Soar Workshop
Semantic Memory Tutorial

Nate Derbinsky

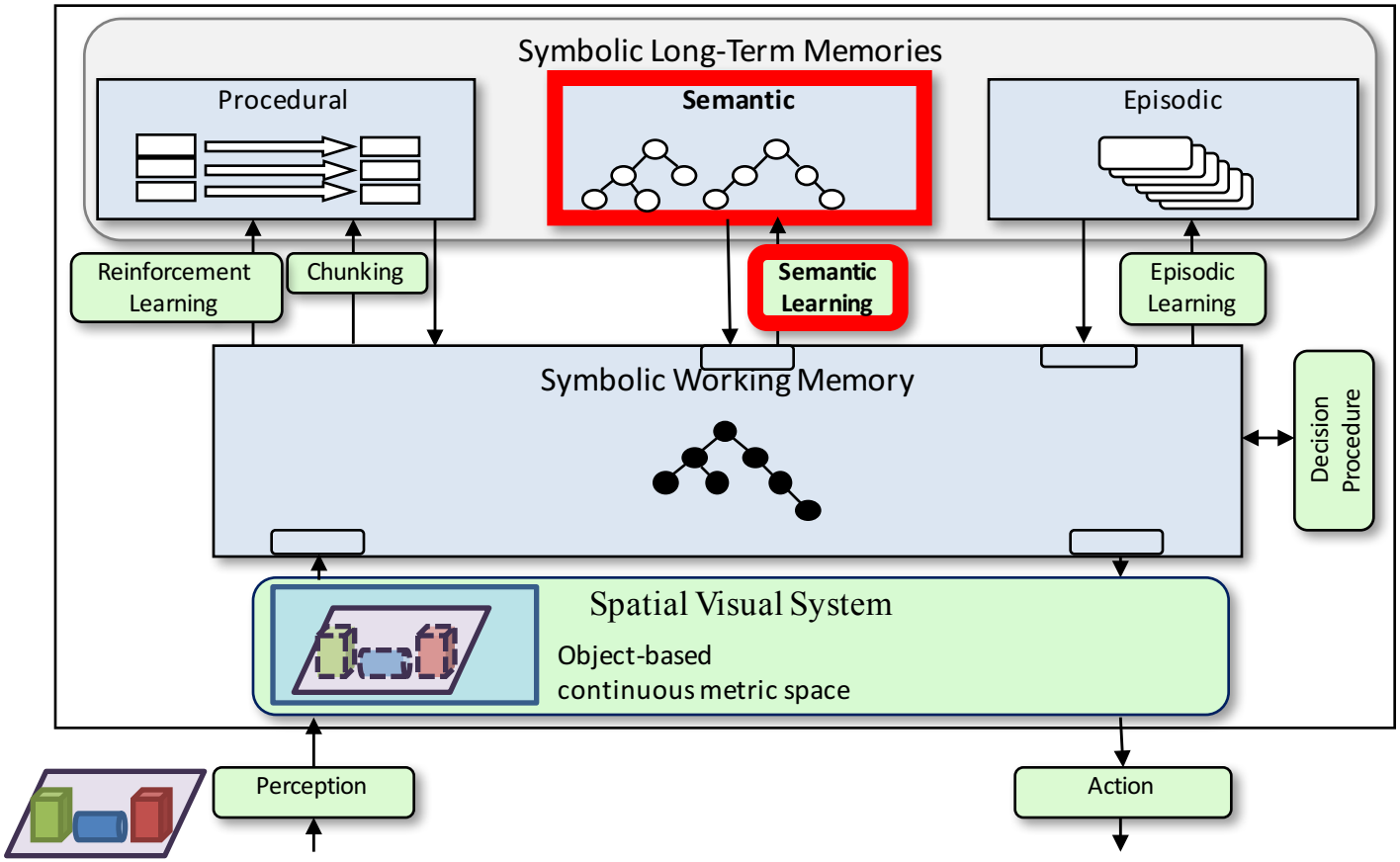
While waiting… download WordNet

http://web.eecs.umich.edu/~soar/tutorial16

Agenda

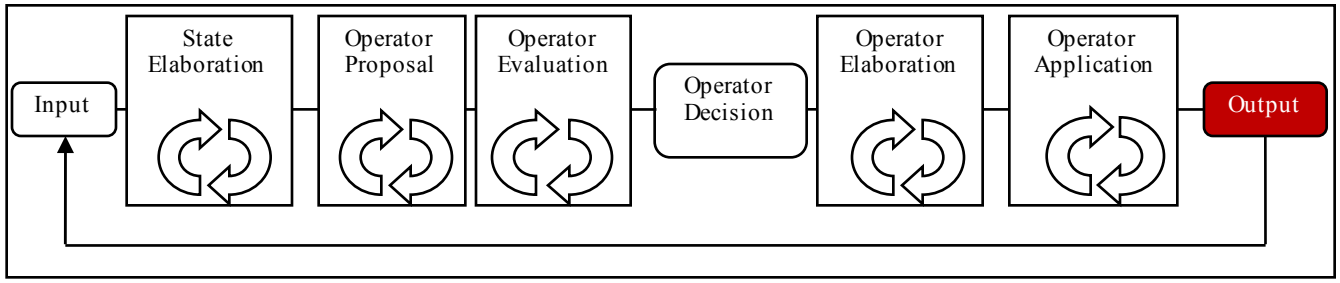
- Big picture
- Basic usage
- WordNet demo
- Additional resources

Soar 9


Symbolic Long-Term Memories
Procedural
Semantic
Episodic
Reinforcement Learning
Chunking
Semantic Learning
Episodic Learning
Symbolic Working Memory
Decision Procedure
Spatial Visual System
Object-based
continuous metric space
Perception
Action

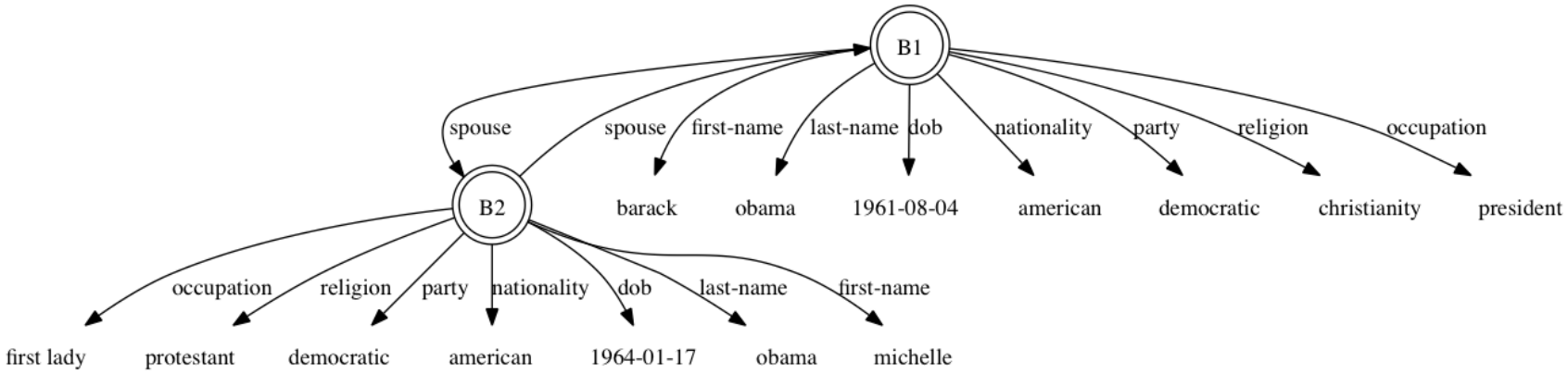
Soar Basic Functions

1. Input from environment
2. Elaborate current situation: *parallel rules*
3. Propose operators via acceptable preferences
4. Evaluate operators via *preferences: Numeric indifferent preference*
5. Select operator
6. Apply operator: Modify internal data structures: *parallel rules*
7. Output to motor system [and access to long-term memories]

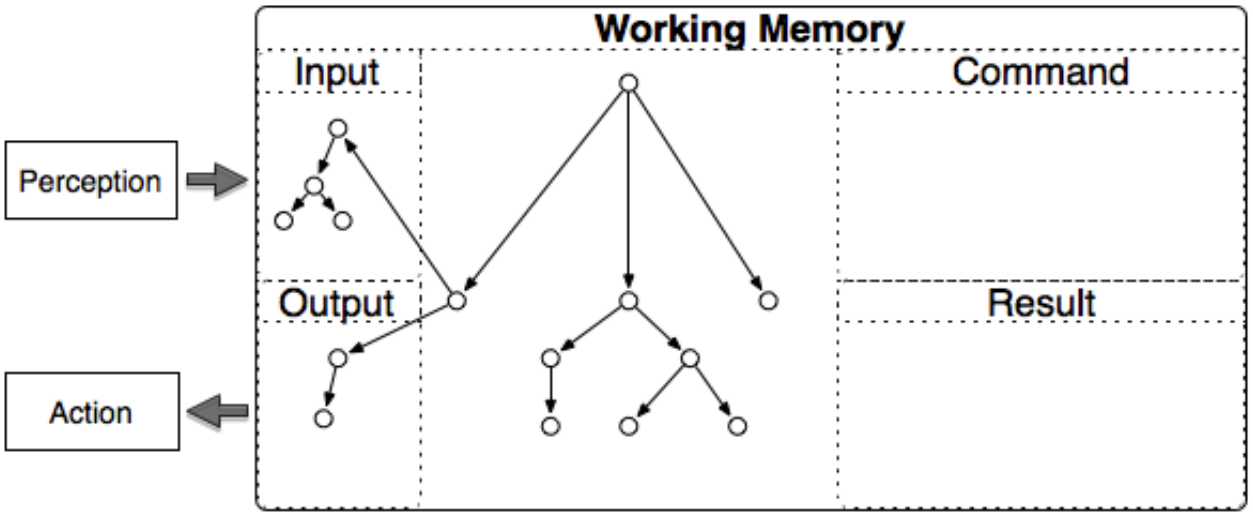


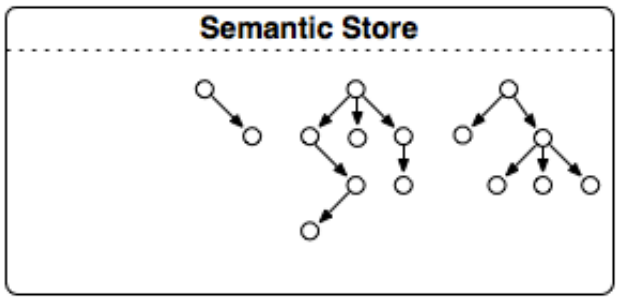
Semantic Memory: Big Picture

Supports deliberate storage and retrieval of long-term “objects,” features, and relations



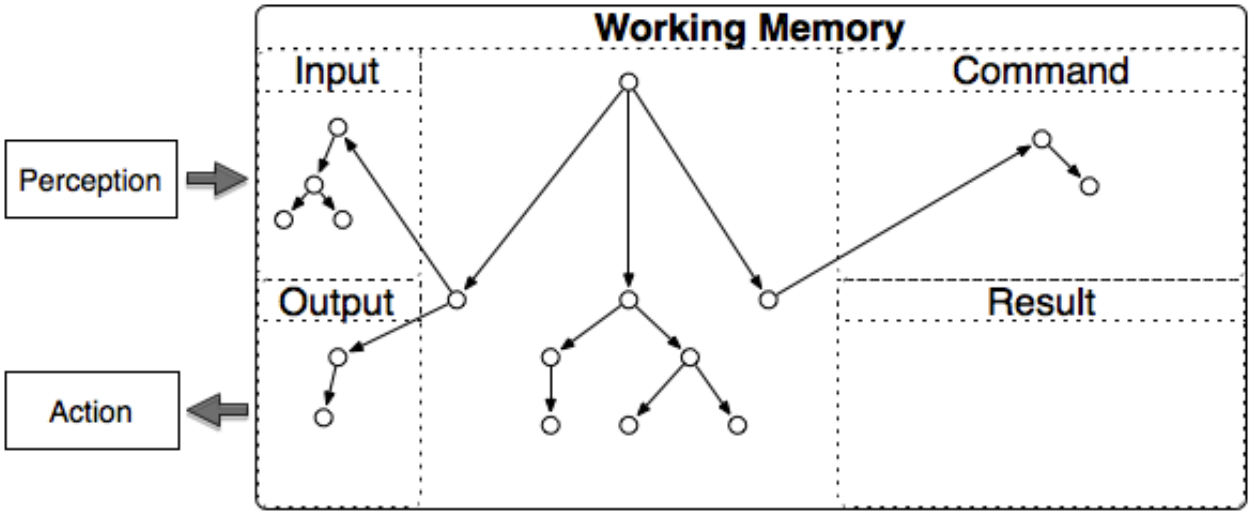
Architectural Integration

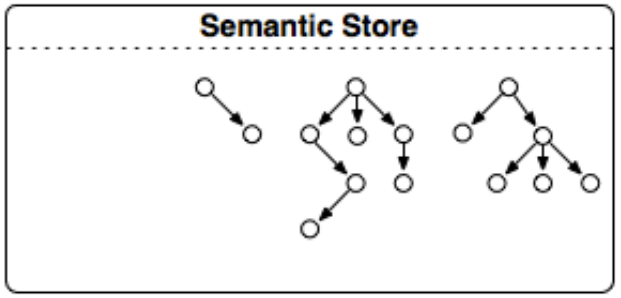




Architectural Integration

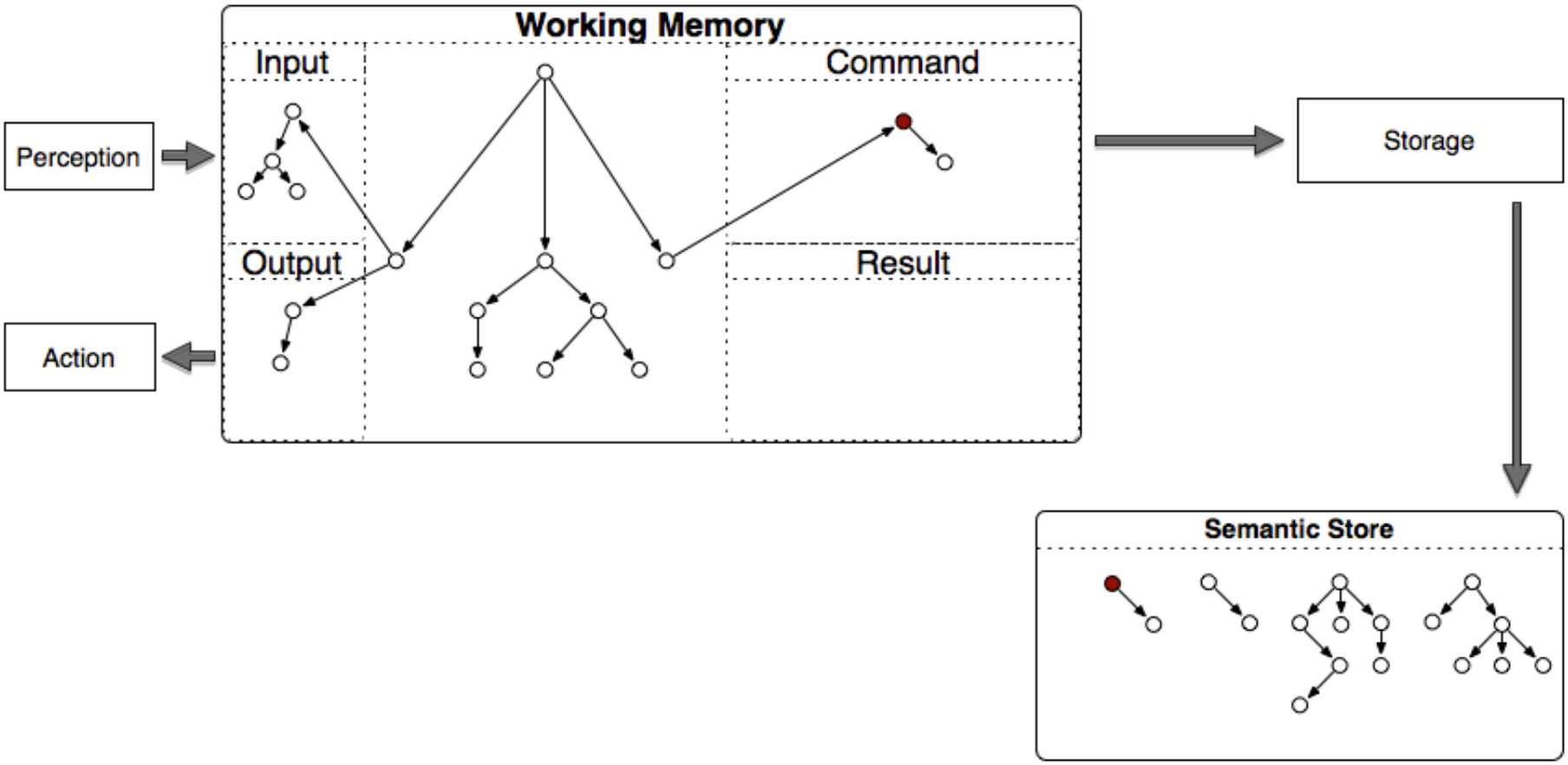
Storage





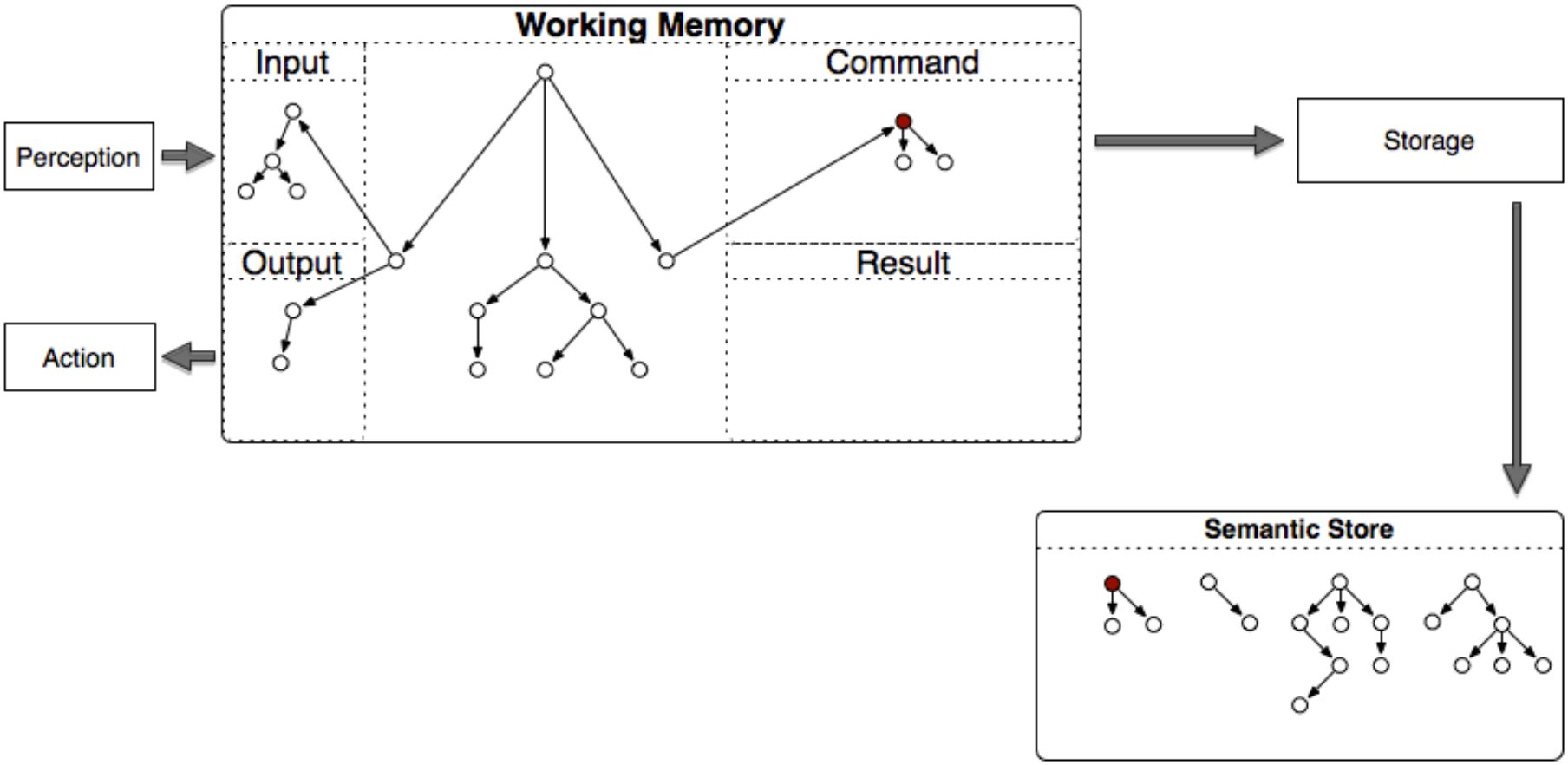
Architectural Integration

Storage



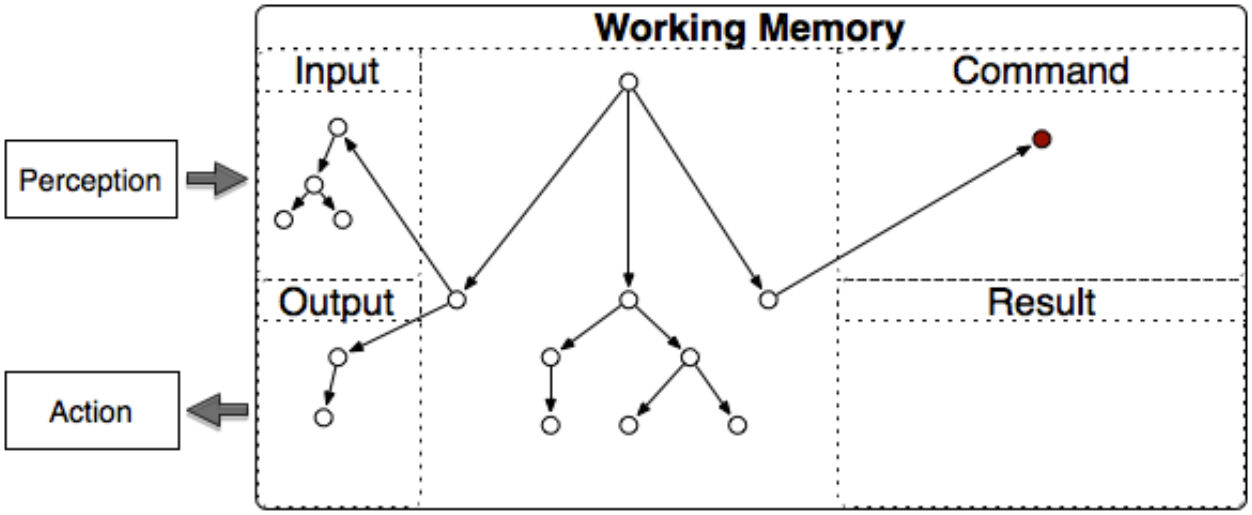
Architectural Integration

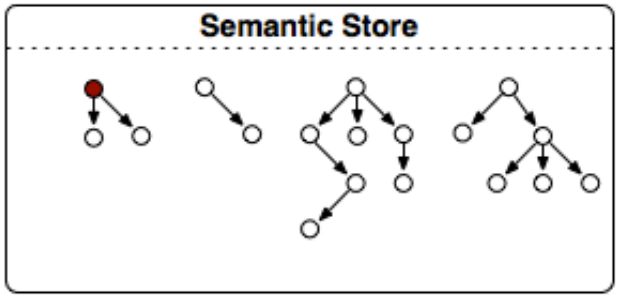
Storage



Architectural Integration

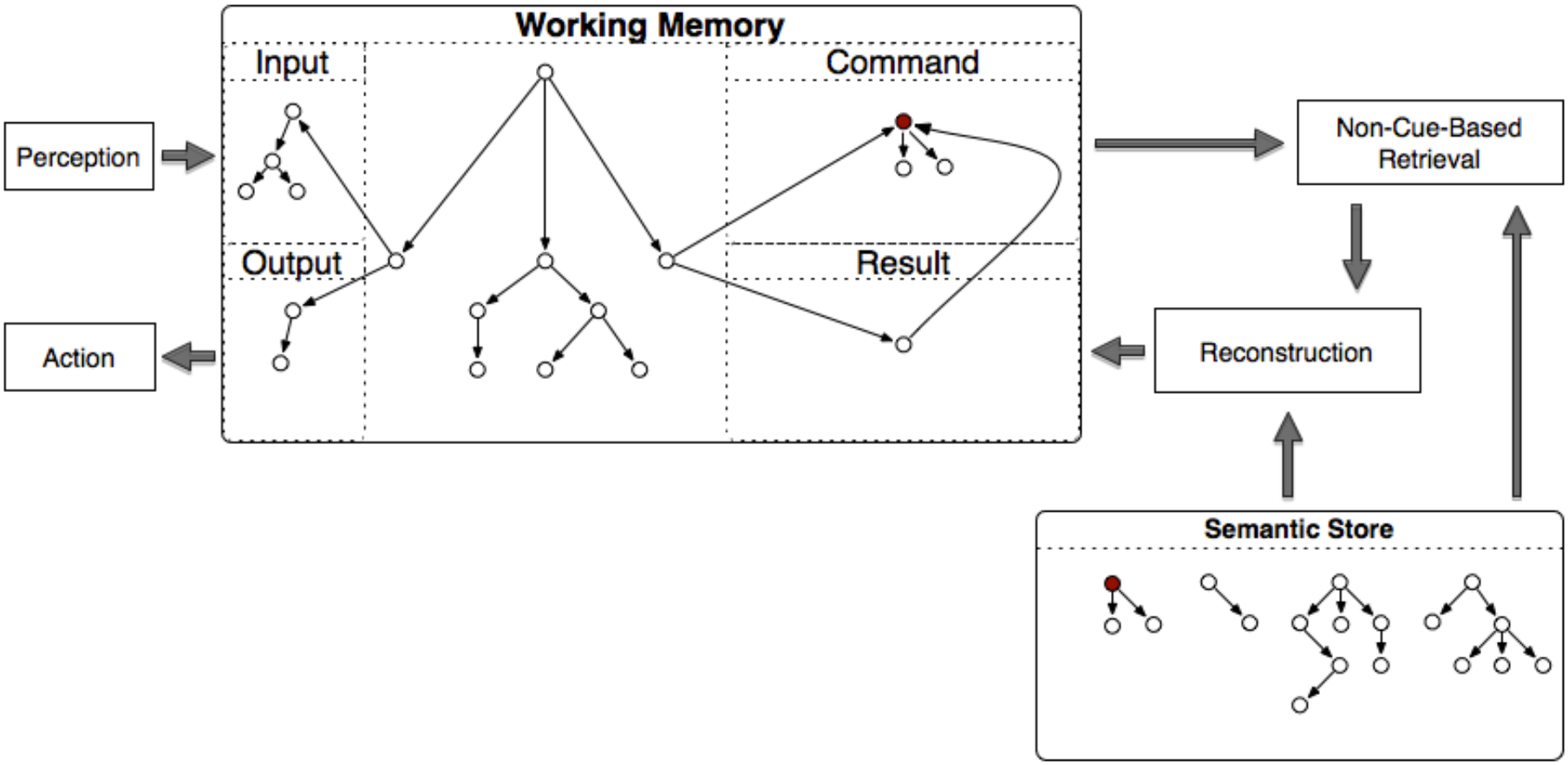
Non-Cue-Based Retrieval





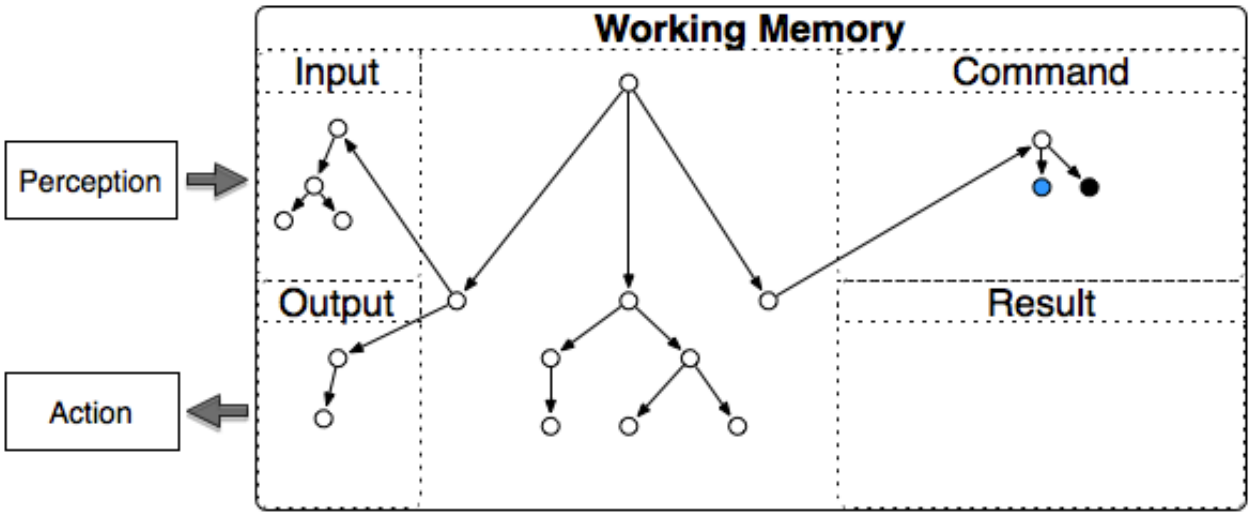
Architectural Integration

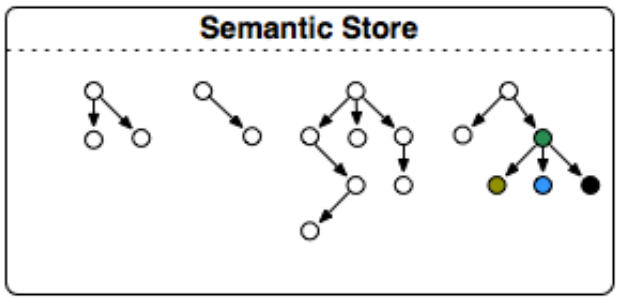
Non-Cue-Based Retrieval



Architectural Integration

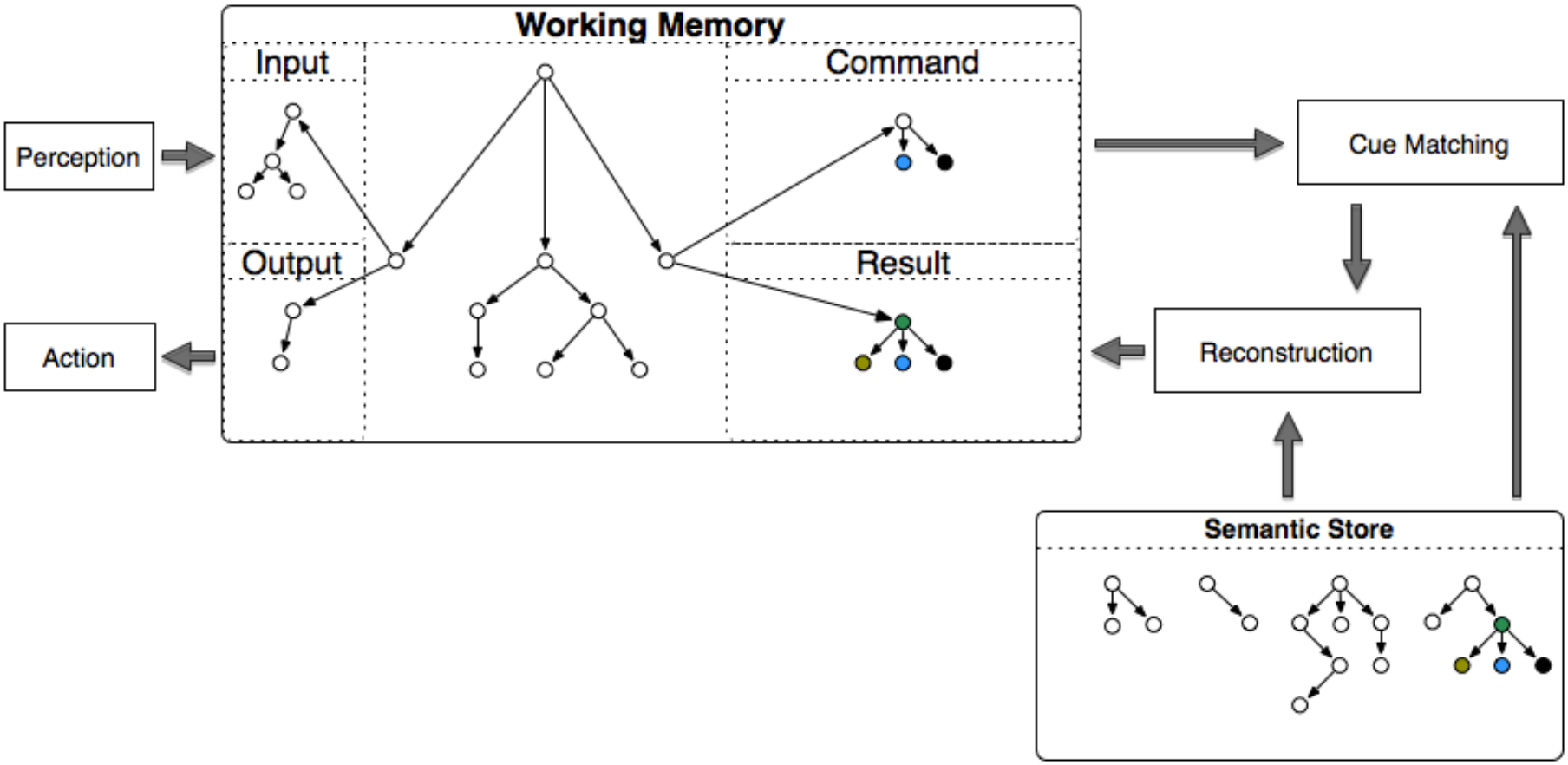
Cue-Based Retrieval





Architectural Integration

Cue-Based Retrieval



Basic Usage

- Working-memory structure
- Semantic-memory representation
- Controlling semantic memory
- Storing knowledge
- Retrieving knowledge

Working-Memory Structure

Soar creates an `smem` structure on each state

- Soar Java Debugger
  - `step 5`
  - `print --exact (* ^smem *)`
  - `print s2`

Each `smem` structure has specialized substructure

- `command`: agent-initiated actions
- `result`: architectural feedback

Semantic-Memory Representation

Similar to working memory: symbolic triples

- All identifiers in semantic memory are *long-term*
  - The letter-number pair (e.g. S5 or C7) is permanently associated with the identifier
  - When printed, long-term identifiers are prefaced with the @ symbol (e.g. @S5 or @C7)
  - When depicted, long-term identifiers are double circles
- Attributes cannot be identifiers (currently)
- The resulting graph is not necessarily connected

Controlling Semantic Memory

Get/Set a parameter:

- `smem [-g|--get] <name>`
- `smem [-s|--set] <name> <value>`

SMem is **disabled** by default. To enable it…

1. `smem`
2. `smem --set learning on`
3. `smem`

Storing Knowledge

Manual

Method of appending via command line (especially useful for loading external KBs)

Agent

Deliberate (via rules) addition/modification

Note: both methods can change existing and/or add new knowledge in semantic memory.

Manual Storage

Syntax: similar to production RHS

```
smem --add {
   (<id1> ^attr1 val1 val2 ^attr2 val1 … )
   (<id2> ^attr3 <id1> val5 … )
   (<id3> ^attr4.attr5 <id3>)
   …
}
```

Manual Storage: Example

Soar Java Debugger

1. `smem --add {`

```
(<a> ^name alice ^friend <b>)
(<b> ^name bob ^friend <a> <c>)
(<c> ^name charley)}
```

2. `smem --print`

```
(@A1 ^friend @B1 ^name alice [+1.000])
(@B1 ^friend @A1 @C4 ^name bob [+2.000])
(@C4 ^name charley [+3.000])
```

Agent Storage

Syntax

```
(<smem> ^command <cmd>)
(<cmd> ^store <id1> <id2> …)
```

- Requires that SMem is enabled (slide 16)
- Processed at end of phase in which rule fires
- Multiple identifiers may be stored at once
- Storage is **not** recursive

Result

```
(<smem> ^command <cmd> ^result <r>)
(<cmd> ^store <id1> <id2> …)
(<r> ^success <id1> <id2> …)
```

Agent Storage: Example

- Soar Java Debugger
  1. `smem --set learning on`
  2. `watch 5`
  3. `source`
  4. `run 4 -p`
  5. `print --depth 10 s2`

```
sp {propose*init
   (state <s> ^superstate nil
              -^name)
-->
   (<s> ^operator <op> +)
   (<op> ^name init)
}
```

```
sp {apply*init
   (state <s> ^operator.name init
              ^smem.command <cmd>)
-->
   (<s> ^name friends)
   (<cmd> ^store <a> <b> <c>)
   (<a> ^name alice ^friend <b>)
   (<b> ^name bob ^friend <a> <c>)
   (<c> ^name charley)
}
```

Examining the Trace

```
=>WM: (30: C4 ^name charley)
=>WM: (29: B1 ^friend A1)
=>WM: (28: B1 ^friend C4)
=>WM: (27: B1 ^name bob)
=>WM: (26: A1 ^friend B1)
=>WM: (25: A1 ^name alice)
=>WM: (24: C2 ^store A1)
=>WM: (23: C2 ^store B1)
=>WM: (22: C2 ^store C4)
=>WM: (21: S1 ^name friends)
--- Change Working Memory (PE) ---
=>WM: (33: R3 ^success @A1)
=>WM: (32: R3 ^success @B1)
=>WM: (31: R3 ^success @C4)
```

Semantic-Store Statistics

`smem --stats`

- `Nodes`: number of long-term identifiers
- `Edges`: number of features/relations
- `Stores`: number of agent stores

Retrieving Knowledge

Non-Cue-Based

Add the features/relations of a known long-term identifier to working memory

Cue-Based

Find a long-term identifier that has a set of features/relations and add it to working memory with its full feature/relation set

Common Constraints:

- Requires that SMem is enabled (slide 16)
- Only one per state per decision
- Processed during *output* phase
- Only re-processed if WM changes to commands
  - Meta-data (status, etc) automatically cleaned by the architecture

Non-Cue-Based Retrieval

Syntax

```
(<smem> ^command <cmd>)
(<cmd> ^retrieve <long-term identifier>)
```

Result

```
(<smem> ^command <cmd> ^result <r>)
(<cmd> ^retrieve <long-term identifier>)
(<r> ^<status> <long-term identifier>
     ^retrieved <long-term identifier>)
```

Where `<status>` is…

- `failure`: `<long-term identifier>` is not long-term
- `success`: else (adds all features/relations to WM)

Non-Cue-Based Retrieval: Example

- Soar Java Debugger
  1. `smem --set learning on`
  2.
```
smem --add {
   (@A1 ^name alice ^friend @B1)
   (@B1 ^name bob ^friend @A1 @C4)
   (@C4 ^name charley)}
```
  3.
```
sp {ncb
   (state <s> ^superstate nil
              ^smem.command <cmd>)
  -->
   (<cmd> ^retrieve @B1)}
```
  4. `run 5 -p`
  5. `print --depth 10 s2`
  6. `smem --stats`

Non-Cue-Based Retrieval: Debrief

- Be cautious of long-term identifiers in rules
  - Only legal if already in semantic store
  - Will occur via chunking
- Only features/relations of @B1 added to WM
  - Features/relations of @A1, @C4 would require additional `retrieve` commands
- Statistics kept about number of `retrieve` commands processed
  - `smem --stats`
    - (“`Retrieves`”)
- Meta-data cleaned up during *output* phase

Cue-Based Retrieval: Syntax

```
(<smem> ^command <cmd>)
(<cmd> ^query <q>)
(<q> ^attr1 val1
     ^attr2 <val2>
     ^attr3 @V3 …)
```

The augmentations of the *query* form hard constraint(s), based upon the value type…

- Constant: exact match
- Long-Term ID: exact match
- Short-Term ID: wildcard

Cue-Based Retrieval: Result

```
(<smem> ^command <cmd> ^result <r>)
(<cmd> ^query <q>)
(<r> ^<status> <q>
     ^retrieved <long-term identifier>)
```

Where `<status>` is…

- `failure`: no long-term identifier satisfies the constraints
- `success`: else (adds all features/relations to WM)

Ties are broken by a bias (default: recency)

- See `activation-mode` parameter in Manual
- When you execute `smem -p`, the bias value is indicated

Cue-Based Retrieval: Example

- Soar Java Debugger

1. `smem --set learning on`
2.
```
smem --add {
   (@A1 ^name alice ^friend @B1)
   (@B1 ^name bob ^friend @A1 @C4)
   (@C4 ^name charley)}
```
3.
```
sp {cbr
   (state <s> ^superstate nil
              ^smem.command <cmd>)
  -->
   (<cmd> ^query.name alice)}
```
4. `run 5 -p`
5. `print --depth 10 s2`
6. `smem --stats`

Prohibition

Cue-based retrievals can optionally prohibit the retrieval of one-or-more long-term identifiers

Syntax

```
(<smem> ^command <cmd>)
(<cmd> ^prohibit <lti-1> <lti-2> …)
```

Prohibition: Example

- Soar Java Debugger

1. `smem --set learning on`
2.
```
smem --add {
   (@A1 ^name alice ^friend @B1)
   (@B1 ^name bob ^friend @A1 @C4)
   (@C4 ^name charley)}
```
3.
```
sp {prohibit
   (state <s> ^superstate nil
              ^smem.command <cmd>)
  -->
   (<cmd> ^query.name <some-name>
          ^prohibit @A1 @C4)}
```
4. `run 5 -p`
5. `print --depth 10 s2`

Also Useful: Manual Query

```
smem -q {(<cue> ^name charley)}
```

WordNet Demo

https://github.com/SoarGroup/Domains-WordNet

- Scripts to convert WN-LEXICAL to SMem
  - Output: `smem --add { …`
    - >821K long-term identifiers, >3.97M edges, ~88MB
  - Source: ~5-10 minutes, ~1GB memory

- SMem uses a SQLite backend
  - Has the ability to save semantic stores to disk and use disk-based databases
    - smem --backup <filename>

WordNet: Make Disk Store

- Soar Java Debugger
  - `source wn.soar`
    - ~5-10 minutes
  - `smem --stats`
  - `smem --backup path/to/filename.db`
    - ~10 seconds

- Soar Java Debugger
  - `smem --set path path/to/filename.db`
  - `smem --set database file`
  - `smem --stats`

WordNet: Representation

“sense” of the “verb” to “soar”

```
smem -q {(<c> ^isa s ^ss-type v ^word soar)}

(@S194181 ^isa s ^sense-number 4
          ^ss-type v ^synset-id 200155406
          ^tag-count 1 ^w-num 1
          ^word soar ^word-lower soar [+194177.000])
```

“gloss” with the “synset-id” 200155406

```
smem -q {(<c> ^isa g ^synset-id 200155406)}

(@G270
    ^gloss |go or move upward; 'The stock market soared after the
cease-fire was announced'|
    ^isa g ^synset-id 200155406 [+425386.000])
```

WordNet Task

wn-senses.soar

Find all definitions, given lexical word/POS

- High-level algorithm
  1. `query:^isa s ^word lex ^ss-type pos`
  2. If successful
     a) `query: ^isa g ^synset-id <sense ^synset-id>`
     b) If successful
        » `write <gloss ^gloss>`
     c) `prohibit: <sense>`
     d) Loop
  3. Else
     a) `(halt)`

Eaters!

Additional Resources

- Documentation
- Readings

Documentation

See the Soar Manual and Tutorial

Additional Topics

- Details of integration with other mechanisms
- Retrieval biases
- Performance
- Usage: commands, parameters, statistics, etc.
- …

Select Readings

http://soar.eecs.umich.edu/Soar-RelatedResearch

2006

- Integrating Semantic Memory into a Cognitive Architecture
  - Yongjia Wang, John E. Laird (Technical Report)

2010

- Extending Soar with Dissociated Symbolic Memories
  - Nate Derbinsky, John E. Laird (AISB)
- Towards Efficiently Supporting Large Symbolic Memories
  - Nate Derbinsky, John E. Laird (ICCM)

2011

- Performance Evaluation of Declarative Memory Systems in Soar
  - John E. Laird, Nate Derbinsky, Jon Voigt (BRIMS)
- A Functional Analysis of Historical Memory Retrieval Bias in the Word Sense Disambiguation Task.
  - Nate Derbinsky, John E. Laird (AAAI)

2012

- Functional Interactions between Memory and Recognition Judgments
  - Justin Li, Nate Derbinsky, John E. Laird (AAAI)

2014

- A Case Study of Knowledge Integration Across Multiple Memories in Soar
  - John E. Laird, Shiwali Mohan (BICA)