle, the environment presents the stems as strings. Both agents with and without spontaneous retrieval must first retrieve the long-term memory representations of the stems. To

correctly solve a Missing Link puzzle, an agent without spontaneous retrieval must retrieve all compound words that contain each stem, as well as the other stem that forms those compound words, before finally checking if a missing link exists that is shared by the compound words of all three stems. The differences in representation only determine how quickly this can be done. With the single-link representation, the agent must do all of these steps. With the double-link representation, the agent does not need to retrieve the compound words, as they are retrieved together with the representation of the clue word. With the direct-link representation, the agent can directly check for a missing link. This is true in the two domain variations as well: the stems must be expanded to detect which three have a missing link, just as they must be expanded to determine whether a solution exists for any three stems.

A different search process is possible with spontaneous retrieval. When the memory element representing each stem is retrieved, activation spreads to its compound words and to the other prefix/suffix. Since the missing link is the only word that receives three activation boosts, it has the highest activation and is the first element to be spontaneously retrieved. The agent can then verify that the spontaneously retrieved word forms compound words with all three stems. The benefit of spontaneous retrieval is therefore that the stem–compound-word–stem connections do not need to be deliberately explored. Alternately, spontaneous retrieval can be thought of as using spreading activation to specify a search criteria based on the *graph topology* around the cue elements, one that is not constrained to return elements that must be direct neighbors of the cue.

Deliberate and spontaneous retrievals should not be thought of as separate strategies, however, but as two mechanisms working towards the same goal. Spontaneous retrieval is returning elements that the agent could deliberately retrieve, given the correct sequence of retrievals with the correct cues — since in this domain deliberate retrieval is only used for brute-force search, spontaneous retrieval allows the agent to skip ahead in its reasoning. However, there are also cases where spontaneous retrieval may mislead the agent; for the Subset Missing Link domain in particular, the retrieval of the distractors could be interleaved with the retrieval of the real stems, allowing time for the activation of the missing link to decay. This may cause a more-recently-boosted element to be returned by spontaneous retrieval instead, which would fail the verification. In this case, the agent must resort to the deliberate strategy, at least until a different element is spontaneous retrieved.

It should be noted that the spontaneity of spontaneous retrieval is not strictly necessary for this domain. Since the puzzles are episodic, one can imagine the agent deliberately retrieving the solution when a new puzzle is presented, using a mechanism that returns that most highly activated element. This agent would do better than a spontaneous retrieval agent, since it is taking advantage of the episodic nature of the environment. To demonstrate the benefits of spontaneous retrieval, however, the agent uses the deliberate, brute-force search, but takes advantage of any spontaneously retrieved knowledge; that is, spontaneous retrieval provides usable knowledge without being asked

to do so. The focus is on spontaneous retrieval complementing deliberate retrieval, especially when brute force becomes more expensive in the Subset and Probabilistic Missing Link variations.

Since deliberate retrievals are used as a fallback, agents must be designed so that they can move from deliberate retrievals to spontaneous retrievals and back, and be able to integrate information from both. This can be achieved by conditioning the processing of retrieved memory elements not by the mechanism that retrieved it, but by the information that it represents. For example, in the Missing Link domain, if the retrieved element is a compound word that contains a stem, it should be stored so that the other prefix/suffix can be retrieved later; on the other hand, if the retrieved element is not a compound word connected to the clues, it may be the missing link, and it should be checked as the answer. The key is that these behaviors depend only on the agent state and how the retrieved element should be used, but not on whether the retrieval was deliberate or spontaneous. Rules that process retrievals only match on the features of the retrieved element, and not on which mechanism retrieved the element or why it is retrieved; in fact, neither mechanism provides the latter. By separating the processing of information from its source, the agent can effectively use both the knowledge that it knows it needs as well as any spontaneously retrieved shortcuts.

5.4.3 Results

All results in this section are averaged over 100 puzzles. Since each puzzle is independent, the agent is reset between puzzles to negate any recency and frequency effects of activation. The knowledge of compound words and their stems is loaded into the agent's long-term memory before each puzzle. Spreading activation is limited to a distance of two, the distance necessary to reach the solution word (from a stem to its compound words, then from the compound words to the missing link); we briefly discuss the effects of alternate settings of this parameter in the conclusion. All agents are written as deliberate agents, with the processing of retrieved elements then modified to be agnostic to the retrieval mechanism. In fact, the "deliberate" and "spontaneous" agents have the exact same rules, with the only difference being that the architecture provides spontaneous retrievals for the "spontaneous" agent. No effort was made to space deliberate retrievals so that spontaneous retrievals could occur.

5.4.3.1 Efficiency of Spontaneous Retrieval

To evaluate the efficiency of spontaneous retrieval, we instrumented Soar to measure the amount of time spent calculating the effects of spreading activation, versus that of selecting the most highly activated element in long-term memory. Averaged over 100 puzzles, an agent with the single-link representation takes an average of 350.7 milliseconds to solve a puzzle, of which 335.2 milliseconds is spent updating activation values, and only 0.85 milliseconds for spontaneous retrieval. Although

this mechanism goes above the reactivity threshold of 50 milliseconds, much of this is due to spreading activation, which is known to be a computationally expensive process [11]. There is on going work to reduce the cost of spreading activation by approximating its results [24], which is outside the scope of this thesis. Nonetheless, spontaneous retrieval itself requires little computation, meaning that as a new mechanism it meets the reactivity threshold.

Since spreading activation also affects the outcome of deliberate retrievals, it is included in the results below for both the deliberate and spontaneous agents.

5.4.3.2 Missing Link

The purpose of this experiment is to demonstrate that spontaneous retrieval can overcome a lack of knowledge of how percepts relate to the desired knowledge. In the worst case, a brute-force search would require the agent to iterate through all possible relations, although in this domain there are only two possible relations (i.e. prefix or suffix). The average number of decision cycles and amount of real time needed to solve a puzzle by different agents are shown in the table below.

Knowledge	**Decision Cycles**		**Real Time (msec)**	
Representation	Delib.	Spon.	Delib.	Spon.
Single	56.1	24.5	2631.6	350.7
Double	60.9	60.9	1167.7	1173.1
Direct	12.0	12.0	267.8	268.3

Table 5.1: Timing of spontaneous retrieval wrt. different knowledge representations

For the single-link representation, the spontaneous retrieval agent takes half the number of decisions to complete the task, and an even smaller proportion of real time. As hypothesized during the discussion of agent design, this is because activation spreads to the missing link, which spontaneous retrieval then puts into working memory. This eliminates many decision cycles during which the deliberate approach is exhaustively expanding the stems and the compound words. The difference in real time is more dramatic because of the cost of spreading activation — since there are fewer retrievals, there are fewer boosts to base-level activation and therefore fewer spreads. This difference would be smaller with a more efficient spreading activation algorithm.

Spontaneous retrieval has little effect on either metric for the other two representations, although the source of this lack of differentiation is different. For the direct-link representation, the missing link could be immediately determined after the stem elements are retrieved, removing the need to search long-term memory. In this case, spontaneous retrieval is unnecessary. For the double-link representation, the majority of agent processing is from retrieving the other prefix/suffix of the compound words; there are no gaps between these uses of long-term memory, leaving no opportunity for spontaneous retrievals to occur. This raises questions about balancing deliberate and spontaneous

retrievals, as well as whether spontaneous retrievals should be placed in a different buffer; these issues are addressed in the conclusion to this section. Regardless, although spontaneous retrievals has no benefits when these knowledge representations are used, it also incurs minimal cost.

The results across all three knowledge representations agree with the original insight that spontaneous retrieval supplements deliberate retrieval when the agent lacks search-guidance knowledge. The more optimized the knowledge base for a task — as more connections are added between stems and their links — the more effective deliberate retrieval becomes, and the less spontaneous retrieval can offer. On the other hand, spontaneous retrieval can provide the agent with relevant knowledge when brute-force search is necessary, greatly reducing processing time. Since the single-link representation best highlights the difference between deliberate and spontaneous retrievals, it is the only representation used for the remaining results.

5.4.3.3 Subset Missing Link

The presence of distractors in the Subset Missing Link domain means that the agent must retrieve more compound words before the missing link can be identified. In terms of search, this increases the number of initial states (memory elements), thereby increasing the size of the search "frontier." To be explicit, distractors are presented in addition to the three stem words, so the agent is presented with five clue words for a puzzle with two distractors. Results from this domain variation are shown in the table below.

Number of	**Decision Cycles**		**Real Time (msec)**	
Distractors	Delib.	Spon.	Delib.	Spon.
0	56.1	24.5	2631.6	350.7
1	65.1	26.6	3048.6	341.4
2	70.9	28.2	5752.1	319.4
3	80.8	32.8	3679.1	613.5
4	84.5	40.0	17591.5	840.0

Table 5.2: Timing of spontaneous retrieval wrt. number of distractors

Although both agents with and without spontaneous retrieval require more resources to deal with distractors, the agent with spontaneous retrieval requires less additional resources to do so. For the spontaneous retrieval agent, this growth is due to the extra decision cycles necessary to retrieve the long-term memory representations of the distractor clues. The deliberate retrieval agent, however, also needs to search for all the compound words for those distractors, hence the larger increase. These results suggest that, for domains where the relevant percepts are not obvious, search-guidance knowledge is doubly important, as search grows exponentially with the number of irrelevant percepts. This is true with spontaneous retrieval as well, but using activation as a heuristic

reduces the exponential.

Despite this increase in resource consumption, spontaneous retrieval allows the agent to sidestep the selection of a cue for deliberate search, and continues to reduce the amount of computation necessary for the agent to solve the puzzle.

5.4.3.4 Probabilistic Missing Link

The goal of this experiment is to show that spontaneous retrieval can overcome the lack of knowledge of when to retrieve from long-term memory. Unlike the previous experiments, here the role of spontaneously retrieved elements is not only to provide the correct answer, but also to be a heuristic for whether an answer exists at all. This difference is reflected in a change in how spontaneous retrievals are used. The agent with spontaneous retrieval does not continue to retrieve words until the solution is found. Instead, when a new puzzle is presented to the agent, it simply attempts to verify whether the first spontaneously retrieved element is the missing link. The results of two different verification procedures are shown: the more costly method (V1) retrieves all compound words of the potential solution, while the more efficient method (V2) directly checks for compound words formed by the potential solution and the stems. For the agent without spontaneous retrieval, no such heuristic for solvability is available, and it must therefore deliberately retrieve all compound words before giving up.

Probability	**Decision Cycles**		**Real Time (msec)**		
Solvable	Delib.	Spon. V1	Delib.	Spon. V1	Spon. V2
1.0	56.1	24.5	2631.6	350.7	100.5
0.9	56.1	25.9	2602.9	442.4	159.6
0.8	55.5	27.6	2464.0	628.4	165.4
0.7	56.0	29.8	2555.8	744.8	178.0
0.6	55.9	30.5	2459.8	777.9	179.9
0.5	53.9	31.9	2099.2	861.9	186.5
0.4	51.4	35.3	1677.9	1135.3	218.3
0.3	50.3	38.1	1574.2	1450.2	247.0
0.2	49.5	40.0	1400.6	1633.9	263.4
0.1	49.0	41.2	1324.1	1768.4	264.3
0.0	47.8	42.7	1099.5	1858.6	316.8

Table 5.3: Timing of spontaneous retrieval wrt. proportion of unsolvable puzzles

The results for the single-link agents are shown in the table above. Although the agent with spontaneous retrieval can often identify solvable puzzles in fewer decisions than agents without spontaneous retrieval, whether it can do so in less real time depends on the verification method.

The deliberate agent gives up sooner on unsolvable puzzles, since those stems tend to have fewer compound words (as it would otherwise increase the probability that a missing link exists). No such trend exists for the spontaneously retrieved element: a word that has the most neighbors would have the highest activation, regardless of whether the puzzle is solvable. This is exacerbated by the quadratic time necessary to complete the naive method of iterating through all compound words (V1), which only takes linear time for the more efficient method (V2). As spontaneous retrieval is less and less likely to return the solution, the verification of the solution could take more time than iterating through the compound words, leading to the tradeoff seen in the results for the naive verification method.

Assuming the cost of verification is low, these results show that spontaneous retrieval can cheaply provide the agent with knowledge, even if the agent is uncertain whether that knowledge exists.

5.4.4 Conclusion on the Generality of Spontaneous Retrieval

In all three variations of the Missing Link domain, the use of spontaneous retrieval leads to agents that more quickly complete their tasks. Spontaneous retrieval provides a short-cut through brute-force search; allows the agent to efficiently ignore distractors; and indicates whether relevant knowledge exists in long-term memory. Both the spontaneity and the spreading activation bias are necessary for these benefits: spreading activation reduces the time needed for search, while the spontaneity allows the agent to search less frequently. Together, these results provide evidence for the hypothesis that spontaneous retrieval can supplement deliberate retrieval as a heuristic to overcome a lack of knowledge of when to search and what to search for.

One objection to these results is that the spontaneity of the spontaneous retrievals was not necessary. Since it was the spreading activation that correctly biased memory retrievals, the agent could have done just as well with only a spreading activation mechanism, and performing deliberate retrievals for every puzzle. In the general case, however, the agent designer would have to explicitly write a rule that performs retrievals on every decision cycle. The advantage of an architectural spontaneous retrieval mechanism is not only that it is more efficient, but also that the fallback heuristic can be employed by other agent designers as well.

While the goal of this evaluation is not to fully explore the space of spontaneous retrieval implementations, we acknowledge that it's possible for alternate implementations to perform differently in the same domain. If the spontaneous retrieval had its own buffer instead of sharing the one with deliberate retrieval, for example, even the results with the doubly-linked representation would likely be an improvement over that of an agent without spontaneous retrieval at all, since the current agent's retrieval is bottlenecked by the buffer. At the same time, a different selection bias

for spontaneous retrieval would likely lead to poorer performance, and we admit that the domain was chosen to match the spreading activation bias. Since spontaneous retrieval does not alter the behavior of deliberate retrieval, however, outside of the computational cost there are no drawbacks to having such a mechanism in the architecture, and the agent could fall back on deliberate retrieval if necessary. As long as the verification of the retrieved answer is faster than iterating through long-term memory to find it, spontaneous retrieval is at best neutral, but often beneficial to agent performance.

That cognitive architecture research has continued without spontaneous retrieval thus far suggests that there is a category of tasks that the field has not tackled — those where the agent is missing procedural knowledge for how to search for declarative knowledge in long-term memory. Given that researchers generally attempt to maximize the performance of the agents by providing it with all necessary knowledge, perhaps it is not surprising that such tasks have not been explored. Nonetheless, it is in such tasks where spontaneous retrieval shows its benefits. Goal re-activation is one such task, and the performance of a spontaneous-retrieval-dependent strategy is evaluated in the next section.

5.5 Evaluation for Goal Re-Activation

The previous section showed that spontaneous retrieval can benefit agents. This section evaluates the same mechanism in goal re-activation in the abstract domain presented in Chapter 4. As with preemptive strategies, the goal of this evaluation is two fold: first, to explore the effectiveness of this strategy with respect to changes in environments with different parameter settings; and second, to find scenarios where spontaneous retrieval fails as a goal re-activation strategy.

The design of the spontaneous retrieval goal re-activation agent must first be explained. Without spreading activation, environmental percepts entering working memory had no direct effect on long-term memory. With spreading activation, when a percept enters working memory, it causes activation to spread from its long-term memory representation. Crucially, goals that the agent has stored into long-term memory also receive a boost in activation, since their features correspond to what the agent perceives. If the resulting activation of the goal is sufficiently high (in particular, if it becomes the most highly activated element), then it will be spontaneously retrieved into working memory.

Throughout the retention interval then, the agent perceives different random percepts from the environment, and various long-term memory elements are spontaneously retrieved. No deliberate retrievals are made at all by the agent. If a goal is spontaneously retrieved, its target is checked against the percepts of the agent for satisfaction. If the target is satisfied, a rule matches and proposes the option of pursuing the goal to the agent; if the target is not satisfied, then the goal is temporarily

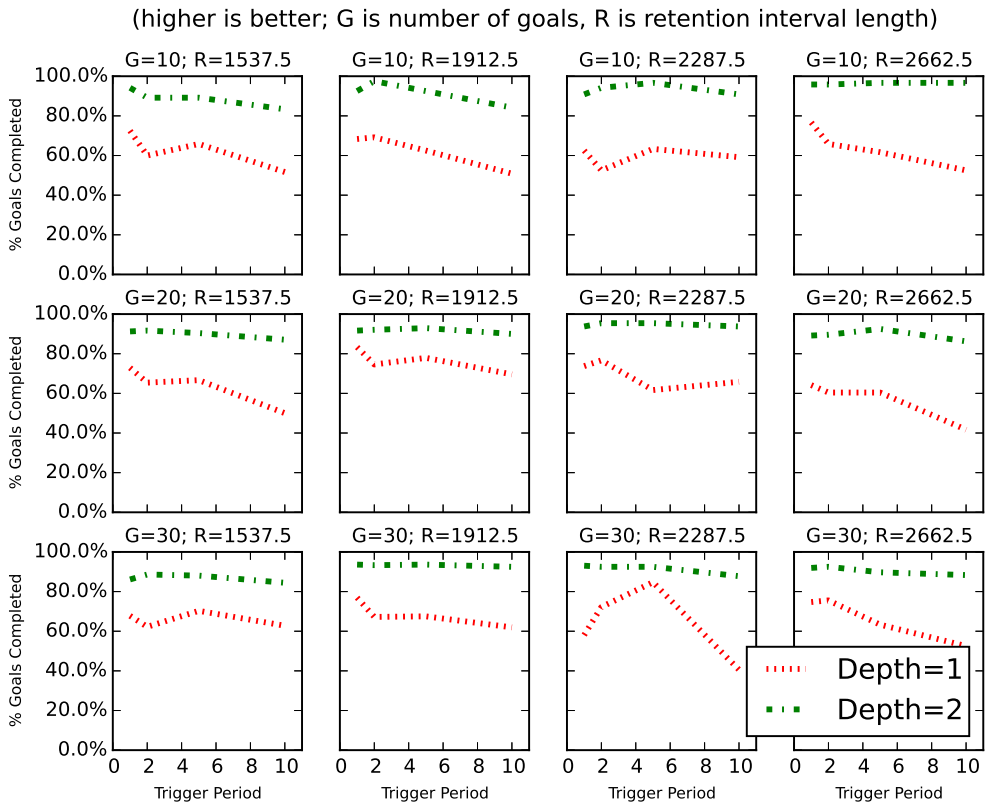

Figure 5.2: Performance of the spontaneous retrieval strategy wrt. number of goals

stored in working memory, until its working memory activation decays and is removed again.

If instead the spontaneously retrieved memory element is not a goal, the agent ignores whatever is retrieved. In general, however, other retrieved non-goal elements may still contribute to the agent's reasoning; while the retrieval of a non-goal may be considered *interference* for goal re-activation, it may in fact be beneficial for the agent overall (as demonstrated in the Missing Link domains). While no additional knowledge exists in the agents tested in the abstract domain, the agent also does not restrict its retrievals to goals. The agents must therefore deal with irrelevant retrievals, although whether this occurs with realistic frequency depends on the real-world domain used for comparison.

Spontaneous retrieval by itself only has a single parameter: the frequency with which long-term memory elements are spontaneously retrieved into working memory. Additionally, the choice of the bias function — that is, spreading activation — has a single parameter: a depth limit on how far activation spreads.

5.5.1 Abstract Domain

Figure 5.2 shows the performance of the spontaneous retrieval strategy, with the percentage of goals completed on the y-axis and the trigger period (that is, the number of decisions between each spontaneous retrieval) on the x-axis. Each subplot contains the results for a particular environmental parameter setting with the total number of goals (G) and the retention interval (R) indicated in the title. Each subplot also contains two lines, which indicate the depth limit of spreading activation.

Across all subplots, spontaneous retrieval maintains a high level of performance. The trigger period of spontaneous retrieval has a small effect, which is to be expected; the further apart the retrievals that occur, the more likely that a goal will not be retrieved in time. The parameter with the largest effect is the spreading activation depth limit (comparing between lines in the same plot), with a limit of two being more effective. We believe this is because more goals are retrieved with a larger limit (since more memory elements are boosted, leading to more turnover in terms of the element with the highest activation); this leads to more goals being kept in working memory, increasing the probably that the goal can be compared against the target percepts at the initiation stage. The evidence from varying the goal encounter rate below also supports this, while a more theoretical analysis of the effect of the depth limit on spreading is in 5.5.2.

On the other hand, neither the number of goals nor the length of the retention interval (comparing across subplots) has a large effect. More goals decrease the performance of the agent, but the effect is slight. This can be understood by considering how spreading activation is a measure of graph centrality. For goal re-activation, only the relevant goal is at the center of the target features; additional goals become peripheral nodes on the graph, and do not affect the outcome of spreading activation. At the same time, it's possible for other goals to be boosted simply by random percepts. Since the activation of an element is a summary of the history of the element; by only looking at the activation value, it is impossible to differentiate between elements whose activation were frequently boosted by a small amount, versus elements whose activation were recently boosted by a large amount. When spreading activation is added to the equation, this means that the most highly activated element may be a result of what occurred in the past instead of a result of the agent's current situation. Thus, a random sequence of percepts that have a large overlap with the target of a goal can cause the activation of that goal to be boosted by a large amount, and when the target of a different goal is satisfied, it would not be spontaneously retrieved despite the spreading activation it receives. This effect depends on the specific sequence of random percepts and does not occur frequently, hence the small effect.

The effect of the length of the retention interval can be explained in the same way. Long-term memory elements are only affected by time through the decay (and occasional boost) of their activation. In this case, the boost from spreading activation is stronger than the decay of activation over time, allowing the relevant goal to be retrieved. This is not without a failure condition, however,

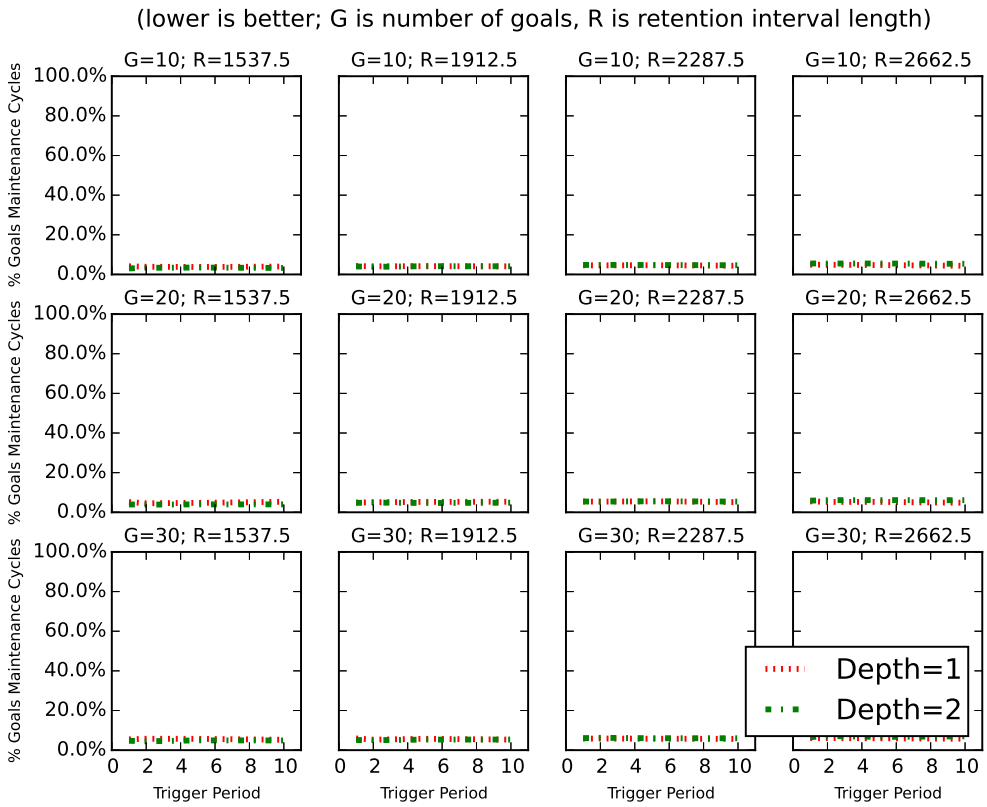

Figure 5.3: Resource usage of the spontaneous retrieval strategy wrt. number of goals

which we explore in Section 5.5.3.

Before we present the cost of the spontaneous retrieval strategy, we must first explain why the cost is non-zero. Since spontaneous retrieval relies entirely on an architectural mechanism, there is no agent deliberation involved in retrieving the goal into working memory. Furthermore, rule matching is automatic, meaning that if a relevant goal is spontaneously retrieved, an operator is automatically proposed. By the definition of computational resource in Section 4.4, spontaneous retrieval in fact has zero cost. However, this metric does not take into account how goals may be spontaneously retrieved when their targets are *not* satisfied, and are temporarily stored in working memory. In essence, spontaneous retrieval can also play a *preemptive* role, where goals are retrieved before they become relevant. It is this small computation used by the agent that is captured in Figure 5.3, which shows the number of decision cycles spent on goal maintenance (y-axis), as a function of the trigger period (x-axis). Again, each subplot contains the results for a particular environmental parameter setting with the total number of goals (G) and the retention interval (R) indicated in the title. As the plots show, the cost of spontaneous retrieval is minimal, remaining at under 10% across all parameter settings.

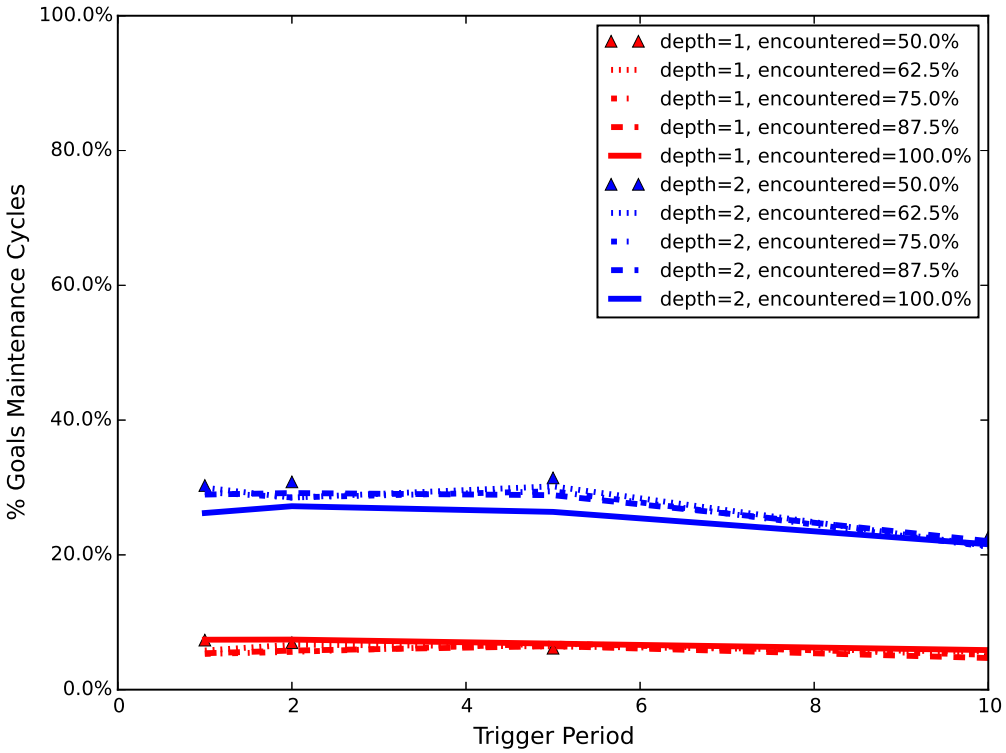

Figure 5.4: Resource usage of the spontaneous retrieval strategy wrt. goal encounter rate

Finally, Figure 5.4 shows the percentage of decision cycles spent on goal maintenance when the goal encounter rate is varied. The parameter has little effect in general, although unlike in Figure 5.3, strategies with a deeper limit on spreading activation leads to more goal maintenance operations. Recall that the goal maintenance operations only occur when a goal is spontaneously retrieved and its targets are *not* satisfied. Since no new element is retrieved if the most highly activated element has not changed, this results suggests that a deeper spreading activation limit leads to more turnovers, or equivalently, leads to more changes in which long-term memory element has the highest activation. This may also be the reason why the strategy with this parameter performs better (as show in Figure 5.2): since different goals are more frequently brought into working memory, they have a higher chance of remaining in working memory when their targets are satisfied.

In summary, a strategy that uses spontaneous retrieval is capable of high performance with low cost to the agent. The parameter with the biggest effect on the performance of this strategy is the limit on how far activation could spread, which we explore in the next section.

5.5.2 Encoding Specificity

The evaluation of spontaneous retrieval has, thus far, involved goals that are directly linked to the perceptual features that form their target; the goal is a parent node, with many child nodes that represent the features. This implies that the goal target is always directly perceived, and is never an abstraction over perceptual features. For example, consider the milk-buying task. While a valid target for the goal is for the agent to be standing outside a specific grocery store, the target could also be a more general description of being outside any store that sells milk. This is represented by additional memory elements between the goal and the perceptual features, elements that represent the semantic concept of a generic grocery store. This representation, however, causes problems with preemptive strategies — the strategy must now not only retrieve the goal, but also any intermediate abstractions that connect the goal to the perceptual features. Spontaneous retrieval through spreading activation avoids this problem, as the intermediate elements do not need to be retrieved — it is sufficient for activation to spread through those elements to the goal, such that the goal is retrieved directly.

This section explores the performance of spontaneous retrieval when the goal target is part of a feature hierarchy; an example of such a hierarchy is show in Figure 5.5. Formally, the hierarchy is a layered directed graph, where each layer only has incoming edges from the layer above it, and only has outgoing edges to the layer below it. Specifically, long-term memory is a recognition hierarchy or a feature hierarchy, where higher-level features are composed of a fixed number of lower level features. For example, a dining room consists of tables and chairs, which in turn consists of legs and surfaces, and so on. Such hierarchies are common both in perception (for example, in deep

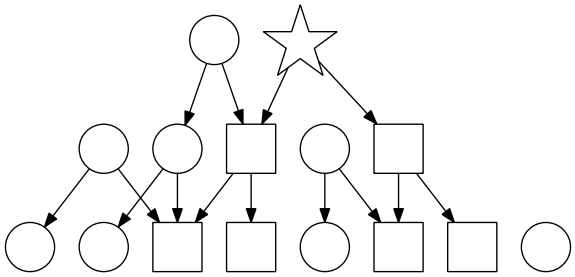

Figure 5.5: A randomly generated knowledge hierarchy, with goals

belief networks) and in logic (for example, ontologies), meaning the results of this experiment may be broadly applicable. In the lowest level of the hierarchy are the perceptual features, while other levels represent abstract concepts built from perceptual features. In the figure, the star-shaped node is a goal, and the square nodes its descendant target features. At the initiation stage, the agent only perceives the square nodes at the lowest level, but it must infer that the goal target is satisfied.

In theory, the spontaneous retrieval strategy can still retrieve the goal when its perceptual-level targets are perceived, since activation spreads through the intermediate features to provide the goal with activation boosts. This thesis is interested in two changes to the agent, to explore whether they would improve the performance of the spontaneous retrieval strategy. First, the agent can use elaboration rules to automatically bring the abstract features into working memory. In this case, the agent has rules that are conditioned on the child features; when the rule fires, the parent feature is retrieved into working memory. Since the goal has more descendants in working memory than other long-term memory elements, it receives a larger spreading activation boost, and is more likely to be spontaneously retrieved.

The second change is for the agent to encode the goal not with its abstract targets, but with its perceptual targets. This corresponds to the idea of *encoding specificity* in psychology, where it refers to the phenomenon that the more the encoding of a piece of knowledge is representative of the context in which it is needed, the more likely a person is able to recall that piece of knowledge [61]. In prospective memory, the principle of encoding specificity has led to the development of *implementation intentions*, where people think of goals as "When I get back to my office and sit in my chair and look at my monitor, I will send an email to my colleague" instead of simply, "I will send an email to my colleague" [41]. The agent modifies its representation of the goal such that it is more likely to be retrieved spontaneously. For the implementation of spontaneous retrieval in Soar, this means altering the connections among long-term memory elements to increase the amount of activation that spreads from the target percepts to the goal. Specifically, instead of storing only the features of the goal, the agent also stores the goal with its lower-level features. For

example, for the goal in Figure 5.5, this means that the goal (the star-shaped node) is stored with the square nodes on the lowest level in addition to those in the intermediate level. This encoding means the goal is now connected to the knowledge hierarchy at a lower level than it would be otherwise. Since the goal is often abstract and at the top of the knowledge hierarchy, we denote the specificity of an encoding by how many levels below the goal it is linked to; in this example, the encoding specificity would be 2.

5.5.2.1 Theoretical Analysis

Given this definition of encoding specificity, we perform initial analysis to determine whether a goal could be spontaneously retrieved. For goal at knowledge level g, elaboration rules that create features up to level e, and a spreading depth of d, the goal must be encoded at specificity level s that satisfies the following relationship:

$$d \geq g - e - s + 1 \tag{5.1}$$

That is, the spreading depth must be able to reach from the highest-level elaborated features to goal target features (plus an extra level to spread from the target to the goal itself). Note that the agent cannot complete any goals when $d = 0$, since the goal would never receive an activation boost from spreading (since no spreading occurs). We can additionally calculate the maximum number of boosts a goal will receive, assuming the knowledge hierarchy has branching factor b:

$$\sum_{i=\max(1,\, s-d+1,\, g-e)}^{\min(g,\, s+d-1)} b^i \tag{5.2}$$

That is, every feature in the levels indicated by the index would boost the goal, a number that is exponential in the branching factor. This classification allows us to group agents across a large parameter space for comparison.

5.5.2.2 Extending the Abstract Domain

In order to evaluate whether spontaneous retrieval and spreading activation can be used with abstract goal targets, the abstract domain is extended to include feature hierarchies. This is a strict expansion of the domain; the goals and features in the previous version of the domain can be considered a two layer hierarchy, where all features are on the bottom and only goals are on the top. This extension allows abstract features to be added in between the layers.

The following algorithm is used to generate random hierarchies; its pseudocode is show in Algorithm 5.1. This algorithm takes as input the desired breadth (w) of the hierarchy at a particular level (h), the branching factor (b), as well as some probability of abstraction (p). First, a single

```
Require: global hierarchy
  function CREATEHIERARCHY(w, h, b, p)
    while |{nodes at level h}| < w do
      level ← 0
      hierarchy.CREATENODE(level)
      while RANDOM( ) < p do
        level ← level + 1
        n ← CREATENODE(level)
        children ← hierarchy.GETNUMNODESATLEVEL(b, level − 1)
        n.ADDCHILDREN(children)
```

Algorithm 5.1: The hierarchy generation algorithm

node at the lowest level (call this level 0) is created. Every time a new node at level n is created, including the first node, there is probability p that a new node is created at level $n+1$. If this new node is created, then b nodes from level n are chosen at random to be its children. The abstraction process is then recursively applied, potentially creating a new node at level $n+2$, $n+3$, and so on. When the abstraction process stops, then the algorithm creates a new level 0 node. This process repeats until level h has w nodes, at which point the algorithm completes.

5.5.2.3 Results

As a reminder, this evaluation is interested in how the agent can best re-activation goals when they are part of a feature hierarchy. For this experiment, the desired breadth of the hierarchy (w) is set equal to the number of goals (P1). The level of the goals (h) is varied from 1 to 3 (perceptual features are at level 0), while the branching factor is fixed at $b = 7$ and the probability of abstraction at $p = 0.7$. These values were chosen to keep long-term memory relatively small, since the cost of spreading activation grows with the size of long-term memory. Additionally, the level of features that elaboration rules created (e) is also varied from 0 (that is, no elaboration rules are used) to 3. The level of encoding specificity (s) of the agent is also varied from 1 to 3.

A number of parameter settings within the parameter space of this experiment fail in completing any goals. These are settings where the goal is at least two steps away from the elaborated features — for example, if elaborations provide features of level 3 and the goal target features are encoded at level 4, thus requiring a two-level spread from elaboration to target features to the goal. Equivalently, this is when $e + s < g$, or where the right hand side of Equation 5.1 is two or more. In these cases, the lower-level features are activated more frequently, causing them to have higher activation than the goal and preventing the goal from being retrieved. This suggests that the activation boosting of a goal is not sufficient to guarantee its completion.

All other parameter settings allow the agent to complete goals. The parameter that is most

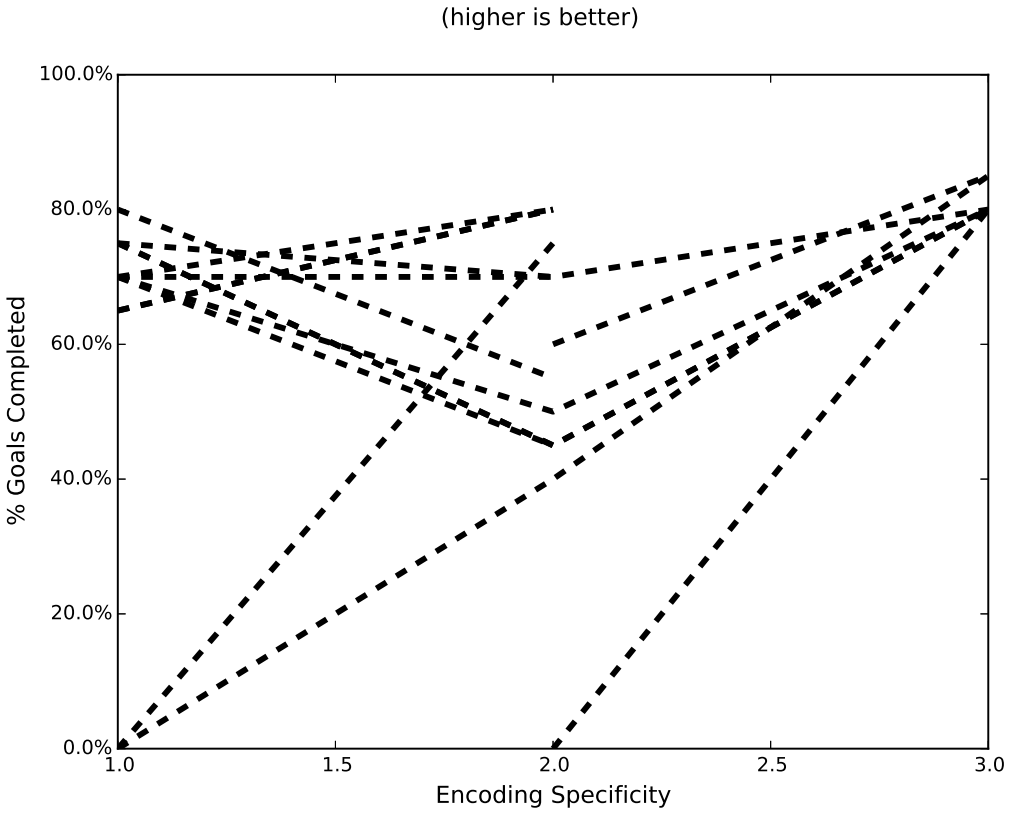

Figure 5.6: Performance of the spontaneous retrieval strategy wrt. encoding specificity

correlated with higher performance is the specificity of the encoding: most increases in encoding specificity lead to an increase in performance, although this trend does not always hold. Figure 5.6 shows the results across the entire parameter space, with the percentage of goals completed on the y-axis and the encoding specificity on the x-axis. Each line in the plot shows an agent with all other parameters fixed except for encoding specificity.

Surprisingly, the effect of encoding specificity is not uniform, with there frequently being a decrease in performance for middle values of specificity. We suspect this is because of a tradeoff between the boost in activation due to spreading, and the amount of activation in other goals. Consider how high-level features are only rarely brought into working memory, since they require many low-level features to be present at the same time. This means that if goals are only linked to high level features (that is, the encoding is not specific), the goals will not receive much spreading activation). The reverse is also true: specifically encoded goals retrieve more boosts in activation, since low level features are frequently perceived; at the initiation stage, the many low-level features will all be present, boosting the goal. With a medium level of specificity, however, other goals are not infrequently boosted, while the initiation boost for a goal is also not particularly large. Since other goals may also have high activation, the initiating goal may not always be spontaneously retrieved — hence the decrease in performance.

Neither the level of elaboration nor the depth limit for spreading activation have a uniform effect on the agent's performance. Although these parameters also affect the number of times a goal is boosted, the problem is that they also boost the activation of all *other* goals in addition to the goal that is being initiated. As with the low-level features from above, it is a high *relative* activation that allows a goal to be spontaneously retrieved. More specific encodings provide a large enough boost at initiation for the single goal to be retrieved, while these other parameters do not. In retrospect, this is not surprising: more specifically encoded goals are linked to more features, which means that there are more opportunities for the activation of the goal to be boosted.

Overall, these results agree with the psychology literature: the best goal encoding should match both environmental parameters (such as how abstract the goal is) and agent parameters (such as the limit to spreading activation), but that more specific encodings in general lead to better performance.

5.5.3 Retention Interval (Revisited)

In the initial experiment in Section 5.5.1, the retention interval had no effect on the performance of the spontaneous retrieval strategy. Learning from the experiments with encoding specificity, however, we suspect that this is due to the "density" of percepts to goals. The features that an agent perceives during the retention interval are randomly selected, and may coincidentally be one of the target features of a goal; that goal would then receive a small boost in activation. Since all percepts

are equally likely, all goals would receive roughly equal numbers of activation boosts, meaning no single goal is particularly highly activated (or particularly un-activated either).

This idea can be framed as one of *resting activation* — activation that a goal would have during the retention interval, which is determined by an equilibrium formed by the increase in activation due to spreading from random input and the decrease in activation due to decay. Changes in either would move the resting activation value; if the decay rate is increased, or if there is less activation from random input (as would be the case if the input did not contain target features at all), the resting activation value would decrease. Again, it is not the resting activation that directly determines the performance of the agent, but the relative activation of a goal at initiation time; this is why the decay rate has no effect, since it affects the activation of all goals. Conversely, if a goal has low resting activation compared to other goals, the activation spread from its target features may not be sufficient to make it the most activated element, preventing its spontaneous retrieval.

To explore this possibility, we modified the abstract domain such that during the retention stage, the target features of a single goal (the *blocked* goal) are never presented to the agent until initiation. The percepts that contain these features are re-generated until there is no overlap; this loop occurs in the INTERPOLATION function in Algorithm 4.1. We call this the Leave One Out percept sequence, as opposed to the Normal percept sequence. For that goal, there should be much less activation boosts from spreading as compared to other goals, leading to a lower resting activation level. In this case, a longer retention length (as activation decays after the goal is initially stored) may cause the spontaneous retrieval of the goal to fail, preventing the agent from pursuing the goal.

As expected, the Leave One Out sequence results in much more variance in the activations of goals. The least activated goal in a Normal percept sequence is 1.44 standard deviations away from the mean, while with a Leave One Out sequence, the least activated goal is 12.1 standard deviations away (the standard deviations were calculated without the left-out goal). This leads to the goals not being retrieved for completion as the retention interval increases, as shown in Figure 5.7. The significant decrease in performance in the Leave One Out sequence is due to the filtered features, which not only affects the activation of the blocked goal, but also of other goals that share features with the blocked goal. Since these other goals also receive fewer activation boosts, they are also unlikely to be retrieved spontaneously, leading to a failure in their pursuit. At the same time, the remaining goals have roughly equal resting activation, and thus the length of the retention interval has no effect at this level.

This result could be interpreted in two ways. On one hand, for random percepts, using spontaneous retrieval for prospective memory suffers no degradations in performance, which suggests that it may be preferable to monitoring strategies. On the other hand, goals with target features that are never encountered outside of initiation are unlikely to be retrieved with the current mechanism. We do not know of any studies that look at the baseline frequencies of goal target features, nor of

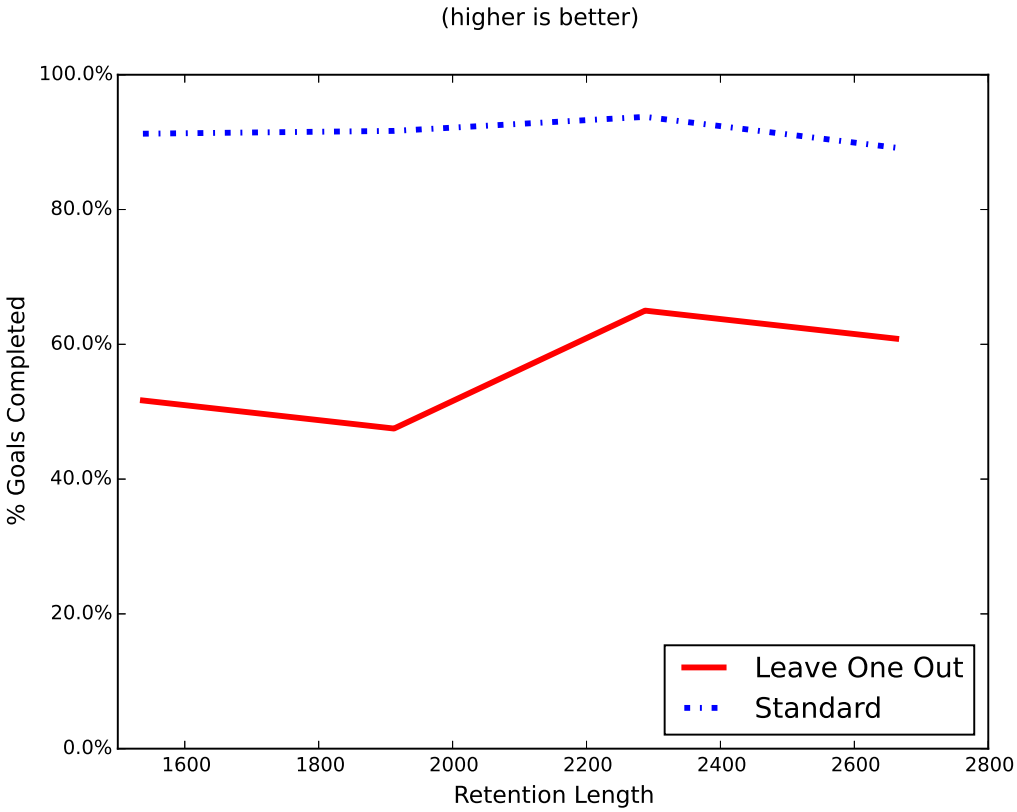

Figure 5.7: Performance of the spontaneous retrieval strategy wrt. different percept sequences

studies that examine human prospective memory performance where performing the goal requires satisfying multiple disjoint target features. It is possible that human performance exhibits similar patterns under such situations; alternately, a better model may be a hybrid strategy where occasional monitoring-like retrievals prevent the activation of goals from dropping too low.

5.6 Spontaneous Retrieval Strategy Summary

This chapter presented the spontaneous retrieval strategy, where goals are automatically retrieved into working memory. In implementing this strategy, the new architectural mechanism of spontaneous retrieval is created and evaluated in the Missing Link domain, showing that in certain cases it allows the agent to speed up long-term memory search. Applying spontaneous retrieval to goal re-activation in the abstract domain, spontaneous retrieval shows consistent performance across environmental parameters such as the number of goals, the retention interval. With spreading activation, the target satisfaction of more abstract goals could also be detected, although a specific encoding remains the parameter with better performance. The one drawback of spontaneous retrieval is a dependency on the temporal dynamics of the domain, namely, that if a goal does not receive activation boosts over a prolonged period, it would lead to its failure to be spontaneously retrieved.

For agent designers considering a spontaneous retrieval strategy for prospective memory, the spreading depth has the most impact on the results. The optimal spreading depth depends on multiple factors, including the structure of long-term memory, the structure of goal targets, as well as the percepts from the environment. The experiments in this chapter present the best-case scenario for all three, where there does not exist extraneous clusters in long-term memory; where the goal is centrally-located from the goal target features; and where percepts do not themselves form clusters in long-term memory. As a result, a relatively simple spreading activation scheme allows the agent to achieve high performance. More exploration in the space of implementations for both spreading activation and spontaneous retrieval are necessary for when these factors are not as ideal. Since spontaneous retrieval is not yet a common mechanism in cognitive architectures, it would also be useful to understand where the mechanism lies in the space outlined in Section 5.2, and what choices were made to ensure the mechanism leads to better task performance. Generally, the parameters that affect spontaneous retrieval are not well understood, and this may be an impediment to quickly developing agents that use spontaneous retrieval.

CHAPTER 6

A Hybrid Strategy and Strategy Comparisons

The previous chapters explored two different strategies for goal re-activation separately; however, the strategies are not mutually exclusive. This chapter briefly explores whether their combination can lead to better performance or efficiency in Section 6.1. The three classes of strategies — preemptive, spontaneous retrieval, and hybrid — are then compared under different environmental conditions in Section 6.2: Given an environment and task with a particular set of properties, which goal re-activation strategy should be used?

6.1 A Hybrid Strategy

The previous two chapters presented preemptive strategies and spontaneous retrieval strategies, each of which has failure conditions. Preemptive strategies require a large number of goal maintenance operations, especially when the targets of goals are never satisfied, while spontaneous retrieval strategies fail for a particular goal if its activation in long-term memory is too low compared to other items in memory. These particular weaknesses appear complimentary. The problem of boosting long-term memory activation is similar to the problem of preventing goals from being forgotten from working memory; they both require periodic access to the memory element in question, if only on different levels of the memory hierarchy (as described in Section 2.3). At the same time, spontaneous retrieval brings forgotten goals into working memory, which may allow for a reduction in the frequency of preemptive triggers. A hybrid strategy may allow agents to overcome both issues.

This chapter is specifically concerned with the use of a hybrid strategy for individual goals, and not with agents that adaptively employ either preemptive or spontaneous retrieval strategies depending on the goal. The latter is already possible, as each strategy does not prevent the use of the other; this is the common definition of "hybridization" in polling/interrupt schemes [16]. The insight here is that the strategies are also not mutually exclusive at a more basic level, that they in fact compensate for each other's weaknesses.

Hybrid strategies have not previously existed, since spontaneous retrieval strategies are new to this thesis. If we consider spontaneous retrieval as the baseline, however, we can see similarities between hybrid strategies and strategies that use *external memory*. The problem we are trying to solve in spontaneous retrieval is the possibility of a leave-one-out scenario, where one goal has much lower activation than other elements in long-term memory. One solution is to boost the activation of goals periodically. Hybrid strategies do this by performing deliberate retrievals, while the use of external memory achieves the same effect by manipulating the environment. For example, the prototypical use of external memory is tying a string around one's finger, with the agent forming a link between the string and some goal. This works because every time the string is felt or seen, the agent would be reminded of the goal, thus boosting its long-term memory activation and preventing the activation from dropping too low. While the use of external memory is not within the scope of this thesis, preemptive deliberate retrievals may nonetheless accomplish the same effect, through preventing the leave-one-out scenario from occurring.

6.1.1 Evaluation

The hybrid strategy evaluated in this section is simple: spontaneous retrieval is enabled while the agent runs, with the rules for both preemptive and spontaneous strategies as part of the agent. Since spontaneous retrievals only occur when no deliberate retrievals are occurring, there are no conflicts as to which strategy takes precedence when the target of a goal is satisfied. To be specific, during the retention interval, spontaneous retrievals of goals can occur. If the retrieval occurs when the goal target is satisfied, the agent has the option of pursuing the goal. Otherwise, the goal is stored in working memory, and treated as though it has been deliberately retrieved. Simultaneously, periodic deliberate retrievals would also bring goals into working memory, which are also maintained; if the target of any of these goals become satisfied, rules propose an operator for the agent to pursue the goal. Importantly for the hybrid strategy, the deliberate retrieval of a goal leads to a boost in the goal's long-term memory activation, thus making its future spontaneous retrieval more likely.

The goal of this evaluation is to explore whether a hybrid strategy can lead to agents that have better goal re-activation performance with little additional goal maintenance cost. For this reason, the results here focus on the cases where previous strategies have failed: where preemptive strategies have low efficiency, and where spontaneous retrieval fails to retrieve goals. A middling range of timing trigger period is chosen to demonstrate the possibility of performance improvement. For spontaneous retrieval, since the goal is to demonstrate additional improvement, the best set of parameters is used (trigger period of 1 decision cycle and a spreading depth of 2).

The performance of this strategy in the abstract domain is shown in Figure 6.1, which shows the percentage of goals completed on the y-axis and the total number of goals on the x-axis.

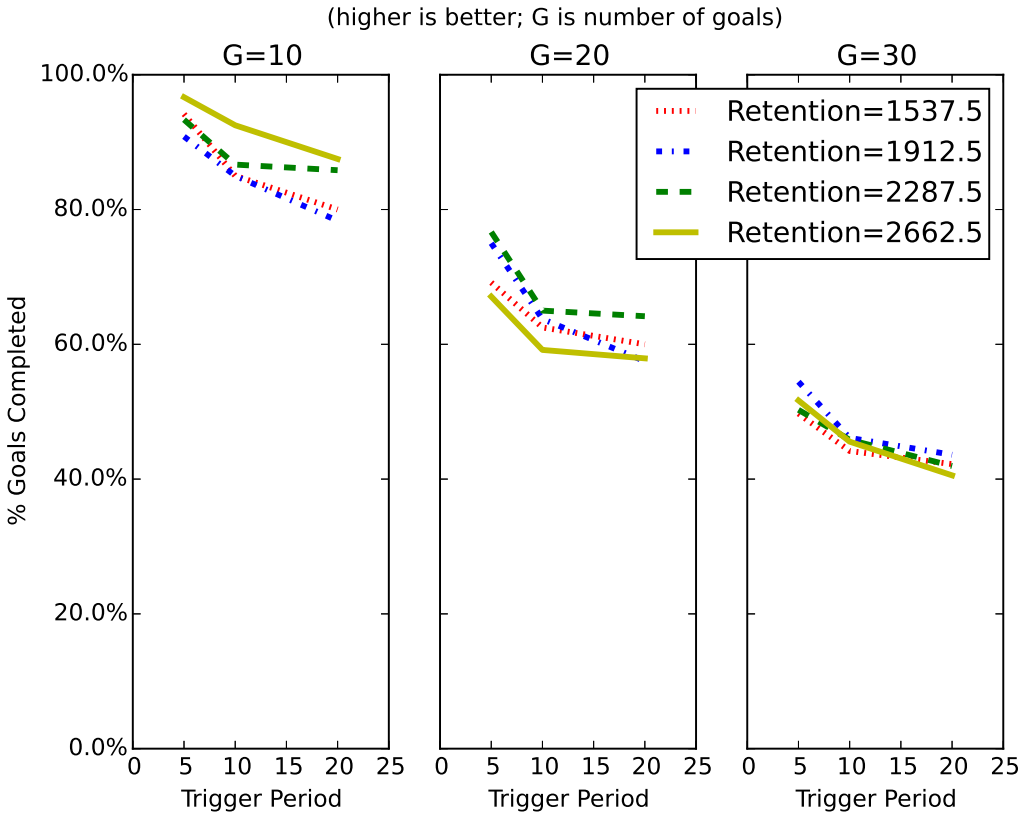

Figure 6.1: Performance of the Hybrid Strategy wrt. Number of Goals

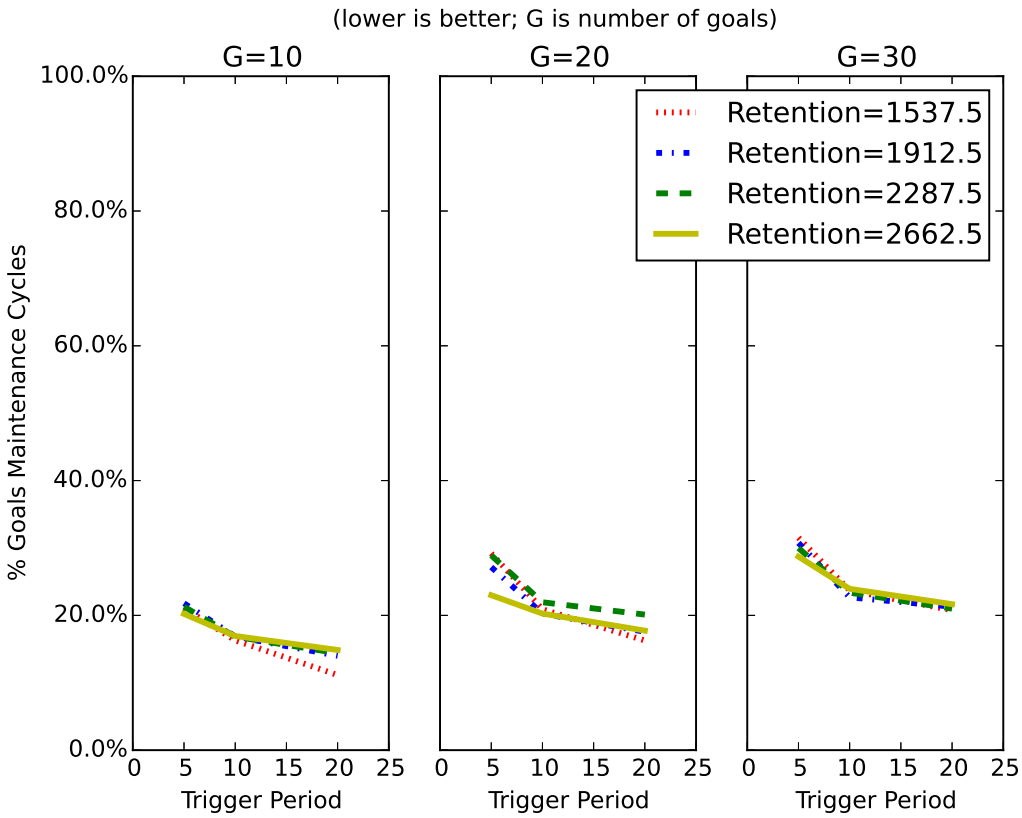

Figure 6.2: Resource Usage of the Hybrid Strategy wrt. Number of Goals

Each subplot shows an environment with a different number of goals, while each line represents an environment with a different retention interval. These results show that there is a decline in performance as the number of goals is increased (comparing between subplots), and that the length of the retention interval has a noisy but overall neutral effect on performance (comparing lines in the same subplot). Compared to the results from the equivalent preemptive retrieval strategy from Figure 4.5, however, the hybrid strategy performs better, although still not as well as the spontaneous retrieval strategy (Figure 5.2). Whereas the performance of a pure preemptive retrieval strategy with a trigger period of ten decision cycles could only complete 10% of goals, the equivalent hybrid strategy completes about 80%.

Figure 6.2 shows the number of goal maintenance operations the agent performed on the y-axis, with the layout otherwise the same as the previous figure. As a reminder, this number includes both the deliberate retrievals from the preemptive strategy, as well as the operator to temporarily store spontaneously retrieved goals into working memory. Comparing across subplots, the number of goal maintenance operations increases as the number of goals is increased. This growth neither appears in preemptive retrieval strategies (Figure 4.7) nor in spontaneous retrieval strategy (Figure

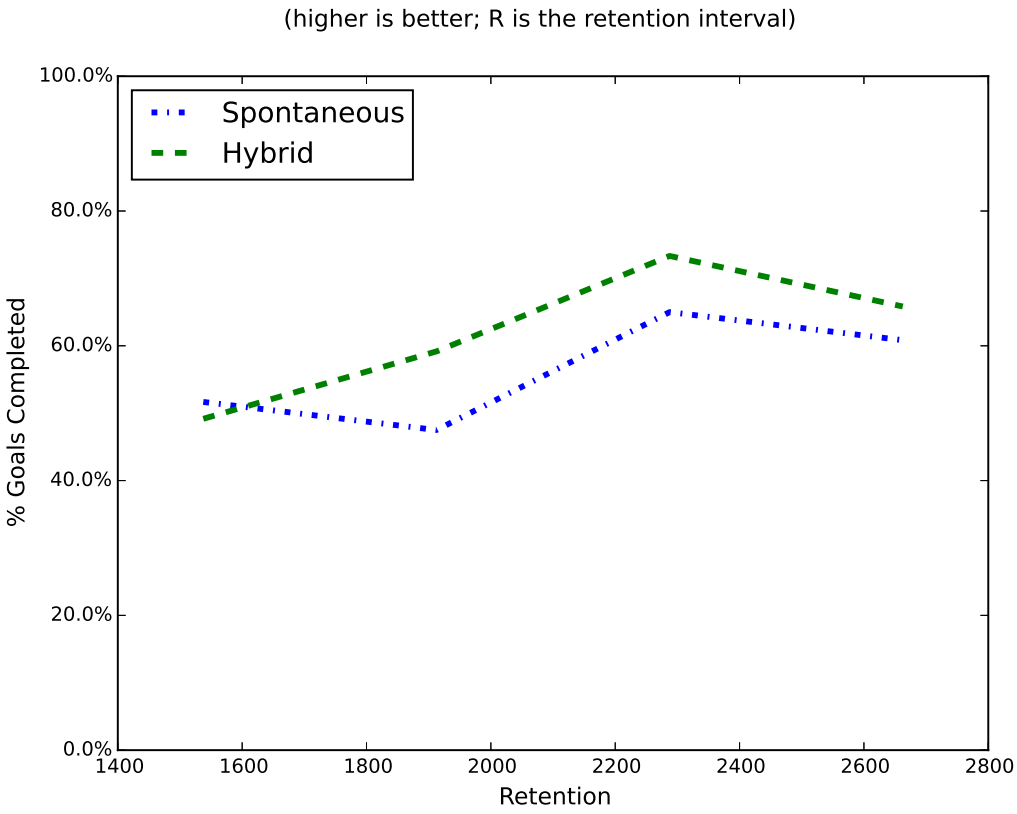

Figure 6.3: Performance of the Hybrid Strategy on the Leave One Out sequence

5.3) alone. A deeper analysis of the data suggests that the increase in goal maintenance is due to the *storage* of spontaneously retrieved goals, indicating that more goals are spontaneously retrieved by the hybrid strategy than by the spontaneous retrieval strategy alone. This is what we would expect if the preemptive retrievals are preventing the activation of goals from decaying, since more goals can become the most highly activated long-term memory element (and thus be spontaneous retrieved). Since the agent must perform deliberate retrievals in addition to storing spontaneously retrieved goals, however, the hybrid strategy requires more goal maintenance operations than either preemptive or spontaneous retrieval strategies alone. This raises the question of whether the increase in performance is a worthwhile tradeoff for an increase in cost, a question we tackle in Section 6.2.

Summarizing the results from the standard abstract domain, hybrid strategies greatly improve performance over preemptive retrieval strategies, although at the cost of requiring more goal maintenance operations as the retention interval increases. However, This section has yet to show that hybrid strategies can also help spontaneous retrieval strategies overcome the lack of periodic activation to goals. A comparison of hybrid and spontaneous retrieval strategies from the Leave One Out percept sequence is shown in Figure 6.3, with the hybrid strategy in green and the spontaneous

retrieval strategy in blue. Recall that this sequence differs from the one normally used in that the target percepts of one of the goals never occur during its retention interval, leading to a decay in the goal's long-term memory activation and potentially preventing it from being retrieved. The results show that the hybrid strategy marginally increases performance, although it does not recover its performance in the environment where no features are blocked at all. This implies that the preemptive retrievals only boost the activation of a subset of goals, but the ones with the lowest activation continue to fail to be retrieved spontaneously.

6.1.2 Hybrid Strategy Summary

Our goal for developing a hybrid strategy was to use the respective strengths of the preemptive and spontaneous retrieval strategies to cover the other strategy's flaws. While the hybrid strategy does out perform each basic strategy at their weak points, it also comes with additional costs — namely that the agent must perform more goal maintenance operations. However, these results alone cannot tell agent designs which of the three strategies to use. To do so, we need to examine how each strategy trade off efficiency with performance — in other words, which strategy lies on the Pareto frontier.

6.2 Strategy Comparison

With the evaluation of the different strategies proposed in this thesis, a meta-strategy can be developed for which strategy should be used given a particular set of environmental properties. Since the two metrics, performance and efficiency, cannot be directly traded off, the goal is instead to find the *Pareto-optimal* strategies for each environmental parameter, where it is impossible for each strategy to gain in performance or efficiency without sacrificing the other metric. On the left side of Table 6.1 are all the environment parameters we explored in this thesis, while on the right are the parameter settings of the strategies that lie on the Pareto frontier for that environment. We used a table to show these results instead of the standard Pareto plot because there are non-trivial number of environments, and because the strategies are tightly clustered, making it difficult to distinguish between them.

This table guides agent designers as to which strategies are worth consideration depending on environmental characteristics. While the exact numeric values of the number of goals and length of the retention interval may not apply to other domains, we can observe some general trends in the table. First, the environments in which preemptive strategies could be optimal strategies are cases where there are fewer goals to complete. We understand this through the idea of a carrying capacity — once the number of goals is above that capacity, preemptive strategies are inefficient

# Goals	Retention Length	Pareto-Optimal Strategies
10	1537.5	Preemptive Retrieval (trigger=1) Preemptive Rehearsal (trigger=10, rehearsals=50) Preemptive Rehearsal (trigger=50, rehearsals=10) Spontaneous (trigger=1, depth=2)
10	1912.5	Preemptive Rehearsal (trigger=50, rehearsals=10) Spontaneous (trigger=2, depth=2)
10	2287.5	Preemptive Rehearsal (trigger=50, rehearsals=10) Spontaneous (trigger=5, depth=2)
10	2662.5	Preemptive Rehearsal (trigger=50, rehearsals=20) Spontaneous (trigger=10, depth=2)
20	1537.5	Preemptive Rehearsal (trigger=50, rehearsals=20) Spontaneous (trigger=2, depth=2)
20	1912.5	Preemptive Rehearsal (trigger=50, rehearsals=20) Spontaneous (trigger=5, depth=2)
20	2287.5	Preemptive Rehearsal (trigger=50, rehearsals=10) Spontaneous (trigger=2, depth=2)
20	2662.5	Preemptive Rehearsal (trigger=50, rehearsals=20) Spontaneous (trigger=5, depth=2)
30	1537.5	Preemptive Rehearsal (trigger=50, rehearsals=20) Spontaneous (trigger=2, depth=2)
30	1912.5	Spontaneous (trigger=1, depth=2) Spontaneous (trigger=2, depth=2)
30	2287.5	Spontaneous (trigger=1, depth=2)
30	2662.5	Spontaneous (trigger=1, depth=2) Spontaneous (trigger=2, depth=2)

Table 6.1: Pareto-optimal strategies for goal re-activation wrt. number of goals and retention interval length

and perform poorly, thus fail to remain on the Pareto frontier. As a result, spontaneous retrieval strategies dominate when preemptive strategies fail. We expect this trend to generalize beyond the abstract domain, as the carrying capacity phenomenon is independent of the environment.

A second observable trend is that in the majority of cases where preemptive strategies are Pareto-optimal, it is the rehearsal strategy that is preferred. The only exception to this is when the agent has few goals with short retention intervals, when the retrieval strategy is also Pareto-optimal. A closer inspection of the data reveals that the preemptive retrieval strategy out performs the other three strategies for that environment parameter setting. From this, we conclude that the preemptive retrieval strategy is in general too resource intensive, although it works well for limited environments.

Third, we note that the retention interval has no effect on the choice of strategies. This is a result of how neither strategy is affected by the retention interval, its effects small compared to those of other parameters.

A fourth trend is not obvious from the table. For all of the environmental parameter settings, the strategies are clustered into two tight groups. One group, often comprising the spontaneous retrieval strategy, emphasizes the performance of the agent (that is, most of its prospective goals are completed). The other group, in contrast, is extremely efficient for whatever portion of goals it could complete. We suspect that in the abstract domain, spontaneous retrieval has such high performance (and a relatively low cost) that only a drastic tradeoff is sufficient for other strategies to be on the Pareto frontier. This is congruent with how many of the preemptive strategies have a long trigger period, as this would lower the cost of the strategy sufficiently.

The results of a similar analysis performed on environments with varying rates of encountered goals is shown in Table 6.2. In this case, all the Pareto-optimal strategies are version of spontaneous retrieval, which is unsurprising given the ongoing costs of preemptive strategies. Similar to the results above, most environment parameter settings only result in strategies that optimize only for performance (here represented by spontaneous retrieval strategies with a trigger period of 1) and those that optimize only for efficiency (strategies with a trigger period of 5).

Finally, we note that no hybrid strategy is Pareto-optimal in any environment parameter setting — while it does increase the performance of preemptive strategies, it still does not compare to the performance of spontaneous retrieval. In fact, the majority of parameters only have two Pareto-optimal strategies. Since there are only two metrics being compared, it is necessarily the case that one of these optimal strategies optimizes for performance and the other for efficiency. By examining the parameters of the strategies, it is obvious that the preemptive strategy is "optimizing" for efficiency by the long periods between rehearsals. We conclude that preemptive strategies are the preferable choice only when the constraints on agent resources are extreme, as otherwise spontaneous retrieval strategies provide high performance at a relatively low cost.

Goal Encounter Rate	Pareto-Optimal Strategies
0.5	Spontaneous (trigger=2, depth=2)
0.625	Spontaneous (trigger=1, depth=2) Spontaneous (trigger=5, depth=2)
0.75	Spontaneous (trigger=1, depth=2) Spontaneous (trigger=5, depth=2)
0.875	Spontaneous (trigger=1, depth=2) Spontaneous (trigger=5, depth=2)
1.0	Spontaneous (trigger=1, depth=2) Spontaneous (trigger=5, depth=2)

Table 6.2: Pareto-optimal strategies for goal re-activation wrt. goal encounter rate

6.3 Goal Re-activation Strategies Summary

From this analysis, we see that preemptive strategies are only Pareto-optimal when the number of goals is small. This result is significant in that preemptive strategies can be implemented without any architectural modifications, and can therefore be immediately applied to other architectures other than Soar. Under ideal circumstances, preemptive strategies perform well, especially if the goal targets are known during agent design, in which case a perceptually-triggered retrieval strategy can be used. Otherwise, preemptive strategies err on the side of efficiency while sacrificing performance.

A number of environmental properties, however, may indicate that preemptive strategies may not be appropriate. If the retention interval is lengthy, for example, preemptive strategies require consistent goal maintenance operations, meaning that the cost of completing a goal increases proportionally with its retention interval. The natural extension to this scenario is where the retention interval is infinite and the goal targets may *never* be satisfied, where the cost may continue to grow over the entirety of the agent's lifetime. Alternately, if the foreground task is cannot be interrupted by the goal maintenance operations, preemptive strategies may also not be the appropriate choice.

Spontaneous retrieval, together with spreading activation, are both new mechanisms to Soar, although related mechanisms exist in other architectures. In ACT-R, a mechanism known as *buffer stuffing* may be immediately used to implement a spontaneous retrieval strategy; on the other hand, the "analogical tickler" in Companions may require modifications, since it does not use activation as an underlying mechanisms and the selection bias may not be as suitable for goal re-activation. By allowing the architecture to heuristically determine what may be useful to the agent and *when* it may be useful — in this case, the goal when its target is satisfied — spontaneous retrieval frees the agent from having to deliberately bringing goals into working memory. This allows the strategy to complete goals at low cost, while remaining relatively robust to both the number of goals and the

retention interval.

As with all heuristics, there are cases where spreading activation may mislead spontaneous retrieval. Spreading activation is used here as an approximation of centrality on a network, with spontaneous retrieval selecting the most central element to be recreated in memory. It is easy to image scenarios where the perceptual features do not center around the goal target. To use the existing domain of Subset Missing link (presented in Section 5.4.1), if the goal was one of the distractors, then by the design of the domain the goal would not be central to the clue words. None of the strategies developed in this thesis are particularly well-suited for this scenario, since preemptive strategies would likely incur costs in terms of word puzzle performance, while spontaneous retrieval would fail to retrieve the goal at the initiation stage.

Ultimately, given the current mechanisms of cognitive architectures, there is likely no single, universal strategy (or mixture of strategies) that would guarantee the re-activation of suspended goals at their initiation stages. While spontaneous retrieval, as presented in this thesis, achieves high performance at low cost on a subset of goal re-activation tasks — specifically, where the goal is the most-centrally-located element in long-term memory from the percepts — this heuristic fails on even simple domains where the assumption of centrality assumption does not hold. Given the success of spreading activation and spontaneous retrieval, the next step may be to consider alternate methods of increasing the information flow between working memory and long-term memory, such that relationships other than centrality can also be transmitted. The goal re-activation problem in cognitive architectures is caused by the partial observability of long-term memory, and reducing the separation between the memory systems may allow agents to flexibly manage its goals regardless of the delay between encoding and initiation.

CHAPTER 7

Conclusion and Future Work

This chapter concludes the thesis by revisiting the contributions listed in the introductory chapter, and discussing some of the questions that the thesis either has set aside or has raised about goal re-activation and cognitive architectures in general.

7.1 Contributions

The main contributions of this thesis and a brief discussion are listed below.

- **Definition of the goal re-activation problem for cognitive architectures, and identification of a circular knowledge dependency as its central challenge.** Chapter 3 shows that the problem is one of circular dependencies for taking action. Although this thesis is about retrieving goals, the circular dependency problem is in fact one that affects all knowledge in long-term memory. For the cognitive architecture community, where there has been continued research on the relationship between working and long-term memory, laying out the knowledge retrieval problem as these dependencies allows a more focused approach in designing or refining architectural mechanisms. There are other types of agents that may encounter the same circular dependency problem as well — one example would be agents with a networked knowledge base, with the same underlying problem of partial observability with respect to the knowledge available to the agent's decision making process.

  Separately, the parallels between goal re-activation and prospective memory mean that the definition may also benefit memory researchers. This computational description of prospective memory is crisper than current definitions in psychology, even if the distinction between human working and long-term memory is not as well articulated. For the cognitive modeling community, however, this definition allows models to focus on the core issue of prospective memory, which is about the control of memory retrieval mechanisms; only once the mechanism is determined should the focus turn to the precise timing of retrievals.

- **Generalization of existing strategies, and development of new strategies, for goal re-activation.** Chapters 4, 5, and 6 presented three classes of strategies for goal re-activation. These strategies follow from the understanding that goal re-activation is a problem of circular dependencies. Preemptive strategies retrieve goals into working memory before they are needed, and provides a framework that accommodates the majority of existing models of goal re-activation. Spontaneous retrieval strategies, on the other hand, is a new strategy that only retrieval goals when they are relevant. Although the spontaneous retrieval memory mechanism itself is not new to the cognitive architecture literature, this is the first demonstration of its utility that we know of; a theoretical understanding of this mechanism is the fourth major contribution of this thesis below. Finally, a hybrid preemptive-spontaneous strategy is also possible, as the two classes of strategies are not mutually exclusive. While this is not an exhaustive taxonomy of strategies for goal re-activation, it provides a framework in which other strategies can be understood.

  This taxonomy of goal re-activation strategies is of the most interest to Soar developers, and also to the developers of other cognitive architectures to a lesser degree. With the recent focus on task learning [34] and the ability for agents to efficiently execute learned tasks at the right time, the goal re-activation strategies developed in this thesis are likely to be incorporated into existing agents. Separately, cognitive modelers may also be interested in this taxonomy of strategies, one based on the functional requirements rather than on the use of cognitive resources, as it hints at a unified theory for how prospective memory is performed.

- **Implementation and evaluation of strategies with respect to environmental and task properties.** We evaluated the three classes of strategies in Chapters 4, 5, and 6. The strategies were compared in Section 6.2 across environments with varying number of goals and average retention interval lengths, which demonstrated the relative strengths and weaknesses of each strategy. In general, however, strategies that use spontaneous retrieval dominate strategies that do not. However, there are also adversarial environments in which spontaneous retrieval performs poorly, such as the leave-one-out scenario, the use of deliberate retrieval in a hybrid strategy may increase performance. The dominance of spontaneous retrieval strategies justifies its further exploration as a solution to the goal re-activation problem.

  The abstract evaluation domain allows agent designers to extract general trends in the performance of the strategies, which may allow an informed choice in future agents. Cognitive modelers, however, may also find the evaluations from this thesis useful, especially since these strategies can be transferred to ACT-R. If ACT-R is taken as a good model of human cognition and memory, then the performance of these strategies should exhibit trends that are also found in people. For example, the negligible effect of the retention interval directly

contradicts the observed phenomenon that people are less likely to remember a prospective goal the longer they have to wait. While the results in this thesis cannot be directly compared to human behavior, it does suggest that additional factors are necessary to explain the degradation of prospective memory performance over time. Thus, although the domain does not represent a naturalistic task, it may help tease apart otherwise confounding factors in models of prospective memory.

- **Characterization of the benefits of spontaneous retrieval.** Chapter 5 introduced a spontaneous retrieval strategy, but also justified spontaneous retrieval as an architectural mechanism, as a heuristic for when agents lack search knowledge. The key is that spontaneous retrieval provides bidirectional access to knowledge, as compared to the standard unidirectional access from deliberate retrieval. The fact that the cognitive architecture community has not yet encountered a problem where spontaneous retrieval is necessary suggests that there is a class of problems that have not been tackled — where the agent does not have procedural knowledge of which facts are relevant, or how to get to the relevant knowledge. In other words, cognitive architectures have yet to tackle a domain where knowledge search is necessary; instead, in most domains the agent knows exactly how to access knowledge, or at worst, must iterate through a small, prescribed set of memory elements. In both of these cases, it is implied that the agent also knows when to search memory, knowledge that the agent does not have when facing the goal re-activation problem. One reason that spontaneous retrieval has yet to receive widespread use may be because it is difficult to create domains where the agent has the relevant knowledge but cannot take advantage of it; the knowledge requirements of most current task domains are sufficiently simple that it can be coded into the agent. With a theory of the benefits of spontaneous retrieval, the cognitive architecture can revisit the current use cases of memory, to consider where a deliberate search may not be the most efficient method of extracting knowledge.

7.2 Future Work

In the process of exploring two basic strategies for goal re-activation, there remain open questions about the problems that have not been addressed. This section discusses some of these issues, as well as other questions that this thesis raises.

7.2.1 Goal Re-Activation

There are several other strategies for goal re-activation that were not explored in this thesis. One of them, noticing-plus-search, was mentioned in Section 3.2.3 as a strategy that mixes preemptive and

spontaneous strategies. In particular, it requires that the agent generate spontaneous metamemory judgments, which are then used as the trigger for the retrieval of a goal. Although one form of metamemory judgment — that of recognition, or of having perceived something before — was implemented in Soar (see Appendix B), there remains no goal re-activation strategy that uses the judgment. Progress was halted mostly due to the large amount of noise in the judgment, as too many percepts are often recognized for the judgment to be a useful trigger for retrieval. An open question for implementing this strategy is therefore the *type* of metamemory judgment that is required and how it might affect goal re-activation performance. At a higher level, the cognitive architecture community still lacks a theory of why metamemory judgment may be useful, nor have we explored the space of metamemory judgments and their mechanisms. We need to first answer these preliminary questions before noticing-plus-search can be implemented and evaluated.

A second strategy that was not explored is the use of external memories; that is, changing the environment to lead to a likely retrieval of the goal at a later time. This strategy can be explained by the knowledge dependency framework in several ways. For example, the external memory strategy of tying a string to one's finger may play a similar role to the periodic retrievals in the hybrid strategy — the frequent perception of the string may keep the long-term memory activation of the goal from decaying too much, or may lead the agent to attempt to retrieve suspended goals into working memory. Alternately, a different external memory strategy for the goal of buying milk may be to place an empty milk carton in the passenger seat. This is a strategy that addresses the perceptual knowledge dependency cycle mentioned in Section 2.1.2, as the unexpected appearance of the carton is likely to trigger a memory retrieval. It is notable that the recognition of such surprise may itself be considered a type of metacognitive judgment, although in general noticing-plus-search has no relation to the use of external memory. An open question for the latter type of external memory strategy is whether the agent requires a model of its own perceptual system and attentional algorithms, in order to decide how to change the environment such that it will get the attention of its future self. Since this question has not been explored, external memory strategies remain unimplemented.

For the future researcher that implements these additional strategies, a more complex domain may also be needed. While the abstract domain allows variation in the basic parameters of a prospective memory task, it is also unrealistic in many ways. The percepts in this domain do not conform to any structural or temporal pattern, which may be necessary for the use of external memory. In fact, the agent has no ability to modify the environment at all in the current domain, rending the question of external memory moot. Additionally, in the current evaluations, semantic memory only contains a feature hierarchy, an extreme simplification of the structures found in knowledge bases. If instead semantic memory was filled with content from Cyc [38], and if the agent was presented with more semantically-rich input, the elements that are spontaneously retrieved

would be much more varied. This is especially true if the foreground task also requires deliberate retrieval, which would cause further changes to long-term memory activation (as shown in the hybrid strategy evaluation). While it is feasible that spontaneous retrieval strategies will continue perform well, it is also possible that these additional memory elements would cause retrieval interference. This would cause goals to have much lower activation than other knowledge, essentially leading to a leave-one-out scenario. Further experiments are needed to determine which will occur.

More analysis is also necessary to understand the cases where goal re-activation is a problem. Given the long history of research on memory mechanisms and goal-driven behavior, it is surprising that goal re-activation has not already been studied. One explanation is that it requires a particular type of ignorance on the part of the agent, that of not knowing when a goal would be relevant. It is possible that the domains currently tackled by the cognitive architecture community simply have not included such uncertainty. Regardless, a broader evaluation of goal re-activation strategies requires identifying the characteristics of domains where the problem arises.

7.2.2 Spontaneous Retrieval

One of the results of this thesis is the proposal of spontaneous retrieval as a new architectural mechanism. This proposal is justified by a theory of when spontaneous retrieval is useful, and is applied to the goal re-activation problem with success. Since this thesis is not on spontaneous retrieval, however, the space of possible implementations has not been explored. As suggested in Section 5.2, spreading activation is only one possible heuristic out of many for selecting the most relevant long-term memory for spontaneous retrieval. For this thesis, the feature hierarchy in long-term memory meant that spreading activation served as a good heuristic, but a structure-mapping-based heuristic [19] may perform better in a case-based reasoning domain. It may also be possible for the agent to learn the best heuristic by treating spontaneous retrieval as a machine learning problem: which memory element should be returned given the previous access history? Such algorithms may draw on existing work on network predictions, which is an area still under active research, especially when the relationships between entities are not homogeneous (unlike, for example, a social network of friends). Another source of inspiration may be the modeling of human associative memory, of which existing work exists in ACT-R [60]; again, it's possible that agents could learn the correct associations between working memory elements and the element to be retrieved. However, there is an additional challenge is that cognitive architectures currently can not determine which retrievals are correct and which are not — or at least, there has not been work showing that determining the correct retrievals is possible across domains. Nonetheless, these alternate memory biases are worth exploring; the only conclusion that can be drawn at this point is that the heuristic must match the knowledge representation and structure imposed by the task, and

that a broad survey of domains where spontaneous retrieval could be used would help constrain the design of the mechanism.

A separate concern from the heuristic used in spontaneous retrieval is how it is integrated into a cognitive architecture and its results used by the agent. The current design of reusing the buffer from deliberate retrieval means the agent has some control over whether spontaneous retrieval occurs. Thus far, however, no agent has taken advantage of this control: both the Missing Link agent from Section 5.4.2 and the goal re-activation agent merely wait for spontaneous retrievals to occur. While the hybrid strategy agent uses both deliberate and spontaneous retrieval, its design ignored any potential interactions between the two. As mentioned in Section 5.2, spontaneous retrieval is ultimately a mechanism for transferring relevant knowledge from long-term memory to working memory, and there is little existing work on how to design agents that do this successfully. It is possible that the optimal strategy requires deliberate retrieval under certain circumstances, while not deliberately retrieving in other situations so as to allow spontaneous retrievals to occur. A related problem is how an agent should cope with retrievals that are *ir*relevant. As demonstrated in the Missing Link domain, the lack of constraints on the spontaneously retrieved element means it must also be checked for usefulness. Since the only control the agent has is through deliberate retrieval, one possibility is to use deliberate retrieval to guide or direct spontaneous retrievals. In general, the interaction between deliberate and spontaneous retrieval remains unexplored.

Finally, it is possible that other mechanisms outside of deliberate and spontaneous retrieval exists. The metamemory judgments mentioned above may be one such mechanism. It is useful to think of the mechanisms as belonging on a spectrum: the recognition judgment only gives a single bit of information, while spontaneous retrieval results in a memory element being returned. The tradeoff is that recognition judgments are inexpensive to compute, while efficient spreading activation remains a research problem. It is possible that other points on this spectrum exist, and can provide agents with more fine-grained control over its memory usage. This possibility highlights how the space of memory mechanisms outside of deliberate retrieval remains poorly explored, and while this thesis offers an analysis of spontaneous retrieval, much work remains to be done.

7.2.3 Knowledge Search

Perhaps the most exciting area of research that this thesis hints at is the exploration of knowledge search. Knowledge search is the problem of retrieving the right knowledge for use in reasoning at the right time. Although knowledge search is hypothesized to be a necessary part of a generally intelligent agent [47], it has not been a topic addressed by the cognitive architecture community. This is in part because cognitive architectures were previously constrained by the single memory retrieval mechanism, that of deliberate retrieval. Knowledge search becomes trivial under the

implicit assumptions of deliberate retrieval, that the agent has all the information it needs to decide to retrieve, and that memory is well organized such that enumerating through memory elements is rarely required. This thesis shows that the assumptions do not always hold, and that spontaneous retrieval may be a more efficient mechanism for knowledge search. Similarly, additional mechanisms such as metamemory judgments provide a richer space of strategies for knowledge search, and allows for cases where the agent may be missing knowledge. Together, this means that the study of how agents retrieval knowledge is now necessary, for agents to create what may be called a *memory search strategy*. Creating such a strategy that is robust to domain and knowledge base content requires bringing together ideas not only from knowledge representation and databases, but also from decision making in order to understand how to best use the sources of information (metamemory) and the actions (deliberate and spontaneous retrieval) available. Additionally, this research may also shed light on how people retrieve knowledge from their own memory system as well, as it is known that human memory search is strategic [8]. The goal is to create agents that can reliable retrieve and use knowledge from new knowledge bases, without the need for reprogramming by the agent designer.

7.3 Conclusion

This thesis has defined and provided solutions for the goal re-activation problem in cognitive architectures. The effectiveness of the solutions argue for the uniform treatment of goals as knowledge, and that additional memory mechanisms — namely, spontaneous retrieval — are necessary for generally intelligent agents.

While the original problem is defined for the interaction between working memory and long-term memory, the circular knowledge dependency is a more general problem that applies not only to cognitive architectures, but also to other artificial intelligence agents with remote access to knowledge bases.

APPENDIX A

Description of ACT-R

The architecture that bears the most similarity to Soar is ACT-R [3], which was developed to model human memory and other phenomenon in detail. ACT-R was the first architecture to implement a long-term declarative memory, as an outgrowth of the original Human Associative Memory model [4]. In ACT-R, knowledge is represented as *chunks* (not to be confused with Soar's chunking mechanism). Chunks are stored and processed by *modules*, which perform specialized operations such as perception, motor actions, and memory access. These modules are only accessible through their buffers, each of which contains at most one chunk. The buffers of all the modules together act as ACT-R's working memory, and they contain the only knowledge that rules can match against and directly modify. Since the size and the number of buffers are fixed, ACT-R's working memory has limited capacity; in order to access a module, the current contents of its buffer must be replaced. This can be seen as a type of forgetting due to decay, where a chunk is only useful if no other information is needed in its buffer.

To overcome this extremely rapid forgetting of information from working memory, all chunks that have existed in various buffers are automatically stored into the *declarative memory* module. This module acts as ACT-R's long-term memory, allowing knowledge no longer available in working memory to be recovered. To retrieve knowledge from declarative memory, the agent must query DM with a partial description of the desired knowledge; knowledge that best matches the description will be recreated in the buffer, with probability proportional to its activation. A chunk in declarative memory may also be forgotten, based on its *base-level activation*, which summarizes its access history (Figure 2.3). Activation decreases exponentially over time, a process called decay, but is boosted whenever the chunk appears in a buffer, either due to retrieval or due to perception from the environment. If the activation of a chunk falls below a threshold, the chunk becomes unretrievable, and can only be boosted again through perception. Additionally, which chunk gets retrieved during a memory search depends on the relative activations of the matching chunks; the higher a chunk's activation, the more likely it is retrieved. Forgetting in ACT-R therefore occurs on two levels: knowledge may be forgotten from working memory when it is overwritten, and knowledge may be unretrievable from long-term memory due to both decay and interference.

In light of this hierarchy, ACT-R's long-term memory forgetting is no different from Soar's working memory forgetting, although the process of recovery is more involved.

APPENDIX B

Metamemory Judgments in Soar

Noticing-plus-search (NPS) strategies incorporate components of both spontaneous retrievals and preemptive strategies. NPS works by using a second, automatic, non-retrieval channel to memory, which we call *metamemory judgments* or simply memory *metadata* (examples below). This channel signals that parts of working memory might have connections to elements in long-term memory. Thus drawing the agent's attention to (*noticing*) a memory element, the agent may choose to use it as a cue to *search* memory for the source of this connection. On retrieval, if the source of this connection is an intention, the agent then verifies that the target is present and performs the necessary action. Since procedural knowledge is not dependent on the recognition of the target but instead on metadata about the agent's perceptions, NPS strategies remove the second dependency in Figure 3.1. As an example of noticing, a metamemory judgment may be made when the agent sees a bottle of milk at work, prompting it to search for and retrieve the intention to buy milk.

The key knowledge that enables NPS strategies is the metadata that the agent receives from the architecture. Psychology literature suggests several kinds of metamemory judgments that may indicate an object or event has been previously stored in memory [64]. The most direct are familiarity and recognition judgments, which convey similarities between current and previous perceptions. More subtle is the noticing of discrepancies between the expected and actual processing fluencies. Regardless of the type, metamemory judgments suggest cues with which the agent could search memory. As with spontaneous retrievals, metamemory judgments do not benefit only prospective memory; the long-term memory connection may originate from some source other than an intention. Unlike spontaneous retrievals, however, the agent can decide whether to retrieve and whether to add additional cue constraints; for prospective memory, the agent could specify the result to be an intention, thus filtering out irrelevant memory elements.

The likely failure point for NPS strategies is not the number of irrelevant memory retrievals, but the number of judgments that are unrelated to prospective memory. The agent must decide which of many judgments are related to uncompleted intentions, such that the number of (potentially unfruitful) memory retrievals is minimized. This is the same problem as with spontaneous retrievals:

both strategies require trading off between the generality of an architectural mechanism and the performance on prospective memory tasks. Since the strategies differ in where the architecture must provide information, they also differ in where this trade-off occurs. Our minimal assumption that goals are ordinary memory elements does not suggest a point in this trade-off; we leave the exploration of this space for future work.

APPENDIX C

Complete List of Experimental Parameters

A complete list of additional parameters for the abstract domain (Section 4.7) is listed below.

- Working memory decay rate — the base-level activation decay rate for working memory. Set to 0.5.
- Working memory forgetting threshold — the threshold at which, if the activation of a working memory element drops below, the element is removed from working memory. Set to 0.
- Long-term memory decay rate — the base-level activation decay rate for working memory. Set to 0.5.
- Random seed — the random seed which determines the feature hierarchy in long-term memory, the encoding and initiation time of goals, and the random percepts that are presented to the agent.
- Number of features per goal/abstract feature — the branching factor of the feature hierarchy. Set to 7.
- Percept change μ, σ — the normal distribution from which the probability of whether a perceptual feature is added/removed for the next timestep is drawn. Set to 0.5 and 0.05 respectively.

BIBLIOGRAPHY

[1] Erik M. Altmann and J. Gregory Trafton. Memory for goals: An activation-based model. *Cognitive Science*, 26(1):39–83, 2002.

[2] John R. Anderson. *The Adaptive Character of Thought*. Psychology Press, 1990.

[3] John R. Anderson. *How Can the Human Mind Occur in the Physical Universe?* Oxford University Press, 2007.

[4] John R. Anderson and Gordon H. Bower. Recognition and retrieval processes in free recall. *Psychological Review*, 79(2):97–123, 1972.

[5] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A nucleus for a web of open data. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The 4nd International Semantic Web Conference (ISWC)*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer Berlin Heidelberg, 2007.

[6] Lars Braubach and Alexander Pokahr. Representing long-term and interest BDI goals. In Lars Braubach, Jean-Pierre Briot, and John Thangarajah, editors, *Programming Multi-Agent Systems (ProMAS-7)*, pages 29–43. IFAAMAS, 2009.

[7] Lars Braubach, Alexander Pokahr, Daniel Moldt, and Winfried Lamersdorf. Goal representation for BDI agent systems. In Rafael Bordini, Mehdi Dastani, Jürgen Dix, and Amal Seghrouchni, editors, *Programming Multi-Agent Systems (ProMAS-3)*, volume 3346 of *Lecture Notes in Computer Science*, pages 44–65. Springer, 2005.

[8] Paul W. Burgess and Tim Shallice. Confabulation and the control of recollection. *Memory*, 4(4):359–412, 1996.

[9] Stuart K. Card, Thomas P. Moran, and Allen Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum, 1983.

[10] Nicholas L. Cassimatis, Perrin G. Bignoli, Magdalena D. Bugajska, Scott Dugas, Unmesh Kurup, Arthi Murugesan, and Paul Bello. An architecture for adaptive algorithmic hybrids. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 40(3):903–914, 2010.

[11] Yang Chen, Milenko Petrovic, and Micah H. Clark. SemMemDB: In-database knowledge activation. In *Proceedings of the 27th International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, pages 18–23, 2014.

[12] Richard P. Cooper. The role of falsification in the development of cognitive architectures: Insights from a Lakatosian analysis. *Cognitive Science*, 31(3):509–533, 2007.

[13] Nate Derbinsky, John E. Laird, and Bryan Smith. Towards efficiently supporting large symbolic declarative memories. In Dario D. Salvucci and Glenn F. Gunzelmann, editors, *Proceedings of the 10th International Conference on Cognitive Modeling (ICCM)*, pages 49–54, 2010.

[14] Nate Derbinsky, Justin Li, and John E. Laird. A multi-domain evaluation of scaling in a general episodic memory. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI)*, pages 193–199, 2012.

[15] Scott A. Douglass and Christopher W. Myers. Concurrent knowledge activation calculation in large declarative memories. In *Proceedings of the 10th International Conference on Cognitive Modeling (ICCM)*, pages 55–60, 2010.

[16] Constantinos Dovrolis, Brad Thayer, and Parameswaran Ramanathan. HIP: Hybrid interrupt-polling for the network interface. *SIGOPS Operating Systems Review*, 35(4):50–60, 2001.

[17] Renée Elio. On modeling intentions for prospective memory performance. In *Proceedings of the 28th Annual Conference of the Cognitive Science Society (CogSci)*, pages 1269–1274, 2006.

[18] Judi Ellis. Prospective memory or the realization of delayed intentions: A conceptual framework for research. In Maria A. Brandimonte, Gilles O. Einstein, and Mark A. McDaniel, editors, *Prospective Memory: Theory and Applications*, pages 1–22. Lawrence Erlbaum, 1996.

[19] Kenneth D. Forbus, Dedre Gentner, and Keith Law. MAC/FAC: A model of similarity-based retrieval. *Cognitive Science*, 19(2):141–205, 1995.

[20] Kenneth D. Forbus and Thomas R. Hinrichs. Companion cognitive systems: A step toward human-level AI. *AI Magazine*, 27(2):83–95, 2006.

[21] Kenneth D. Forbus, Matthew Klenk, and Thomas R. Hinrichs. Companion cognitive systems: Design goals and some lessons learned. *IEEE Intelligent Systems*, 24(4):36–46, 2009.

[22] Charles L. Forgy. *On the Efficient Implementation of Production Systems*. PhD thesis, Carnegie Mellon University, 1979.

[23] Fernand R. Gobet and Peter Lane. The CHREST architecture of cognition the role of perception in general intelligence. In *Proceedings of the 3rd Conference on Artificial General Intelligence (AGI)*, 2010.

[24] Maurice Grinberg, Vladimir Haltakov, and Hristo Stefanov. Approximate spreading activation for efficient knowledge retrieval from large datasets. In *Proceedings of the 2011 Conference on Neural Nets WIRN10: Proceedings of the 20th Italian Workshop on Neural Nets*, pages 326–333, Amsterdam, The Netherlands, The Netherlands, 2011. IOS Press.

[25] J. E. Harris. Remembering to do things: A forgotten topic. In J. E. Harris and Peter E. Morris, editors, *Everyday Memory, Actions and Absent-mindedness*, pages 71–92. Academic Press, 1984.

[26] Taher H. Haveliwala. Topic-sensitive PageRank. In *Proceedings of the 11th International Conference on World Wide Web (WWW)*, pages 517–526, New York, NY, USA, 2002. ACM.

[27] Pernille Hemmer and Mark Steyvers. A Bayesian account of reconstructive memory. *Topics in Cognitive Science*, 1(1):189–202, 2009.

[28] Douglas L. Hintzman. Research strategy in the study of memory: Fads, fallacies, and the search for the "coordinates of truth". *Perspectives on Psychological Science*, 6(3):253–271, 2011.

[29] Randolph M. Jones and Robert E. Wray. Comparative analysis of frameworks for knowledge-intensive intelligent agents. *AI Magazine*, 27(2):57–70, 2006.

[30] Matthew Klenk, Matthew Molineaux, and David W. Aha. Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence*, 29(2):187–206, 2013.

[31] Asher Koriat. Metamemory: The feeling of knowing and its vagaries. In Michel Sabourin, Fergus I. M. Craik, and Michèle Robert, editors, *Advances in Psychological Science, Volume 2: Biological and Cognitive Aspects*, pages 461–469. Psychology Press, 1998.

[32] Cody Kwok and Dieter Fox. Reinforcement learning for sensing strategies. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS) 2004*, volume 4, pages 3158–3163, 2004.

[33] John E. Laird. *The Soar Cognitive Architecture*. MIT Press, 2012.

[34] John E. Laird. Report on the NSF-funded workshop on taskability, 2014.

[35] Christian Lebiere, 2014. Personal communication, 2014-08-27.

[36] Christian Lebiere and Bradley Best. Balancing long-term reinforcement and short-term inhibition. In *Proceedings of the 31st Annual Conference of the Cognitive Science Society (CogSci)*, 2009.

[37] Christian Lebiere and Frank J. Lee. Intention superiority effect: A context-switching account. *Cognitive Systems Research*, 3(1):57–65, 2002.

[38] Douglas B. Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

[39] Justin Li, Nate Derbinsky, and John E. Laird. Functional interactions between memory and recognition judgments. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI)*, pages 228–234, 2012.

[40] Richard L. Marsh, Gabriel I. Cook, and Jason L. Hicks. An analysis of prospective memory. *Psychology of Learning and Motivation*, 46:115–153, 2006.

[41] Mark A. McDaniel and Gilles O. Einstein. *Prospective Memory: An Overview and Synthesis of an Emerging Field*. Sage, 2007.

[42] Sarnoff A. Mednick. The associative basis of the creative process. *Psychological Review*, 69(3):220–232, 1962.

[43] George A. Miller. WordNet: A lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[44] Steven Minton. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42(2-3):363–391, 1990.

[45] Shiwali Mohan, Aaron Mininger, James Kirk, and John E. Laird. Acquiring grounded representations of words with situated interactive instruction. *Advances in Cognitive Systems*, 2, 2012.

[46] Thomas O. Nelson and Louis Narens. Metamemory: A theoretical framework and new findings. *Psychology of Learning and Motivation*, 26:125–173, 1990.

[47] Allen Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.

[48] Mark E. J. Newman. A measure of betweenness centrality based on random walks. *Social Networks*, 27(1):39–54, 2005.

[49] Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. A goal deliberation strategy for BDI agent systems. In T. Eymann, F. Klügl, Winfried Lamersdorf, M. Klusch, and M. Huhns, editors, *Third German Conference on Multi-Agent System TEchnologieS (MATES-2005)*. Springer, 2005.

[50] Christopher K. Riesbeck and Roger C. Schank. *Inside Case-Based Reasoning*. Psychology Press, 1989.

[51] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

[52] Dario D. Salvucci and Niels A. Taatgen. Threaded cognition: An integrated theory of concurrent multitasking. *Psychological Review*, 115(1):101–130, 2008.

[53] Colleen M. Seifert and Andrea L. Patalano. Opportunism in memory: Preparing for chance encounters. *Current Directions in Psychological Science*, 10(6):198–201, 2001.

[54] Abigail J. Sellen, Gifford Louie, J. E. Harris, and A. J. Wilkins. What brings intentions to mind? An *in situ* study of prospective memory. *Memory*, 5(4):483–507, 1997.

[55] Marin D. Simina and Janet L. Kolodner. Opportunistic reasoning: A design perspective. In *Proceedings of the 17th Annual Conference of the Cognitive Science Society (CogSci)*, pages 78–83, 1995.

[56] Javier Snaider, Ryan McCall, and Stan Franklin. The LIDA framework as a general tool for AGI. In *Proceedings of the 4th Conference on Artificial General Intelligence (AGI)*, 2011.

[57] Ron Sun. The CLARION cognitive architecture: Extending cognitive modeling to social simulation. In Ron Sun, editor, *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation*, pages 79–102. Cambridge University Press, 2006.

[58] Joshua B. Tenenbaum, Thomas L. Griffiths, and Charles Kemp. Theory-based Bayesian models of inductive learning and reasoning. *Trends in Cognitive Sciences*, 10(7):309–318, 2006.

[59] John Thangarajah, James Harland, David Morley, and Neil Yorke-Smith. Suspending and resuming tasks in BDI agents. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '08, pages 405–412. IFAAMAS, 2008.

[60] Robert Thomson, Aryn Pyke, Laura Hiatt, and Greg Trafton. An account of associative learning in memory recall. In *Proceedings of the 37th Annual Conference of the Cognitive Science Society (CogSci)*, 2015.

[61] Endel Tulving and Donald M. Thomson. Encoding specificity and retrieval processes in episodic memory. *Psychological Review*, 80(5):352–373, 1973.

[62] Manuela Veloso, Martha E. Pollack, and Michael T. Cox. Rationale-based monitoring for planning in dynamic environments. In *AIPS 1998*, 1998.

[63] Linda M. Wills and Janet L. Kolodner. Explaining serendipitous recognition in design. In *Proceedings of the 16th Conference of the Cognitive Science Society*, pages 940–945, 1994.

[64] Andrew P. Yonelinas. The nature of recollection and familiarity: A review of 30 years of research. *Journal of Memory and Language*, 46(3):441–517, 2002.