Knowledge Representation and Reasoning

University "Politehnica" of Bucharest

Department of Computer Science

Fall 2012

Adina Magda Florea

Lecture 11

Soar: an architecture for human cognition

Lecture outline

- About models of cognition
- Soar – a cognitive architecture
- Soar evolution
- Basic Soar architecture
- Soar processing cycle
- Extended Soar architecture

1. About models of cognition

- Cognitive science - Christopher Longuet-Higgins, 1973
  - Comprises psychology, artificial intelligence, philosophy, neuroscience, linguistics, anthropology, sociology, biology, and other
  - Problems: memory organization, learning, decision mechanisms, logic, planning, neural networks, brain organization, etc.

About models of cognition

- Psycholinguistics – GPP
  - A semantic short circuit
  - to-and-fro understanding
  - Collin, Quillian – **the cognition economy principle**

❑ Without her contributions we failed

❑ Without her contributions failed to come in

❑The old man saw the boat

❑The old man the oars

About models of cognition

- Psychology - behaviors that people seem to exhibit
  - Steinberg item recognition paradigm
  - the time to decide whether a test item was on a memorized list of items increases linearly with the length of the list

2. Soar – a cognitive architecture

- Each individual discipline contributes with what Newell called *microtheories* — small pieces of the big picture
- Newell proposed in 1990 the **unified theories of cognition** (UTCs)
- Soar as a candidate UTC
- **A GENTLE INTRODUCTION TO SOAR, AN ARCHITECTURE FOR HUMAN COGNITION: 2006 UPDATE**
- JILL FAIN LEHMAN, JOHN LAIRD, PAUL ROSENBLOOM

THE IDEA OF ARCHITECTURE

- We can think of a software application as having an architecture
  - a particular fixed set of mechanisms and structures that define a set of functionalities
- BEHAVIOR = ARCHITECTURE + CONTENT
- Soar - a ***cognitive architecture*** seen as a <u>theory</u> of the <u>mechanisms</u> and <u>structures</u> that underlie human cognition

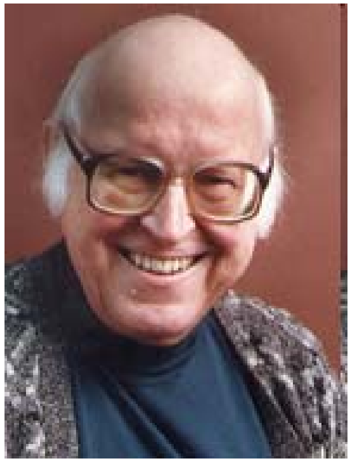
Physical symbol system hypothesis

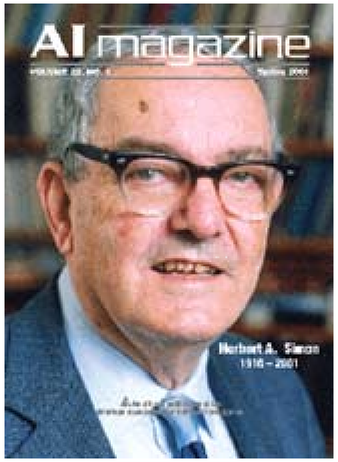
- The Soar architecture is based on the view that a symbolic system is necessary and sufficient for general intelligence
- A **physical symbol system** takes:
  - physical patterns (symbols)
  - combines them into structures (expressions)
  - manipulates them (using processes) to produce new expressions

Physical symbol system hypothesis

- "A physical symbol system has the necessary and sufficient means for general intelligent action."

Alan Newell and Herbert Simon





Cognitive architecture

- A cognitive architecture consists of:
  - memories for storing knowledge
  - processing units that extract, select, combine, and store knowledge
  - languages for representing the knowledge that is stored and processed
- A cognitive architecture distinguishes between:
  - knowledge that is acquired over time and
  - the fixed cognitive architecture that is common across all tasks.

3. Soar evolution

- Soar architecture - substantial evolution and refinement, with 8 major versions between 1982 and 2007.
- The basic approach of pure symbolic processing
- Long-term knowledge represented as production rules
- A general and flexible architecture for research in cognitive modeling: behavioral and learning phenomena.
- Knowledge-rich agents that could generate diverse, intelligent behavior in complex, dynamic environments

Soar evolution

- Soar was missing some important capabilities that we take for granted in humans
- Added:
  - New learning mechanisms
  - New long-term memories
  - New forms of non-symbolic processing

Soar evolution

- First version operational in 1982
  - Written by Allen Newell, John Laird, and Paul Rosenbloom at CMU.
- Versions 1-5 written in Lisp.
- Version 6 written in C.
- Version 7 written in C with Tcl/Tk.
- Version 7.0.4 most commonly used.
- Version 7.2 true multi-platform version.
- Version 8.3/4 latest multi-platform versions.

Soar – State Operator ARchitecture

- Function: *intransitive verb*
- Date: 14th century
- **1 a :** to fly aloft or about *b (1)* **:** to sail or hover in the air often at a great height **:** glide *(2) of a glider* **:** to fly without engine power and without loss of altitude
  2 : to rise or increase dramatically (as in position, value, or price) <stocks soar*ed*>
  3 : to ascend to a higher or more exalted level <makes my spirits soar>
  4 : to rise to majestic stature
- Function: *noun*
- Date: 1596
- **1 :** the range, distance, or height attained in soaring
  2 : the act of soaring **:** upward flight

4. Basic Soar architecture

- A **rule based system**, although different
  - Use context dependent knowledge to influence decisions about rule selection
- Work on a full range of tasks;
- Represent and use appropriate forms of knowledge;
- Employ a full range of problem-solving methods;
- Interact with the outside world;
- Learn about the task and its performance.
- . . . .

Basic Soar architecture

- a *goal* - is a desired situation.
- a *state* - is a representation of a problem solving situation.
- a *problem space* - is a set of states and operators for the task.
- an *operator* - transforms the state by some action
- **Problem solving representation** - **States** and **Operators**, but
  - Operators are distributed across rules
  - Operators are represented as structures in WM

Basic Soar architecture

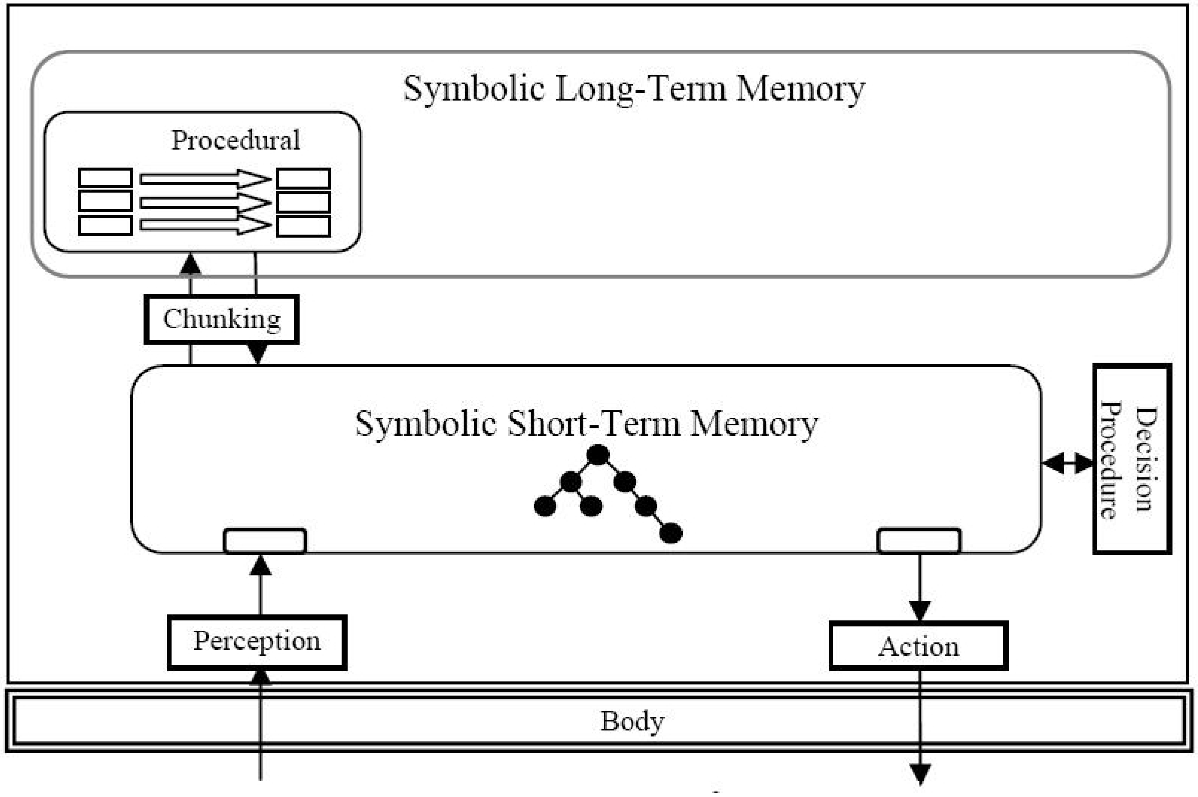
- Knowledge is encoded in *production rules.* A rule has *conditions* on its LHS, and *actions* on the RHS: C --> A

Rules

```
sp {rule*name (condition)(condition) …
--> (action) (action) …}
```

- Two memories are relevant here:
  - the *production memory* (PM), permanent knowledge in the form of production rules – Symbolic long term memory
  - *working memory* (WM), temporary information about the situation being dealt with, as a collection of *elements* (WMEs) – Symbolic short term memory

Structure of Soar



Rules in Soar

- The goal (or *desired*) state is to achieve

  (<s> ^hungry no).

- Operators are named Eat and Drink:
  - Eat can apply to any state with **(^hungry yes)**, and yields a new state with **(^hungry no)**
  - Drink can apply to any state with **(^thirsty yes)**, and yields a new state with **(^thirsty no)**.
  - Production rules offer knowledge about:
    - when to *propose* the operator
    - how to *compare* operators and *select* an operator
    - how to *apply* (or *"implement"*) the operator

Rules in Soar

```
sp {ht*propose-op*drink
   (state <s> ^problem-space.name hungry-thirsty
                ^thirsty yes)
  -->
   (<s> ^operator <o>)
   (<o> ^name drink)}
```

IF we are in the hungry-thirsty problem space, AND

in the current state we are thirsty

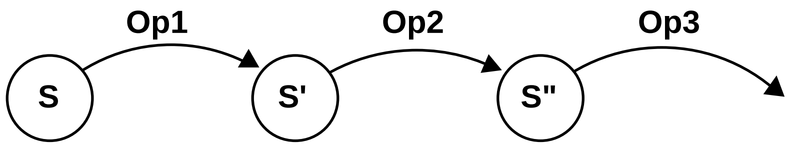
THEN propose an operator to apply to the current state,

and call this operator “drink”.

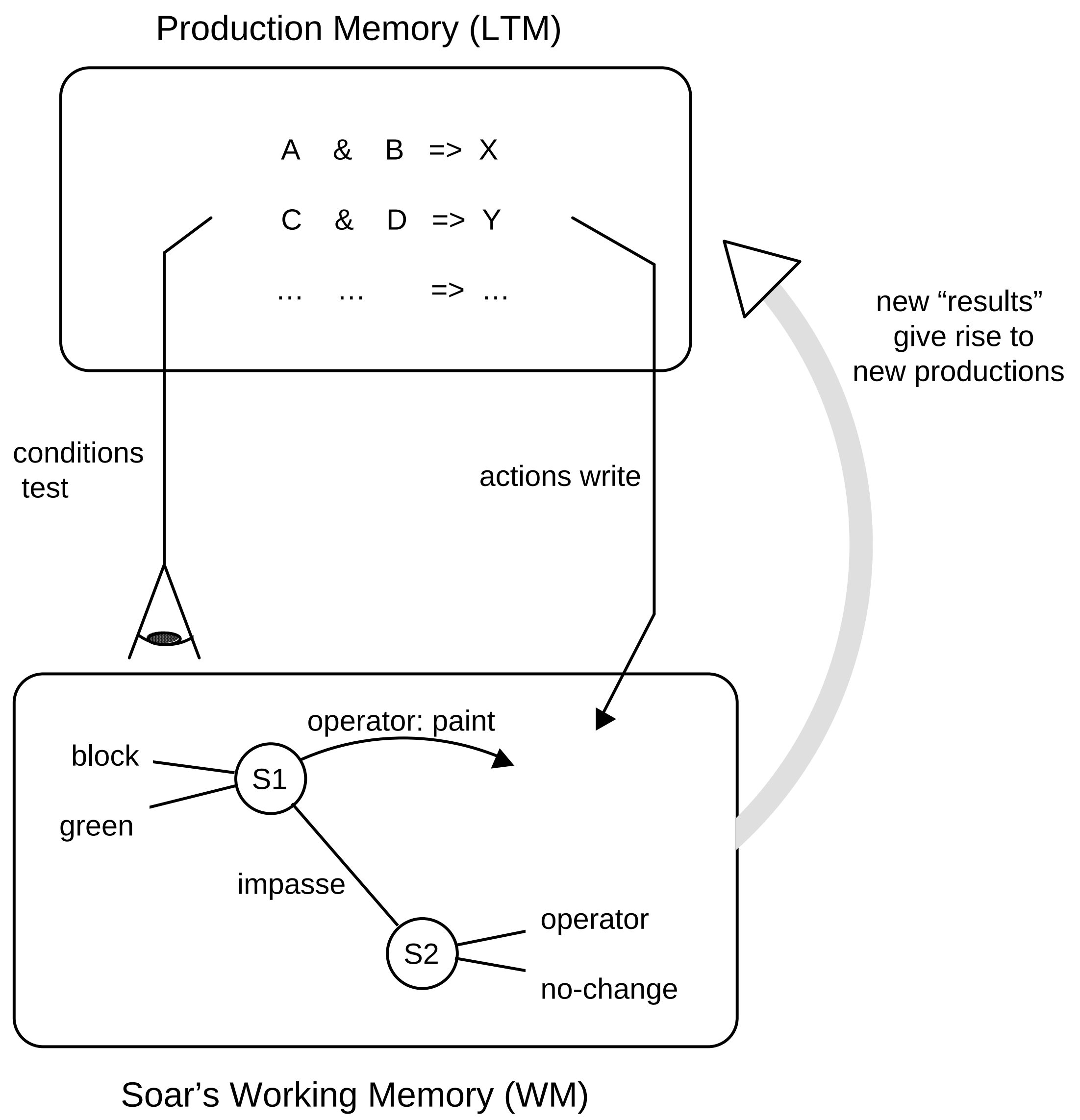
```
  sp {ht*propose-op*eat
     (state <s> ^problem-space.name hungry-thirsty
                 ^hungry yes)
  -->
   (<s> ^operator <o>)
   (<o> ^name eat)}
```

Basic Soar behaviour

- Behaviour is seen as occurring in a problem space, made up of States (S) and Operators (O or Op).
  (The implementation, however, has changed somewhat from Newell's 1990 book.)
- **Fluent behaviour** is a cycle in which an ***operator*** is selected, and is applied to the current state to give a new current state.



Rules and operators



Basic Soar behaviour

- The main activity is the repeated ***selection*** and then ***application*** of one operator after another.
- *If something prevents that process from continuing?*
  - Soar knows of no operators to apply to a given state
  - It knows of several, but has no knowledge of how to choose?
- In such cases, an ***impasse*** occurs, explained soon

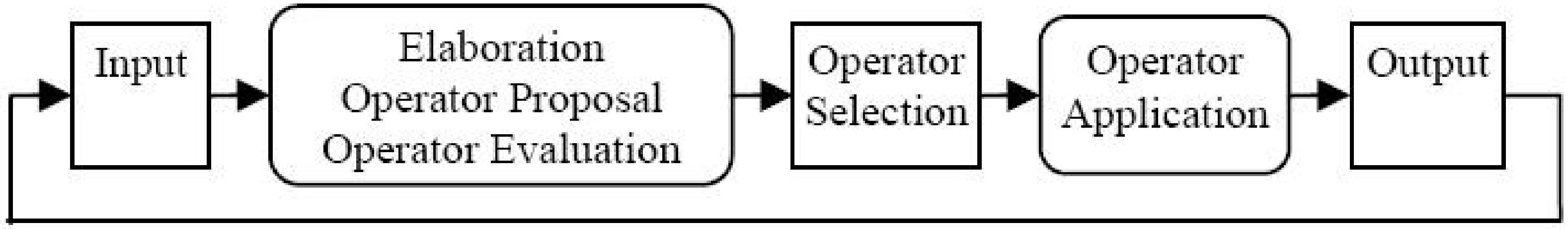
Rules and operators

- **Rules that *propose* operators** - create a data structure in WM representing the operator and an *acceptable preference* so that the operator can be considered for selection.
- **Rules that *evaluate* operators** and prefer one operator to another or provide some indication of the utility of the operator for the current situation
- **Rules that *apply* the operator** by making changes to WM that reflect the actions of the operator.
- Changes - internal or may initiate external actions in the environment

Rules and operators

- **Rules that *propose* operators**
- **Rules that *evaluate* operators**
- **Rules that *apply* the operator**
- make possible to incrementally build operator knowledge structures
- the definition of operators may change over time

5. Soar processing cycle



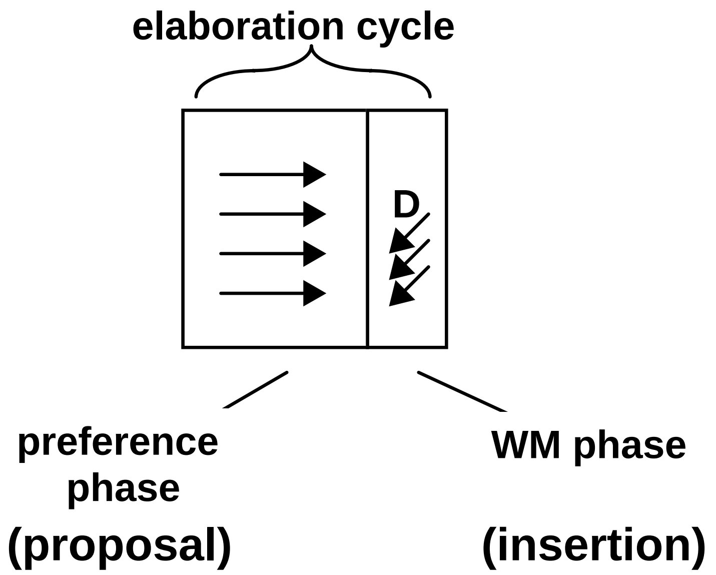
- **Input -** Changes to perception are processed and sent to short-term memory
- **Elaboration** - Rules compute entailments of short-term memory

Elaboration

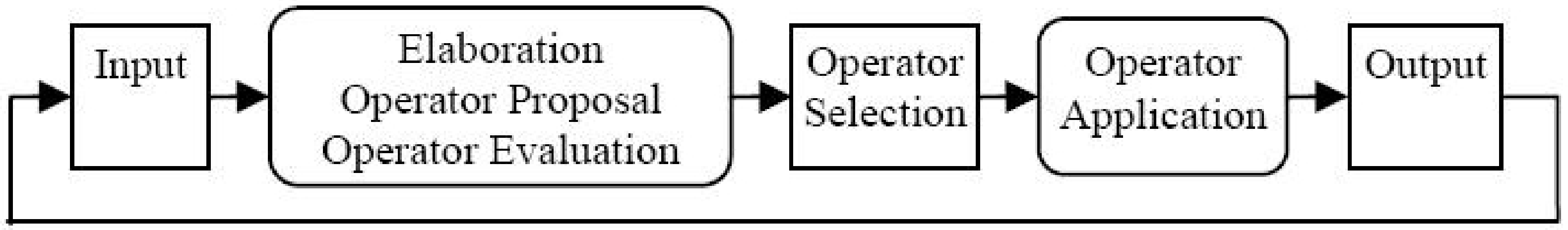
- Soar has no conflict resolution to decide on a single rule to fire at each cycle. Instead, all productions satisfied fire in parallel.
- The rule firings have the effect of changing WM so that more productions will have their conditions satisfied. So they fire next, again in parallel.
- This process of *elaboration cycles* continues until there are no more productions ready to fire, i.e. ***quiescence***

Elaboration

- Changes to WM are just **elaborations** – conclusions or suggestions in case of preferences, and not actions in the world or changes to internal structures that should persist
- Several operators are suggested as candidates for the next step



Soar processing cycle



- **Operator Proposal -** Rules propose operators that are appropriate to the current situation based on features of the situation tested in the condition of the rules.
- **Operator Evaluation** - Rules create preferences which of the proposed operators are to be preferred, based on the current situation and goal. The preferences can be symbolic (A is better than B), or numeric (the estimated utility of A is .73).

Proposal and evaluation

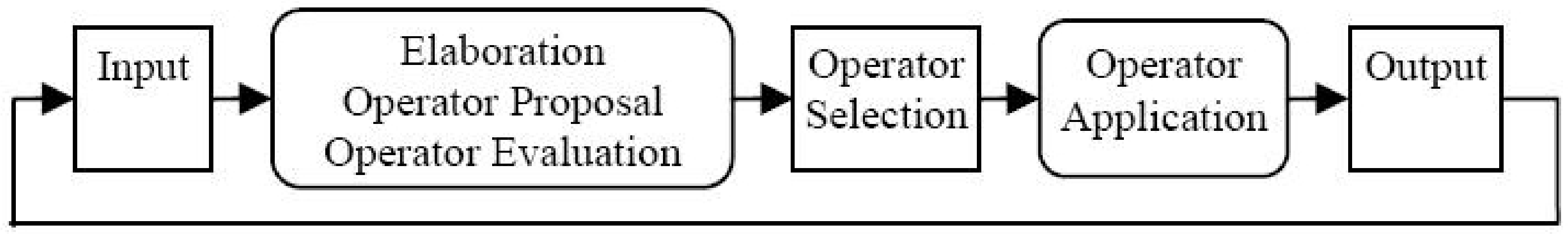
- With parallel rule firings there could be inconsistencies in WM
- Rules vote for changes to WM through ***preferences***
  - **+** *acceptable:* an object must have an acceptable to be considered.
  - **-** *reject:* the object cannot be considered.
  - **>** *better, best:* one object is better than another, or (unary) the object is best.
  - **<** *worse, worst:* similarly.
  - **=** *indifferent:* one object is indifferent to another, or (unary, and more usually) the object is indifferent.
  - **&** *parallel:* the object can be one of multiple values.

Memory

Soar has three memories:

- **Working memory -**
    - **holds knowledge about the current situation**
- **Production memory -**
    - **holds long-term knowledge in the form of rules**
- **Preference memory -**
    - **stores suggestions about changes to working memory**
        - **Allows Soar to reason about what it does**
        - **If it cannot, then Soar invokes a subgoal and learns about the result**

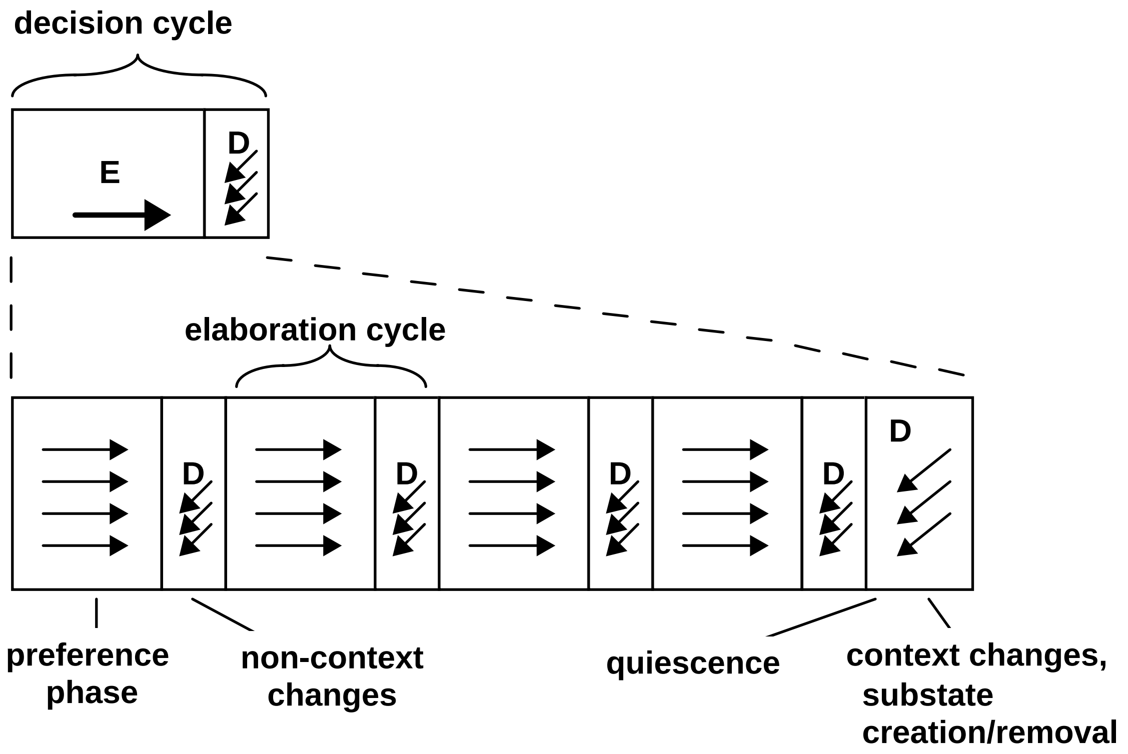
Soar processing cycle



- **Operator Selection** - A fixed decision procedure combines the generated preferences and selects the current operator
- If the preferences are insufficient for making a decision, an ***impasse*** arises and Soar automatically creates a **substate** in which the goal is *to resolve that impasse*

Impasse

- **Operator Selection**
- **Impasse** – an architectural event that arises whenever a decision procedure cannot resolve preferences in WM to select a new operator

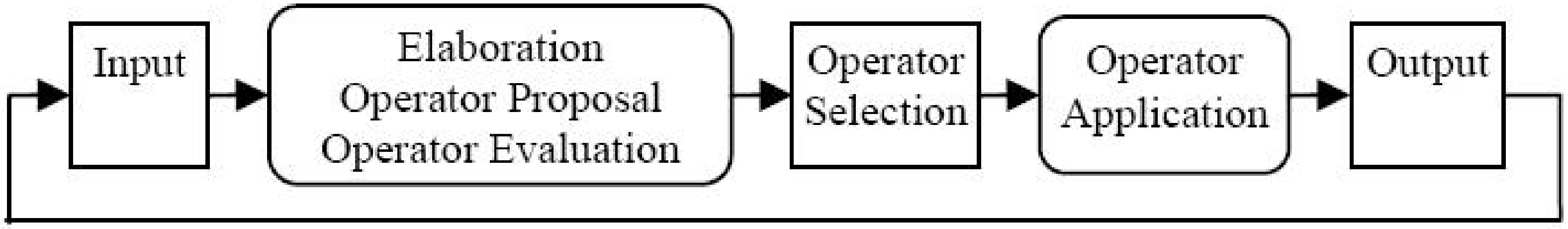


Impasse

- In a created **substate**, Soar recursively uses the same processing cycle to select and apply operators, leading to automatic, reactive meta-reasoning

- The impasses and resulting substates provide a mechanism for Soar to deliberately perform any of the functions (elaboration, proposal, evaluation, application) that are performed automatically/reactively with rules

Soar processing cycle



- **Operator Application** - The actions of an operator are performed by rules that match the current situation and the current operator structure
- **Output** - Any output commands are passed on to the motor system

Operator Application

- **Operator Application**
- If there is insufficient application knowledge, an **impasse** arises with a substate
- This leads to dynamic task decomposition, where a complex operator is performed recursively by simpler operators

More on impasses

There are **four types of impasses** in Soar.

- (1) **A state no-change impasse** - no operators are proposed for the current state
- (2) **An operator tie impasse** - there are multiple operators proposed, but insufficient preferences to select between them
- (3) **An operator conflict impasse** - there are multiple operators proposed, and the preferences conflict
- (4) **An operator no-change impasse** – an operator has been selected but there are no rules to apply it
- *(2) and (3) needs control knowledge*

More on impasses

- A state no-change impasse
  - Knowledge of how to implement the operator may be lacking, in which case that is what is needed
  - The preconditions of the operator may not be satisfied, which case requires *operator subgoaling*
  - The operator may be incompletely specified, and need to be augmented

More on impasses

- Whenever an impasse arises, Soar automatically creates a **new state** → provides a context for selecting and applying operators to resolve the impasse
- Once a substate is created, operators can be selected and applied just as they are in the original state
- These operators can change the local state, but they can also change the content of original state
- An impasse can also arise in the substate, which then leads to a stack of states.

More on impasses - **Chunking**

- Whenever **a result is returned from an impasse**, a new rule is learned connecting the relevant parts of the pre-impasse situation with the result
- The next time a sufficiently similar situation occurs, the impasse is avoided
- This leads to a uniform learning mechanism, called ***chunking***

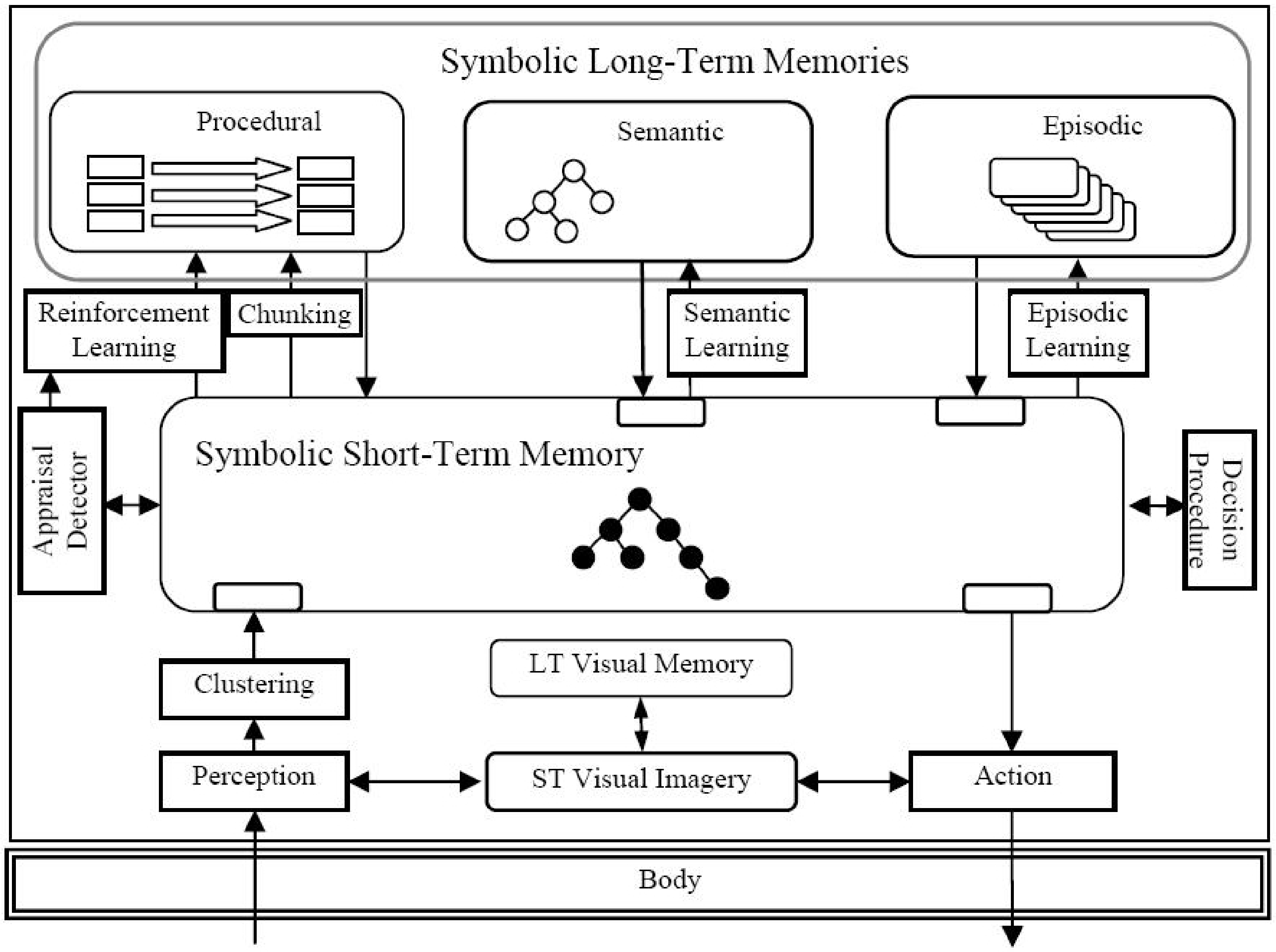
- *More on chunking – next lecture*

To do

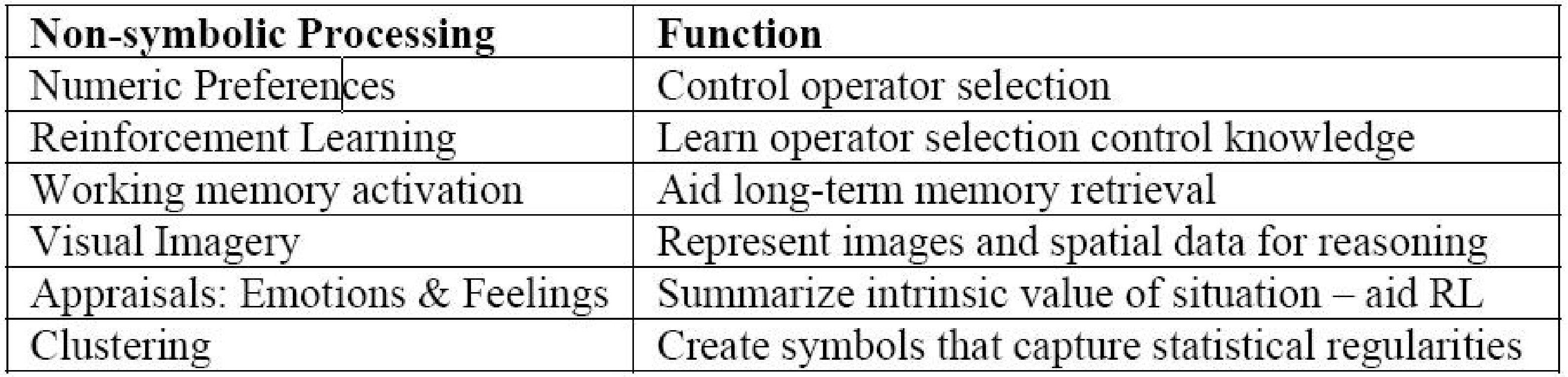
- Soar Tutorial Part 1: Hello World, Visual Soar and Water Jug problem

- http://ai.eecs.umich.edu/soar/sitemaker/docs/tutorial/Soar%20Tutorial%20Part%201.pdf

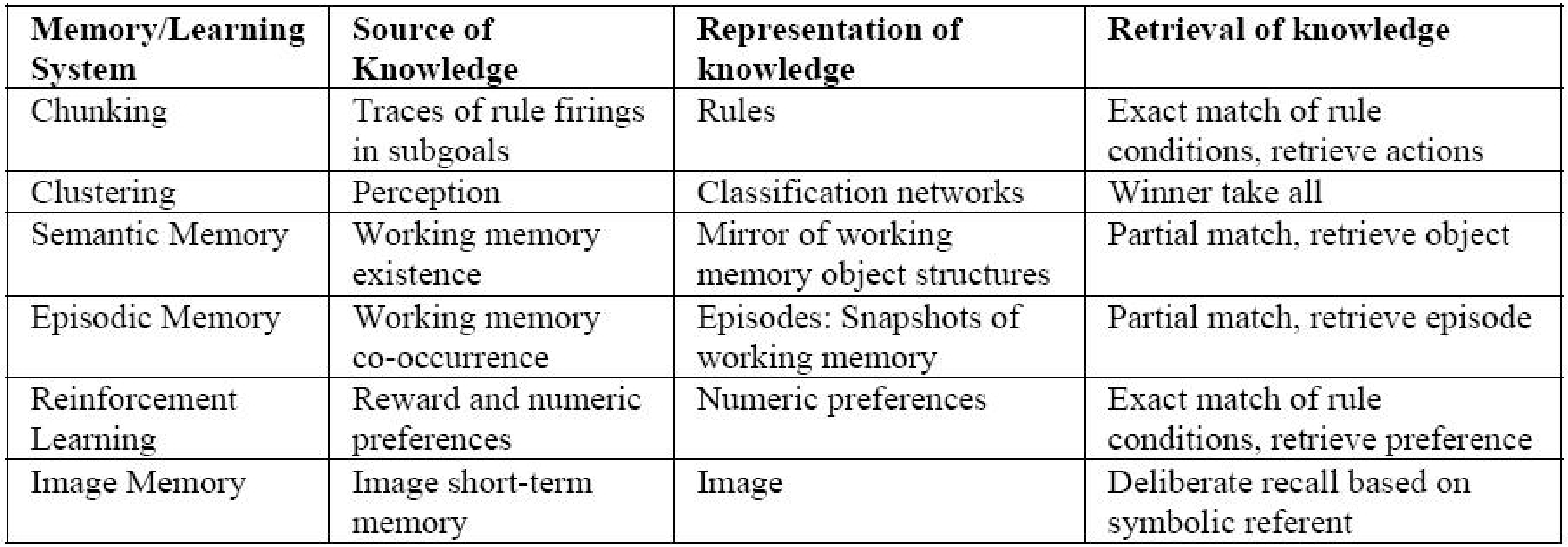
6. Extended Soar architecture



Non symbolic processing in extended Soar



Non-symbolic Processing	Function
Numeric Preferences	Control operator selection
Reinforcement Learning	Learn operator selection control knowledge
Working memory activation	Aid long-term memory retrieval
Visual Imagery	Represent images and spatial data for reasoning
Appraisals: Emotions & Feelings	Summarize intrinsic value of situation – aid RL
Clustering	Create symbols that capture statistical regularities



Memory/Learning System	Source of Knowledge	Representation of knowledge	Retrieval of knowledge
Chunking	Traces of rule firings in subgoals	Rules	Exact match of rule conditions, retrieve actions
Clustering	Perception	Classification networks	Winner take all
Semantic Memory	Working memory existence	Mirror of working memory object structures	Partial match, retrieve object
Episodic Memory	Working memory co-occurrence	Episodes: Snapshots of working memory	Partial match, retrieve episode
Reinforcement Learning	Reward and numeric preferences	Numeric preferences	Exact match of rule conditions, retrieve preference
Image Memory	Image short-term memory	Image	Deliberate recall based on symbolic referent