**Soar-SMem Manual**

11 May 2009

Contributors

Nate Derbinsky

Nicholas Gorski

John Laird

Bob Marinier

Sam Wintermute

Joseph Xu

Table of Contents

1. Document History 3

2. Soar-SMem Motivation 4

3. Working Memory Structure 5

4. Storing Semantic Concepts 6

4.1. Agent Storage 6

4.2. Long-Term Identifiers 6

4.3. Declarative Storage 6

4.4. Soar-SMem Storage 7

5. Retrieving Concepts 8

5.1. Soar-SMem Retrieval Commands 8

5.2. Non-Cue-Based Retrievals 8

5.3. Cue-Based Retrievals 8

6. Soar-SMem Parameters 11

6.1. Parameter Configuration 11

6.2. Parameter Descriptions 11

6.2.1. General 11

6.2.2. Storage 11

6.3. Full Parameter Configuration 12

6.4. Parameter Behavior 12

7. Soar-SMem Statistics 13

8. Soar-SMem Timers 14

9. Trace Information 16

10. Soar-SMem Performance 17

11. Soar-SMem Programmer Reference 18

11.1. Useful Commands 18

11.2. Parameters 18

11.3. Agent Commands 19

11.4. Retrieval Agent Meta-Data 19

# Document History

Version 0.1

Pilot version.

# Soar-SMem Motivation

Soar-SMem is a task-independent, architectural integration of an artificial semantic memory (SMem) with Soar. The SMem mechanism facilitates deliberate recording and querying of semantic chunks as a Soar agent executes.

# Working Memory Structure

Upon creation of a new state within working memory, the architecture will automatically create a structure in working memory called **smem**. Within this structure, agents issue requests to Soar-SMem by populating the **command** identifier with working memory elements (WMEs) and process Soar-SMem generated WMEs in the **result** identifier.

# Storing Semantic Concepts

This section details Soar-SMem storage, including the agent API, long-term identifiers, declarative storage, and format details.

## Agent Storage

An agent stores a concept in semantic memory by issuing a **store** command:

state.smem.command.store <identifier>

Multiple **store** commands can be issued in parallel during a single cycle. Storage commands are processed at the end of Output phase of every decision cycle. Storage is guaranteed to succeed, and thus no status is indicated.

Soar-SMem will store the identifier as well as its direct children. Storing multiple levels of working memory (i.e. grandchildren) is achieved through multiple **store** commands.

## Long-Term Identifiers

When an identifier is stored in semantic memory it is converted into a *long-term*identifier (LTI). The specific letter-number combination that labels the identifier (i.e. S5 or C7) is now permanently associated with the stored concept. Any future retrievals of the concept are guaranteed to return the associated letter-number pair.

Subsequent storage of an LTI will overwrite previous contents within semantic memory. It should be noted that between issuing **store** commands it is possible to have children of a concept in working memory be inconsistent with the long-term children stored in semantic memory.

## Declarative Storage

Soar-SMem provides the ability to declaratively store concepts via the **add** switch of the **smem** command. The format of the command is nearly identical to the working memory manipulation components of the RHS of a Soar production (i.e. no RHS-functions). For instance:

smem --add {

(<arithmetic> ^add10-facts <a01> <a02> <a03>)

(<a01> ^digit1 1 ^digit-10 11)

(<a02> ^digit1 2 ^digit-10 12)

(<a03> ^digit1 3 ^digit-10 13)

}

Unlike agent storage, declarative storage is automatically recursive. Thus, this command instance will add a new concept (represented by the temporary “arithmetic” variable) with three children. Each child will be its own concept with two constant attribute/value pairs. Declarative storage can be arbitrarily complex and use standard dot-notation.

Declarative concepts are stored immediately. Thus, storage parameters (such as **database** and **path**) should be set before issuing any **add** commands.

## Soar-SMem Storage

SMem currently uses SQLite to facilitate efficient and standardized storage and querying of episodes. The episodic store can be maintained in memory or on disk (per the **database** and **path** parameters). If the store is located on disk, users can use any standard SQLite programs/components to access/query its contents.

The **lazy-commit** parameter is a performance optimization. If set to **on** (default), disk databases will not reflect semantic memory changes until the Soar instance ends, to save disk I/O costs.

# Retrieving Concepts

This section details the agent interface to Soar-SMem retrievals, including command protocol, non-cue-based (NCB) retrievals, cue-based (CB) retrievals, and retrieval meta-data.

## Soar-SMem Retrieval Commands

An agent issues a retrieval command to the Soar-SMem system by populating appropriate WMEs on the **command** identifier of a state’s **smem** structure. At the end of each decision cycle, after concept storage, Soar-SMem processes each state’s SMem **command** structure. Results, meta-data, and errors are placed on the **result** identifier of that state’s **smem** structure.

Only one type of retrieval command (which may consist of multiple WMEs) can be issued in a single decision cycle (though multiple states may issue commands). Malformed commands (including attempts at multiple commands) will result in an error.

After a command has been processed, Soar-SMem will ignore it until some aspect of the **command** structure changes (via addition/removal of WMEs). When this occurs, the **result** structure is cleared and the new command (if one exists) is processed.

## Non-Cue-Based Retrievals

An NCB retrieval is a request to retrieve the direct children of a long-term identifier:

state.smem.command.retrieve <lti>

If the supplied identifier is not a long-term identifier, an error will result:

state.smem.result.status failure

Otherwise, two new WMEs will be placed on the **result** structure:

state.smem.result.status success

state.smem.result.retrieved <lti>

If the supplied long-term identifier has children in Working Memory, the **retrieved** LTI will remain unchanged. Otherwise, its direct children will be populated from semantic memory.

## Cue-Based Retrievals

CB retrieval commands are used to search for a concept in the store that exactly matches an agent-supplied cue, while potentially adhering to optional modifiers.

A cue is composed of WMEs that describe a concept’s direct children. A cue WME with a constant value (symbolic or numeric) demands an exact match of both attribute and value. A cue WME with an LTI as its value demands an exact match as well. A cue WME with a non-long-term identifier as its value requires an exact match of attribute, but with any value (constant or identifier).

A cue is issued on the **command** structure as a **query** identifier:

state.smem.command.query <cue>

For instance, consider the following query:

sp {sample\*query

(state <s> ^smem.command <sc>

^lti <lti>

^input-link.foo <bar>)

-->

(<sc> ^query <q>)

(<q> ^name <any-name>

^foo <bar>

^associate <lti>

^age 25)

}

In this example, assume that the “^lti <lti>” match will be an LTI and the value of “foo” from the input-link will be a constant. Thus, the query requests retrieval of a long-term identifier with ALL of the following:

* A child with attribute “name” and ANY value
* A child with attribute “foo” and value equal to the value of variable “<bar>” at the time this rule fires
* A child with attribute “associate” and value referring to the long-term identifier “<lti>” at the time this rule fires
* A child with attribute “age” and integer value 25

If no long-term identifier meets ALL of these qualifications, an error is returned:

state.smem.result.status failure

Otherwise, two WMEs are added:

state.smem.result.status success

state.smem.result.retrieved <lti>

During a cue-based retrieval it is possible that the retrieved LTI is not in Working Memory. If this is the case, Soar-SMem will create a new identifier with letter-number pair as was originally stored.

As with NCB retrievals, if the retrieved concept has direct children in Working Memory, these are not overwritten. Otherwise the direct children of the LTI are added to Working Memory.

It is possible that multiple concepts match the cue equally well. In this case, Soar-SMem will retrieve the LTI that was most recently stored/retrieved.

The CB retrieval process can be further tempered using optional modifiers:

* The **prohibit** command requires that the LTI of the retrieved episode is not equal to a supplied LTI:

state.smem.command.prohibit <bad-lti>

Multiple **prohibit** command WMEs may be issued as modifiers to a single CB retrieval. This method can be used to iterate over all matching concepts.

# Soar-SMem Parameters

The following sections discuss how to configure the Soar-SMem parameters discussed in previous sections.

## Parameter Configuration

Individual configuration parameters are retrieved and manipulated using the **get** and **set** switches of the **smem** command:

smem [-g|--get] <parameter>

smem [-s|--set] <parameter> <value>

Agents can retrieve and change parameters in the actions of rules using the **cmd** function.

## Parameter Descriptions

All Soar-SMem parameters are organized below. The *Protected* field is discussed in Section 6.4).

### General

|  |  |
| --- | --- |
| Purpose | Enable or disable Soar-SMem |
| Parameter | **learning** |
| Values | |  |  | | --- | --- | | **off** | Disable Soar-SMem | | **on** | Enable Soar-SMem | |
| Default | **on** |
| Protected | no |

### Storage

|  |  |
| --- | --- |
| Purpose | Specifies whether the semantic store will be maintained in memory or on disk |
| Parameter | **database** |
| Values | |  |  | | --- | --- | | **file** | Semantic store is maintained on disk | | **memory** | Semantic store is maintained in memory | |
| Default | **memory** |
| Protected | yes |

|  |  |
| --- | --- |
| Purpose | Specifies where on disk the semantic store will be saved |
| Parameter | **path** |
| Values | |  |  | | --- | --- | | **<empty>** | Soar-SMem will create a temporary database file on disk during execution (and delete it after use) | | **<valid path>** | Soar-SMem will use the specified path for its database file on disk - if the file doesn’t exist, it will be created | |
| Default | **<empty>** |
| Protected | yes |

|  |  |
| --- | --- |
| Purpose | Specifies how often the semantic store is saved to disk |
| Parameter | **lazy-commit** |
| Values | |  |  | | --- | --- | | **off** | Updates to the store are saved as they occur | | **on** | Updates remain in memory until the Soar instance ends | |
| Default | **on** |
| Protected | yes |

|  |  |
| --- | --- |
| Purpose | Declares the level to which Soar-SMem timers are enabled (akin to watch levels) |
| Parameter | **timers** |
| Values | |  |  | | --- | --- | | **off** | Timers are disabled | | **one** | Only total Soar-SMem time is recorded | | **two** | High-level timers are enabled (smem\_\*) | |
| Default | **off** |
| Protected | no |

## Full Parameter Configuration

Entering simply the **smem** command (with no switches) will return full parameter configuration information. For example, assuming default configuration, the result of executing **smem** is as follows:

>smem

SMem learning: on

Storage

-------

database: memory

path:

lazy-commit: on

timers: off

## Parameter Behavior

Upon attempting to set a Soar-SMem parameter, the new value is validated. If the value is found to be invalid, the system will use the previous value.

The set of parameters listed above that have a “yes” in the *Protected* field cannot be changed once the Soar-SMem system has been “initialized.” The Soar-SMem system initializes during execution of the first storage/retrieval.

# Soar-SMem Statistics

Feedback from the Soar-SMem system is retrieved using the **stats** switch of the **smem** command:

smem [-S|--stats] <statistic>

If a **statistic** argument is provided, the command returns the value of a specific statistic. The valid statistic arguments are listed below.

|  |  |
| --- | --- |
| Statistic | **mem\_usage** |
| Description | Current SQLite memory usage in bytes |
| Label | Memory Usage |

|  |  |
| --- | --- |
| Statistic | **mem\_high** |
| Description | Greatest SQLite memory usage in bytes since last database initialization |
| Label | Memory Highwater |

Agents can retrieve specific statistics in rule actions using the **cmd** function.

Note that SQLite memory stats are shared amongst all SQLite databases, meaning these numbers include memory used by episodic memory (Soar-EpMem).

Entering the **smem --stats** command with no statistic, or an invalid statistic, will return all statistics. A sample execution may look as follows:

>smem --stats

Memory Usage: 0

Memory Highwater: 0

# Soar-SMem Timers

Time spent on Soar-SMem operations is retrieved using the **timers** switch of the **smem** command:

smem [-t|--timers] <timer>

If a **timer** argument is provided, the command returns the value of a specific timer. The valid statistic arguments are listed below (with their associated level, respecting the **timers** parameter).

|  |  |
| --- | --- |
| Timer | **\_total** |
| Description | Total time spent by Soar-SMem |
| Level | one |

|  |  |
| --- | --- |
| Timer | **smem\_api** |
| Description | Time spent validating agent commands |
| Level | two |

|  |  |
| --- | --- |
| Timer | **smem\_hash** |
| Description | Time spent hashing symbols |
| Level | two |

|  |  |
| --- | --- |
| Timer | **smem\_init** |
| Description | Time spent initializing the semantic store |
| Level | two |

|  |  |
| --- | --- |
| Timer | **smem\_ncb\_retrieval** |
| Description | Time spent adding concepts (and children) to working memory |
| Level | two |

|  |  |
| --- | --- |
| Timer | **smem\_query** |
| Description | Time spent searching for cues |
| Level | two |

|  |  |
| --- | --- |
| Timer | **smem\_storage** |
| Description | Time spent storing concepts |
| Level | two |

Agents can retrieve specific timer values in rule actions using the **cmd** function. Timer values are re-initialized at the same time points as Soar timers.

Entering the **smem --timers** command with no timer will return all timers. A sample execution may look as follows:

>smem --timers

\_total: 0

smem\_api: 0

smem\_hash: 0

smem\_init: 0

smem\_ncb\_retrieval: 0

smem\_query: 0

smem\_storage: 0

# Trace Information

To view Soar-SMem debugging information, use the following watch switch:

watch [-s|--smem]

This function is not enabled by default or through any watch level. At present, this watch level does not serve a function.

# Soar-SMem Performance

Initial empirical results with the *arithmetic* demo agent show that SMem queries carry about a 10-15% overhead as compared to comparable rete matching.

However, Soar-SMem implements some basic query optimization: statistics are maintained about all concept storage. Thus, when a query is issued, Soar-SMem re-orders the cue such as to minimize expected query time. Because only perfect matches are acceptable, semantic memory retrievals will not suffer the same combinatorial search space as the rete.

# Soar-SMem Programmer Reference

The following tables list basic information about each of the Soar-SMem related commands. It is not intended to substitute for this document, but a quick reference for commonly used commands and options.

### Useful Commands

|  |  |  |
| --- | --- | --- |
| Command |  | Description |
| smem |  | Summary table of parameter settings |
| smem [-g|--get] <parameter> |  | Retrieve a Soar-SMem parameter value |
| smem [-s|--set] <parameter> <value> |  | Set a Soar-SMem parameter value |
| smem [-S|--stats] <statistic> |  | Access Soar-SMem statistics |
| smem [-a|--add] |  | Declaratively store concepts |
|  |  |  |
| watch [-s|--smem] |  | Soar-SMem debugging trace |

### Parameters

Parameters noted with a \* are *protected*.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **General** | | | | |
| Parameter Name |  | Acceptable Values |  | Default |
| learning |  | on  off |  | on |
|  |  |  |  |  |
| **Storage** | | | | |
| Parameter Name |  | Acceptable Values |  | Default |
| database\* |  | file  memory |  | memory |
|  |  |  |  |  |
| path\* |  | <empty>  <system path> |  | <empty> |
|  |  |  |  |  |
| lazy-commit |  | off  on |  | On |
|  |  |  |  |  |
| timers |  | off  one  two |  | off |
|  |  |  |  |  |

### Agent Commands

*Storage*

state.smem.command.store <identifier>

*NCB Retrieval*

state.smem.command.retrieve <lti>

*CB Retrieval*

state.smem.command.query <cue>

CB Retrieval Optional Modifiers

state.smem.command.prohibit <bad-lti>

### Retrieval Agent Meta-Data

state.smem.result

^retrieved <lti>

^status << success failure bad-cmd >>