Eris: Episodic Representation and Reasoning in Scone

Scone Project Technical Report 2014-1

Scott E. Fahlman SEF@CS.CMU.EDU
Language Technologies Institute, Carnegie Mellon University, Pittsburgh, PA 15217 USA

Abstract

"Episodic representation" is defined here as representation of knowledge about world-models that change from one time-point to another and about the operations that cause or describe these changes: events, actions, sequences of actions, plans, goals, costs, typical time durations, tools and materials required, etc. "Episodic reasoning" is defined here as reasoning about these episodic representations: answering questions about what is going on, predicting future events, planning, plan recognition, generating plausible explanations, assigning credit or blame, etc. In this paper we present Eris[1], a unified architecture for episodic representation and reasoning in the Scone knowledge-base system. Our approach depends on Scone's built-in mechanisms for representing multiple distinct but overlapping world-models in the same knowledge base and for default reasoning with exceptions.

1. The Problem: Episodic Representation and Reasoning

Most of the symbolic knowledge representation systems currently used by the AI community focus on representing *static semantic knowledge* – that is, the world is described by statements and entities that are static and immutable. Such systems have many uses, but they only capture a part of the knowledge that we humans use in our everyday reasoning. We also make extensive use of *episodic knowledge*, in which the world model changes from one time to another, whether that change is due to specific actions or to natural forces (e.g. food spoiling, or the outside environment becoming dark as night falls).[2]

[1] Eris stands for "Episodic Representation In Scone". Eris is also the name of a Pluto-sized (or perhaps larger) ice-dwarf planet in a very eccentric orbit out beyond Neptune. In Greek mythology, Eris is the goddess of chaos, strife, and discord.

[2] This distinction between semantic and episodic knowledge is inspired by the distinction made in cognitive psychology between "semantic memory" and "episodic memory" – a distinction proposed and popularized by psychologist Endel Tulving (1972). However, in Tulving's work, the notion of episodic memory was specifically autobiographical, referring to scenes and episodes that the subject had actually experienced and often invoking some associated emotional state. Here we use the term "episodic" to refer to any action, event, or change of state that a subject can represent and reason about, whether or not it was actually experienced.

An *episodic knowledge representation* must be able to represent these changing world-models, the operations that are responsible for these changes (events, actions, action-sequences, plans, and goals) and information about these operations: how long they take, how much they cost (by various measures of cost), what pre-conditions must be satisfied in order for the action to take place, what conditions might interfere with success, and so on. The knowledge we have about these events, actions, and plans is at least as rich and varied as our knowledge of objects and static entities, their properties, and their relations.

Episodic reasoning refers to our ability to reason about these actions, plans, and other episodic entities. The system must be able to handle queries such as the following (though not necessarily in natural language):

- Why was action X performed?
- What is likely to occur next?
- How long will it take to perform action sequence X? What steps can be done concurrently? Do some steps involve waiting for a slow change in state to occur, as in cooking or waiting for paint to dry?
- What tools and materials are required in order to execute plan X? Are they available?
- What skills and specialized knowledge are required to execute plan X?

This is just a sample of the kinds of inference that a human-like episodic representation and reasoning system should be able to support. We want to use these episodic representations in a number of ways:

- Representing and reasoning about some actual or imaginary sequence of events.
- Given a few observed events – presumably parts of some larger plan – recognizing what the plan is. For example, if someone is observed getting a spare tire and jack out of a car's trunk, he may be preparing to change a flat tire – an inference that the reasoner should be able to make even if it can't see the damaged tire.
- Supporting a planning system that can consider alternative plan templates or *recipes* and choose a good one for the problem at hand.[3]

2. Eris: An Episodic Representation/Reasoning Architecture for Scone

In the remainder of this paper we will describe an architecture for implementing this kind of episodic representation and reasoning in the Scone Knowledge-Base System (KBS). Scone has been under active development for almost a decade in the Language Technologies institute of Carnegie Mellon University. Scone offers a number of unusual capabilities that are very useful for tasks of this kind.

---

[3] In our current work, we are interested in "good enough" or *satisficing* forms of planning (Simon, 1956), not optimal planning, which often introduces serious problems of computational tractability.

2.1 Relevant Characteristics of Scone

The Scone KBS has been specifically developed to support human-like common-sense reasoning. It offers great speed and scalability, easily scaling up to knowledge bases with a few million entities and statements while running on a standard workstation.

Scone, by itself, is not a complete AI or decision-making system; rather, it is a software component – a sort of smart memory system – that is designed to be used in a wide range of software applications. Like other knowledge-base systems, Scone provides support for representing symbolic knowledge about the world: general common-sense knowledge or detailed knowledge about some specific application domain, or perhaps some of each. Scone also provides efficient support for simple inference: inheritance of properties in a type hierarchy, following chains of transitive relations, detection of type mismatches, and so on. Scone supports flexible search within the knowledge base. For example, we can ask Scone to return all individuals or types in the knowledge base that exhibit some set of properties, or we can ask for a best-match if no exact match is available. See (Fahlman 2006) for a discussion of the marker-passing algorithms that give rise to this efficiency and scalability.

Scone's knowledge base is fundamentally a semantic network, with *nodes* representing conceptual entities (not words) and *links* representing relations between the attached entities. One particularly important type of link is the *is-a link*, which ties an individual or subtype to the more general type of which it is a member. For example, there is an is-a link between {elephant} and {mammal}. The curly-brace notation indicates a concept-node in Scone – there is a many-to-many mapping between these concept-nodes and words in a human language such as English. The concept {elephant} may have multiple names in any given human language, or none; a word such as "mouse" may be ambiguous – that is, tied to more than one distinct concept-nodes.

An is-a link effectively says that all the properties and relations attached to a more general node such as {mammal} are inherited by all of its subtypes and instances, such as {elephant} or {Clyde the elephant}. The inference machinery takes care of this inheritance, so there is no need to actually copy all this information. We say that, because of the is-a link and the inheritance machinery, {elephant} has become a *virtual copy* of the {mammal} prototype. This idea, and the mechanisms that implement it, are explored at greater length in (Fahlman, 2006).

Most types (or classes) in Scone are defined not by a formal definition, but by a *prototype description*, indicating the properties of the typical member of that type. In addition to inheriting relations and properties, subtypes and instances can inherit entire descriptions. A prototype may have many role-nodes that are part of its description, each of which has its own supertypes, properties, and relations to other roles. So if the typical {elephant} has one tail, one nose with very special properties, four legs, and a mother who is a female elephant, all of these structures are inherited – *virtually copied* – whenever we create an instance of an elephant. This inheritance of complex structures implements one kind of *frame system* in the knowledge base (Minsky, 1974). We can build complex structures like elephants or families or companies, and the Scone KBS behaves as if we had copied the entire complex description of the supertype for each subtype or instance.

The subtypes under a given type can be grouped into sets of mutually disjoint types via Scone's *split-link*, which can connect any number of subtypes into what we call a *split-set*. Under {person}, for example, there is a split-set containing {male person} and {female person}.

Normally, an instance X under {person} can be a member of either of these subtypes, but not both at the same time. Scone's inference mechanisms will signal an error if there is an attempt to create an is-a link from X to {male person} if X is already a subtype of {female person}, directly or indirectly. A different split-set under person is {child} and {adult} – again, an instance can be one or the other, but not both. However, it is perfectly legal for X to be both a {male person} and a {child}.

Scone offers expressive power that goes beyond standard first-order logic in two important ways: First, it supports default reasoning with exceptions: if we are told that Tweety is a bird, Scone can conclude that Tweety flies; if we later state that Tweety is a penguin, the conclusion about flying is withdrawn, but Tweety is still a kind of bird. This exception is implemented by placing an is-not-a link between {penguin} and {flying thing}, which over-rides the chain of is-a links.

Second, Scone supports higher-order logic, in which we can make statements about statements and in which this can affect the conclusions we draw. For example, in Scone is it easy to represent "Bob believes that John loves Mary, but it isn't really true." First-order logic is a fine tool for certain simple kinds of reasoning, but if we can't represent and reason about statements like this one, we can't represent the plot of the average situation comedy or children's story. Scone's multiple-context mechanism (see below) is just a convenient way of packaging this capability for higher-order logic.

There are always tradeoffs: In order to achieve decidability and scalability in a system that also offers expressive power beyond that of first-order logic, we have had to give up the idea (common among designers of current knowledge-base systems) that all inference is done by some sort of logical theorem-proving procedure, with guarantees of logical completeness, provable consistency, and soundness. Scone's core inference machinery is designed to support the kinds of inference needed for everyday common-sense reasoning and natural language understanding, but Scone does not perform arbitrary logical inference to unlimited depth. Like a human, Scone does a certain amount of work when new knowledge is added, finding almost all of the simple inferences and contradictions – then it stops.

We believe that this is a good trade-off – probably a necessary trade-off – if we want a KBS that can provide human-like performance in real time on tasks such as story understanding. See (Fahlman, 2008) for a more thorough explanation of this choice and its consequences.

2.2 Multiple Contexts in Scone

The most unusual aspect of Scone is its multiple-context system. I will provide a brief summary of Scone contexts here. For a more extensive explanation of the multiple context mechanism, with examples of how it can be used to emulate several aspects of human-like reasoning, see (Fahlman, 2011).

A *context* is simply an entity in Scone's knowledge base that serves as a container for other knowledge; that is, a context creates a distinct world-model or world-view within the larger KB. Every entity in the Scone KB exists within some context; every statement is tied to some context within which it is valid. Before we make a query or deduction or add new knowledge to the Scone KB, we *activate* a specific context within which that operation, or set of operations, is to

take place. The knowledge in that context participates in the operation; the knowledge in inactive contexts is dormant. The marker-passing algorithms of Scone make it easy and efficient to implement these context-switches.

Scone's contexts are arranged in a hierarchy. Each new context starts as a clone of some pre-existing context. There is one special context called "general" where all of the system's general world-knowledge resides. If we want to investigate some "what if" scenario X, we simply create a new context under "general". This only requires adding new node for X and an inheritance link. Now whenever we activate X, we also activate "general" and all its contents. But the reverse is not true: when we activate "general", we do not see the contents of X. Now we can activate X, add some new statements or perhaps cancel some statements that otherwise would be inherited from "general", and begin to reason about the consequences of this what-if.

Contexts are used in many ways in Scone: to represent hypothetical or counterfactual situations; to represent the world view (and likely actions) of someone with an alternative belief system or an incomplete state of knowledge; to quarantine a collection of information (perhaps supplied by some unreliable informant) that we want to explore, but may or may not choose to believe; or to bundle together a set of statements which, if they are all satisfied, would trigger some action. Because the context mechanism is efficient and lightweight, Scone creates many contexts in a complex web of dependencies. It is also possible to work backwards, identifying a known context if we are given some of its contents.

2.3 Representation of Events and Actions

The multiple-context mechanism gives us a convenient way to represent the state of the world at various points in time – for example, after each step in a sequential plan and before the next step. This is the fundamental mechanism upon which our Eris system is built.

Consider an event-type such as {die}. In the prototype description for this event, there is a role for the {victim}, which must be of type {living thing} – that is, any kind of plant or animal. There is a role for the time at which the event occurs. And there are two very important roles, the {before-context} and the {after-context}. These represent the world-models before and after the event, respectively. In the {before-context} the {victim} is alive; in the after-context, the victim is dead. It is possible for the Scone user – a human or other program – to activate either of these contexts and to explore what conclusions can be drawn there. For example, in the {before-context}, the {victim} can move around and do things; in the {after-context}, the victim can only decompose.

A key point is that we only have to explicitly state what is different about these two contexts. The {before-context} will generally inherit the entire world-model of general knowledge that is present immediately before the event: it may be a sunny Thursday in Pittsburgh, cars and dogs and computers do what they normally do, gravity works according to certain laws, and so on. We don't have to copy any of that into the {before-context} – it is all inherited from the surrounding context. The {before-context} may explicitly contain only a statement that the victim is alive. This serves as a sort of pre-condition, in the style of the old STRIPS planner (Fikes & Nilsson, 1971) – it must be true in order for the event to take place. The {after-context} inherits everything in the {before-context}, but it cancels the statement that the victim is alive and replaces it with a statement that the victim is dead.

If we now create an instance of the {die} event, it will have a specific role-filler for the {victim} role and for the {event time} role. This represents an actual, specific event and not an event type.

An *action* is just an event, with all the same roles and inference machinery, but also an {agent} role, which usually is filled by some {living thing} or possibly {force of nature}. There is a relation stating that the {agent} *caused* the {event}. We can create sequences of events or actions by making the {after-context} of one be the {before-context} of the next one. A complex action may have an {expansion}, which is a sequence of simpler actions that, when taken together, compose the larger action. For example, the {expansion} of a {shoot} action may require loading the gun, pointing the gun, and pulling the trigger, in that order.

2.4 Three Dimensions

These event-types, action-types, instances of events and actions, sequences, and expansions can form a very complex and confusing structure in the knowledge base – a complex tangle of nodes and links. One way to make sense of this is to consider that there are three dimensions here – or rather three intersecting partial orderings.

First there is the *is-a* hierarchy under the {event} type-node. The model for {event} is very simple: it basically just says that *something* has changed, so there is just a {before-context}, and {after-context}, and an {event time}. These are inherited by all of the more specialized subtypes and instances of {event}. One major subtype is {action}, which just adds an {agent} role. And below that are all the different kinds of actions: going places, moving things, saying things, and so on. The type hierarchy can extend down to very specific action types – "going to the Pittsburgh airport from CMU during rush hour", for which we might or might not have a good recipe. The lowest nodes in the is-a hierarchy are specific *instances* of actions and events: "John Smith going to the airport at 5pm on Thursday."

Note that the split-sets described above play a significant role in this is-a hierarchy: The "kill" event-type may be split into "kill with a gun", "kill with poison", "kill by drowning", etc. And it might also be split according to legal status: "execute", "murder", "kill by accident", etc. So we might have an instance of murder by poisoning or an instance of an execution by shooting.

The second partial ordering follows the transitive *part-of* relation, rather than is-a. A complex action may be divided into sub-actions (i.e. *parts of* the larger action), and each of those steps may be divided into even smaller actions, and so on until we reach the level of *atomic actions* that are no longer in the cognitive domain – they are simply executed by the low-level motor-control system of the human or robot. So "go to the airport" may have *subtypes* "go to the airport by taxi", "drive to the airport by car", and so on. But "drive to the airport by car" can be broken into *parts*, such as "get into the car", "start the engine", and so on. "Turn the key in the ignition" is a *part of* going to the airport by car, not a subtype.

Finally, there is the network of *temporal before/after* relations – another partial ordering, since we might not know the temporal ordering of *all* the steps. If you want to shoot someone, you can buy the gun and buy the bullets in either order, but it is important to aim the gun *before* pulling the trigger.

A complex sequence of actions may require many hundreds of links to describe it, but the resulting structure can support many thousands of inferences. The structure virtually contains an immense number of relations, courtesy of the inheritance and virtual copy mechanisms. And within this structure there may be many distinct contexts: one for the state of the world after each step in the plan, and other contexts representing different viewpoints or different possible outcomes.

Much of our current research is aimed at turning simple natural-language descriptions – narratives – into the knowledge structures that can support Eris. And much research is going into modules to handle the reasoning made possible by these structures. For example, we want to be able to answer "why" and "how" questions, and perhaps even "why not" questions. I believe we now have a powerful foundation for Eris, but there is much structure still to be built on top of that foundation.

3. Implementation Status

The basic approach described here is not new. Many of the fundamental ideas were described in the author's Ph.D. thesis (Fahlman, 1978, section 3.9). However, the details were not worked out at that time, and the technology of the day was not ready to support implementation and testing of these ideas. It is only recently that we have returned to this problem of Eris, using the much faster computers available today.

Work on the core Scone system is ongoing, but a usable version of Scone has been running for several years. Scone has been used in a number of application projects at CMU and elsewhere. An open-source Scone release on the Internet is planned for sometime in the next few months.

The Eris scheme described here was implemented in prototype form by E. Cinar Sahin (2008). He tested and demonstrated the system on small problems in two application domains: monitoring events and actions at a conference (part of an intelligent personal assistant project) and detecting national-security threats.

Sahin's implementation was refined and extended by Maria Santofimia Romero (2011), who used the system in an “ambient intelligence” application, monitoring and trying to understand the activities of humans. She and colleagues are also applying this implementation to "smart power grid" problems.

The Scone Research Group at Carnegie Mellon is currently working on a more complete and robust implementation of Eris that will be part of a future Scone release. We currently have a project under way whose goal is to extend this Eris capability and to use it to support human-robot interaction. If a robot assistant is to work and communicate effectively with a human – even on a task as simple as changing a flat tire – there must be a large degree of overlap between the robot’s episodic knowledge and that of the human. They must both understand what the goal is, what plans are available, what plan is being worked on at present, and who is performing what part of the plan.

If they are to communicate effectively, the robot’s knowledge must be organized in such a way that it does not seem totally alien to the human. The robot must be able to make sense of a command like "While I am looking for the lug wrench, you jack up the car." We believe that the architecture described here is ideal for these goals.

There has recently been a good deal of interest at ONR and other agencies in developing AI systems with some measure of moral judgment, or at least the capability of following rather general rules of behavior laid down by humans. We believe that a pre-requisite for such research is to endow these systems with the capability of predicting the possible results of their actions, including both intended results and unintended side-effects. It will also be important to reason about whether an agent could reasonably have predicted certain outcomes. So we have begun doing some exploratory work in this area.

Acknowledgements

The work reported here was supported in part by the U.S. Office of Naval Research, under grant N000141310224. Any opinions, findings, conclusions, or recommendations expressed here are those of the author, and do not necessarily reflect the views of ONR or the U.S. government.

References

Fahlman, S. E. (1979) *NETL: A System for Representing and Using Real-World Knowledge*. Cambridge, MA: MIT Press.

Fahlman, S. E. (2006). "Marker-Passing Inference in the Scone Knowledge-Base System", *Proceedings of the First International Conference on Knowledge Science, Engineering and Management*. Springer Lecture Notes in AI.

Fahlman, S. E. (2008). *In Defense of Incomplete Inference.* Essay in "Knowledge Nuggets" blog: http://scone1.scone.cs.cmu.edu/nuggets/?p=34

Fahlman, S. E. (2011). Using Scone's multiple-context mechanism to emulate human-like reasoning. *Proceedings of the AAAI Fall Symposium on Advances in Cognitive Systems.*

Fahlman, S. E. (2013). The Scone knowledge-base project (home page). http://www.cs.cmu.edu/~sef/scone.

Fikes, R. E. & Nilsson, N. J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence 2*, 189-208.

Hwang, C. H. & Schubert, L. K. (1993). Episodic Logic: A situational logic for natural language processing. In P. Aczel, D. Israel, & S. Peters (eds.) *Situation Theory and its Applications 3* (STA-3), CSLI, 307-452, 1993.

McCarthy, J. & Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer & D. Michie (eds.) *Machine Intelligence 4*, 463-502.

Minsky, M. L. (1974). *A Framework for Representing Knowledge* (MIT-AI-Lab Memo 306). MIT Artificial Intelligence Laboratory, Cmabridge, MA.

Sahin, E. C. (2008), *Episodic Memory Representation in a Knowledge Base, with Application to Event Monitoring and Event Detection*. Master's Thesis, Language Technologies Institute, Carnegie Mellon University, Pittsburgh, PA.

Santofimia Romero, M. J. (2011). *Automatic Service Composition Based on Common-Sense Reasoning for Ambient Intelligence*. Doctoral Dissertation, Universidad de Castilla-La Mancha, Ciudad Real, Spain.

Simon, H. A. (1956). "Rational Choice and the Structure of the Environment". *Psychological Review 63* (2): 129–138.

Tulving, E. (1972). Episodic and semantic memory. In E. Tulving & W. Donaldson (Eds.), Organization of memory, (pp. 381–403). New York: Academic Press.