Models of Cognition and Learning

Andreas Wichert

DEIC

(Página da cadeira: Fenix)

- Unified Theories of Cognition and Learning
  - SOAR
    - ACT-R
  - Associative Computation
  - …

Unified Theories of Cognition

- Is a theory which attempts to unify all the theories of the *mind* in a single framework

- Allen Newell (1990) proposed that the current state-of-the-art in experimental psychology could now support such theories, based on years of accumulated results.

- To assert a unified theory of cognition, one must propose mechanisms by which the results of these human cognitive experiments can be reproduced

- Does the architecture make any attempt to model aspects of human behavior?

- Power Law of Learning:
  - “the logarithm of the reaction time for a particular task decreases linearly with the logarithm of the number of practice trials taken
  - With more practice at a task, people seem to always be getting faster
  - However, the rate of learning decreases the more practice one has

Cognitive architectures

- In cognitive science, the term refers to the architecture of the mind: a fixed structure underlying the flexible domain of cognitive processing.
  - Cognitive architectures – for humans
  - Architectures for intelligent agents – for agents
  - The ambition of SOAR is to be both: a basis for both human and artificial cognition

- Soar is based upon a theory of human problem solving…

  …in which...

- …all problem solving activity is formulated as the selection and application of operators to a state, to achieve some goal

Some Definitions

- 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 spaces

- Represents all tasks as collections of problem spaces
- Problem space:
  - States + operators that manipulate states

- The problem with a production system is the efficiency of matching a number of conditions against the knowledgebase.
- The RETE algorithm

Knowledge

- In order to act, Soar must have knowledge of that domain (either given to it or learned)
- Domain knowledge can be divided into two categories:
  - (a) basic problem space knowledge: definitions of the state representation, the “legal move” operators, their applicability conditions, and their effects
  - (b) control knowledge, which gives guidance on choosing what to do, such as heuristics for solving problems in the domain

Knowledge

- Given just the basic knowledge, Soar can proceed to search it
- But the search will be “unintelligent” (e.g. random or unguided depth first) — by definition it does not have the extra knowledge needed to do intelligent search
- Important basic knowledge centers round the operators:
  - when an operator is applicable
  - how to apply it
  - how to tell when it is done.

Memories

- 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

Memories

- 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.

Long term memory

- Soar's long term memory is a production system
- Productions:
  - have a set of conditions, which are patterns to be matched to working memory
  - a set of actions to perform when the production fires

Production Rules: Form of Knowledge

- Knowledge is encoded in *production rules.* A rule has *conditions* on its LHS, and *actions* on the RHS: C --> A.
- Two memories are relevant here:
  - the *production memory* (PM), permanent knowledge in the form of production rules
  - *working memory* (WM), temporary information about the situation being dealt with, as a collection of *elements* (WMEs).

What Do Rules Look Like?

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

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

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

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,

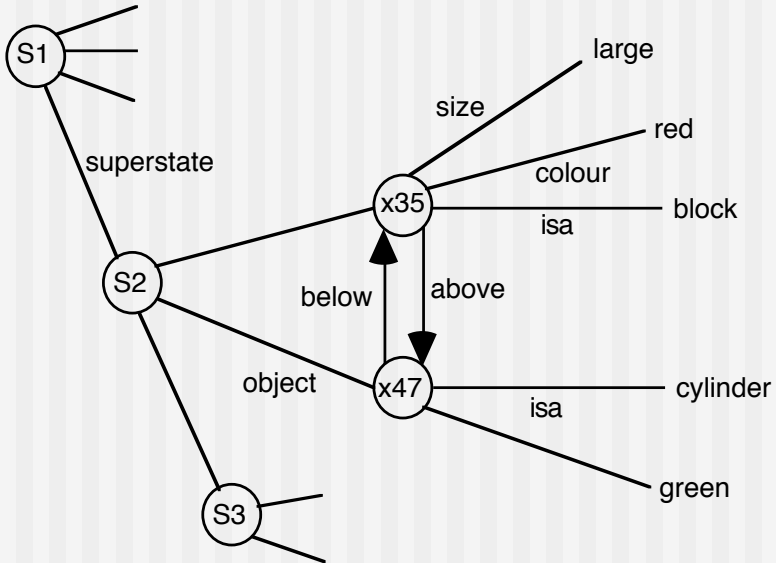
and call this operator “drink”

Concrete Representation

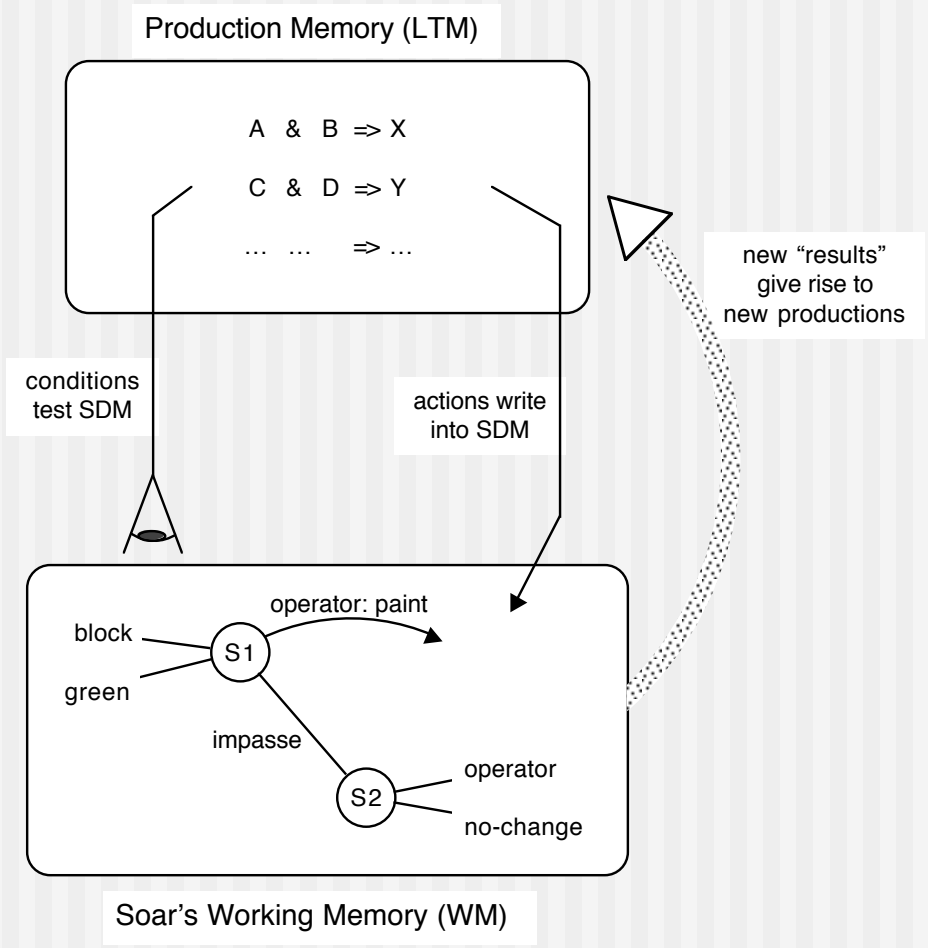
- Soar uses attribute-values to represent information. For example,
  - **(x35 ^isa block ^colour red ^size large)**
- Attributes commonly have a single value. (Theoretical & practical reasons for avoiding multiple values.)
- Operators are state attributes, at most one value:
  - **(s13 ^operator o51)**

Simple diagram of representation

- For informal discussion, it can be helpful to use a simple diagrammatic depiction of Soar objects in WM.
- The coloured blocks of the previous slide might be shown as:



Soar on a Slide



Working memory

- Soar's working memory consists of a set of (Object ^attribute value) elements
- Value may be a symbolic constant, a number, a string, or an Object
- The context stack consists of the context objects currently in working memory: all goals, problem spaces, states, and operators
- All context elements are attached to a goal, and all goals except the top goal point to a supergoal, imposing a linear order on the context stack.

- There must be some chain of elements from a context object to every element in working memory. If this chain is broken the element is removed from working memory
- The Decision Cycle can examine and modify the entire context stack
- The preference memory determines which elements enter and leave working memory

- Soar performs no conflict resolution between competing productions- all productions which match the current working memory fire

Decision cycle

- Soar works in a loop called decision cycle:
  - Elaboration
  - Decision
  - Repeat

- Elaboration:
  - all productions which match the current working memory fire
  - All productions fire in parallel
    - (until no more productions fire)

- Decision
  - examines any preferences put into preference memory
  - chooses the next problem space, state, operator or goal

Impasse

- If there is not enough information (or the information is contradictory) for the decision phase to choose the next slot value, then an impasse results

- There are four types of impasses:
  1. When two are more elements have equal preference, then there is a "tie impasse”
  2. When no preferences are in working memory, this causes a "no-change impasse“
  3. When the only preferences in working memory are rejected by other preferences, then there is a "reject impasse”
  4. A "conflict impasse" results when preferences claim that two or more elements are each better choices then the others

- Impasses occur when there is a lack of applicable knowledge in the current problem space,
- Problem solving needs to take place
  - This problem solving proceeds in the form of an automatically generated subgoal.

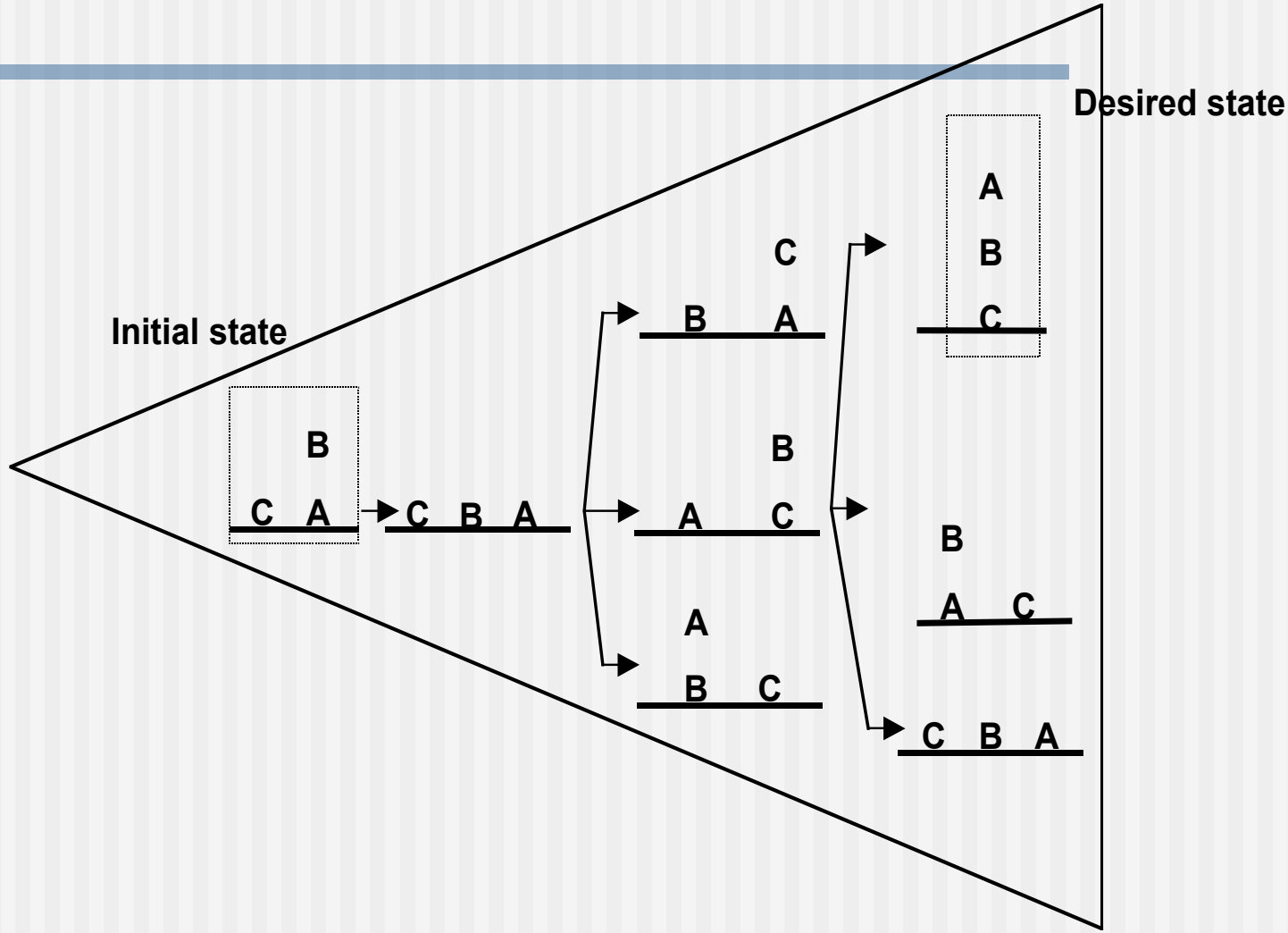
The Context Stack

- There can be several problem spaces (i.e. contexts) active at once
- Each may lack knowledge needed to continue. Then, Soar sees an *impasse* and automatically sets up a new context (or *substate*), whose purpose is to find the missing information and supply it to the context above.
- Each *decision cycle* ends with some kind of change to the context stack. If the knowledge available (i.e. in the productions) specifies a unique next operator, then that change is made. Otherwise, an impasse arises because the immediately applicable knowledge is insufficient to specify the change.

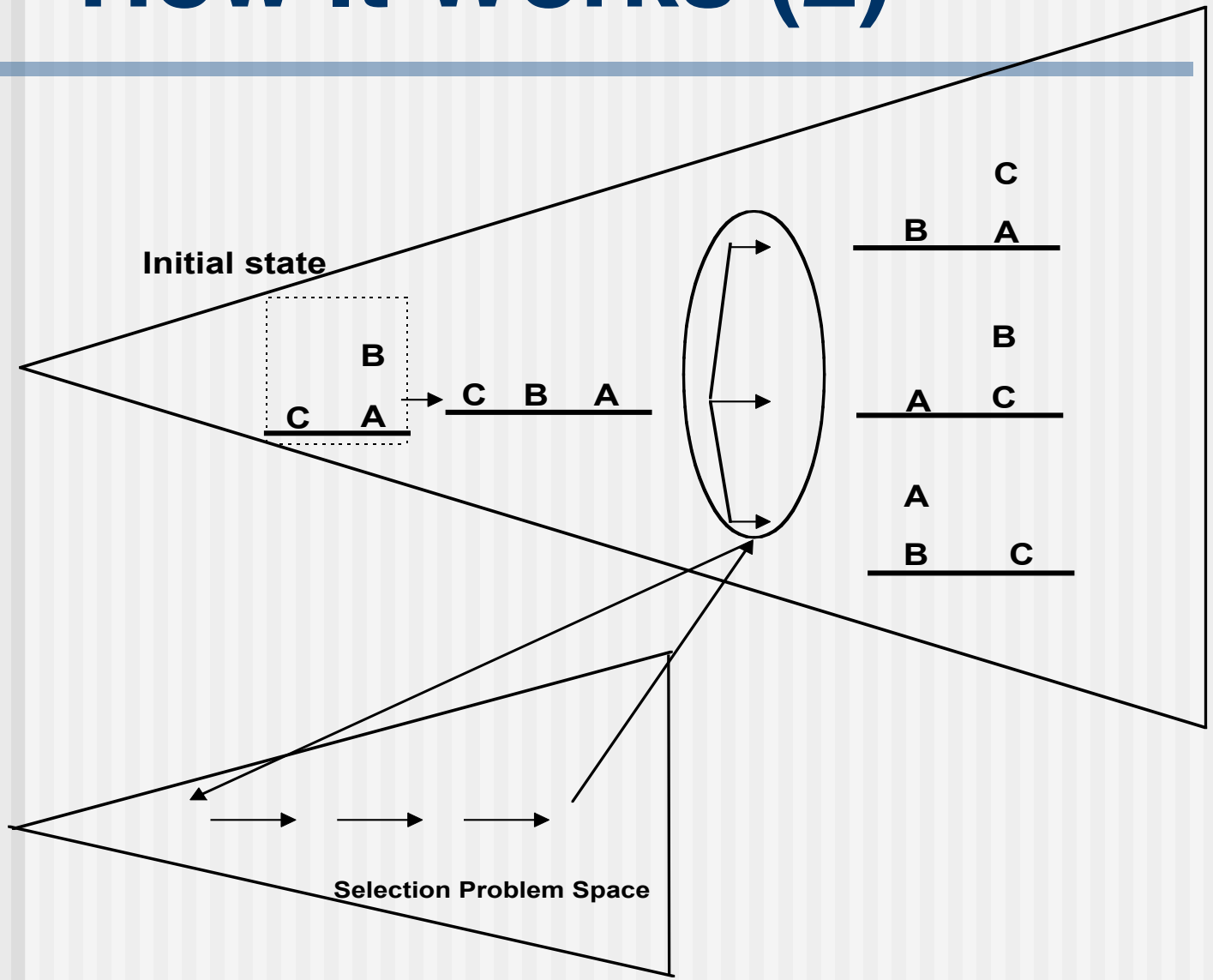
Impasses and Substates

- When Soar encounters an impasse at level-1, it sets up a substate at level-2, which has associated with it its own current state and operators
- The goal of the 2nd level is to find knowledge sufficient to resolve the higher impasse, allowing processing to resume there

How It Works (1)



How It Works (2)



Perceptual-Motor System

- The perceptual and motor subsystem consist of independant modules for each output channel. They run asynchronously with respect to each other and to the remainder of the architecture. The motor modules accept commands from working memory and execute them. Their progress can then be monitored through sensors that are fed back into the system via the perception subsystem
- The perceptual modules deliver data directly into working memory whenever it is available
- All perceptual and motor behavior is mediated through working memory

- Encoding and decoding production are used to convert between high-level structures use by the cognitive system, and the low-level structures used by the perceptual and motor subsystems
- These productions are the same as regular Soar production except that they match on perceptual and motor working memory elements and are independant of context (problem space, state, operator).

- This autonomy from context is critical because it allows the decision procedure to proceed without waiting for quiescense, which may not occur in a rapidly changing environment

Chunking-based Learning

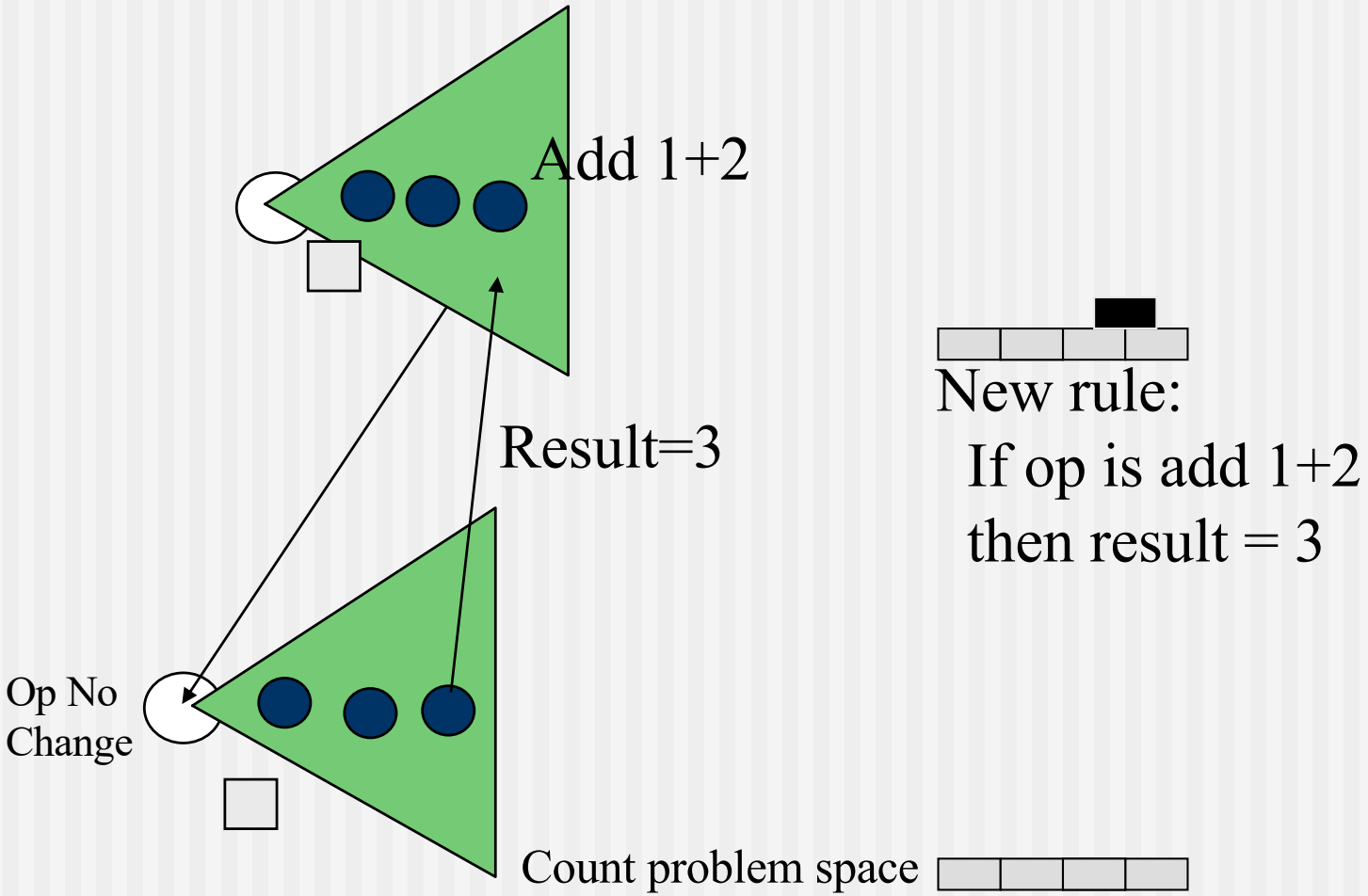
- Psychological phenomena of chunking:
  - The association of chunks (expressions or symbols) into a new, single chunk
  - In Soar, chunking collapses the results of an impasse into a production which can then be fired if the same, or similar situation occurs again, thus avoiding the impasse
  - This leads directly to Soar's ability to move from problematic to routine behavior
  - Since the learning mechanism creates new knowledge in the same form as the rest of the system's knowledge (i.e. productions), the uniformity of the representation is maintained by the learning mechanism

Learning

- Resolving an impasse leads to learning.
- The sole learning mechanism is called *chunking*.
- A chunk is a new production that summarises the processing that was needed to resolve the impasse.
- The chunk will be used in the future to avoid a similar impasse situation.

- The architecture directs the creation of a chunk whenever an impasse is resolved
- Chunking can be viewed as a caching mechanism

Soar, Learning in Action

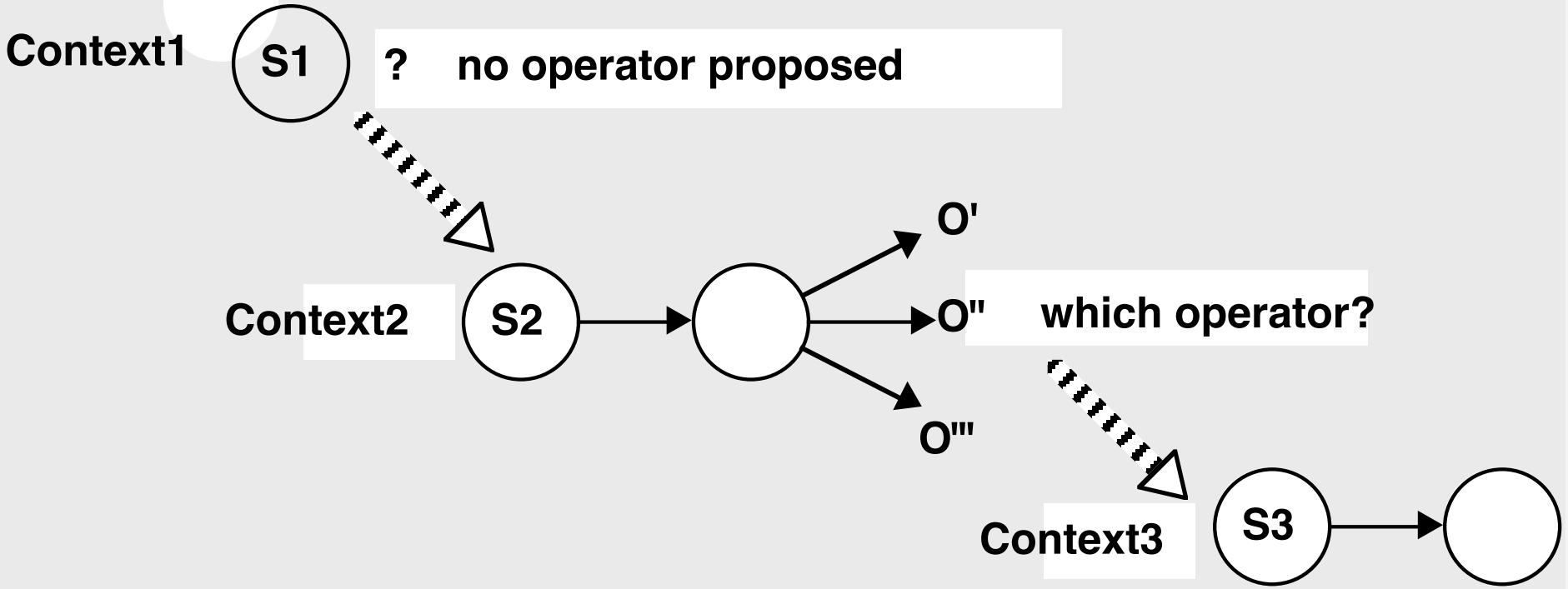


New rule:

If op is add 1+2

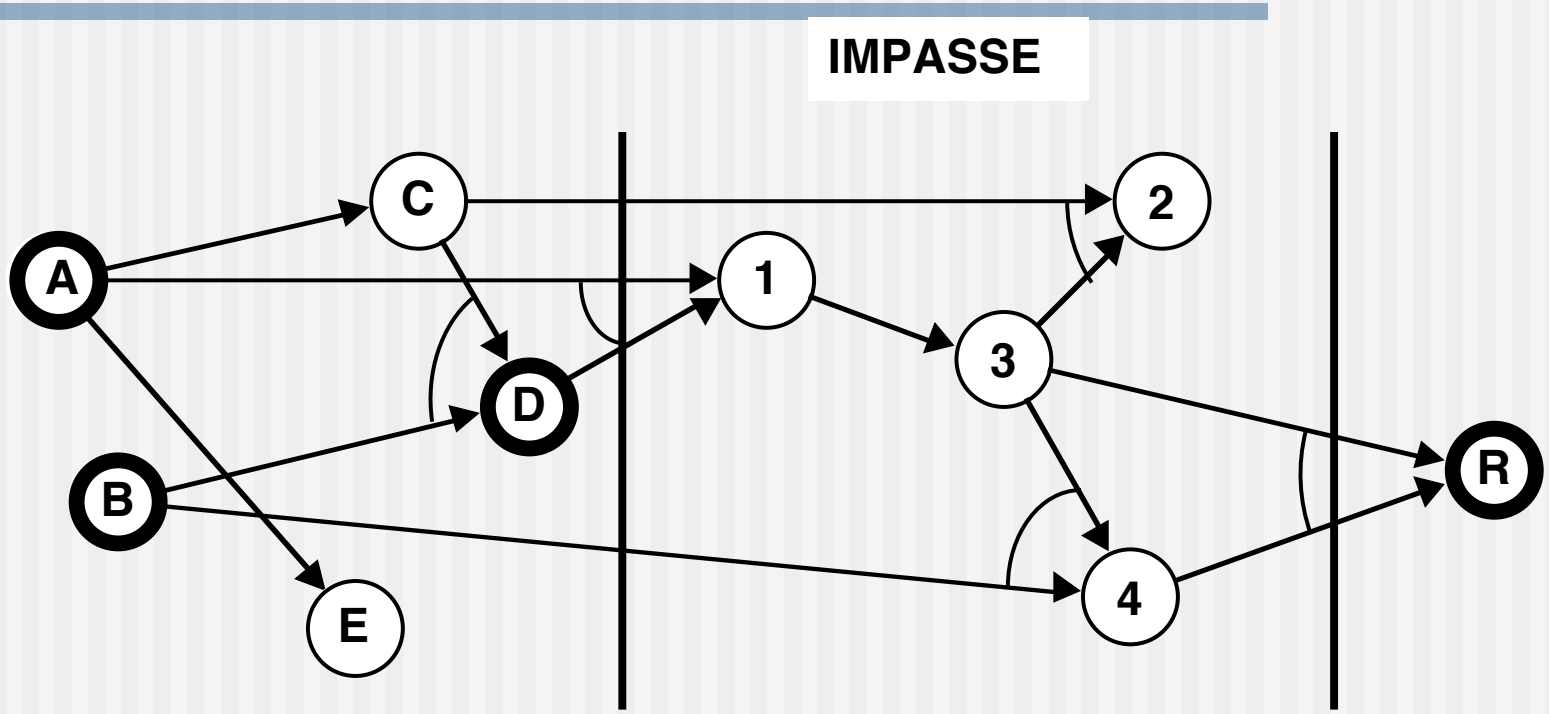
then result = 3

Impasses and Substates



Notice that the architecture's problem solving approach is applied recursively at each level.

Chunking 2: The backtrace



Chunk: A & B & D ⇒ R

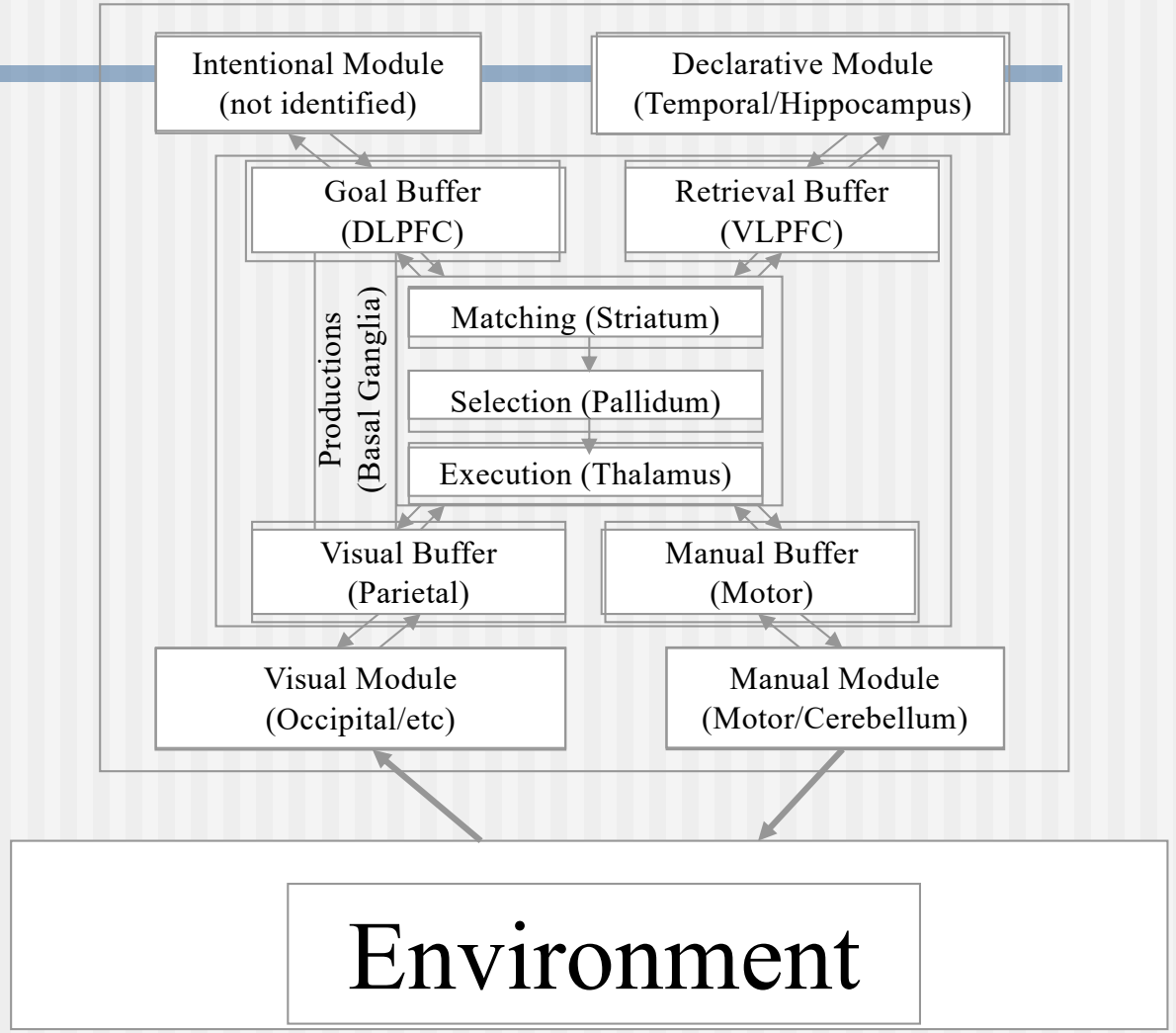
Circles are WMEs or sets of WMEs
Bold circles indicate nodes essential to the resolution
Arrow sets going into a node are rules that fire to add it
Numbered nodes are WMEs in the impasse

Implications

- Problem solving and chunking mechanisms are thus tightly intertwined: chunking depends on the problem solving, and most problem solving would not work without chunking.

- Even when no chunk is actually built (because learning off, or bottom up, or duplicate, or whatever), an internal chunk called a *justification* is formed

ACT-R 5.0


Intentional Module
(not identified)
Declarative Module
(Temporal/Hippocampus)
Goal Buffer
(DLPFC)
Retrieval Buffer
(VLPFC)
Productions
(Basal Ganglia)
Matching (Striatum)
Selection (Pallidum)
Execution (Thalamus)
Visual Buffer
(Parietal)
Manual Buffer
(Motor)
Visual Module
(Occipital/etc)
Manual Module
(Motor/Cerebellum)
Environment

Chunks: Example

(CHUNK-TYPE NAME SLOT1 SLOT2 SLOTN)

(FACT3+4

isa ADDITION-FACT

ADDEND1 THREE

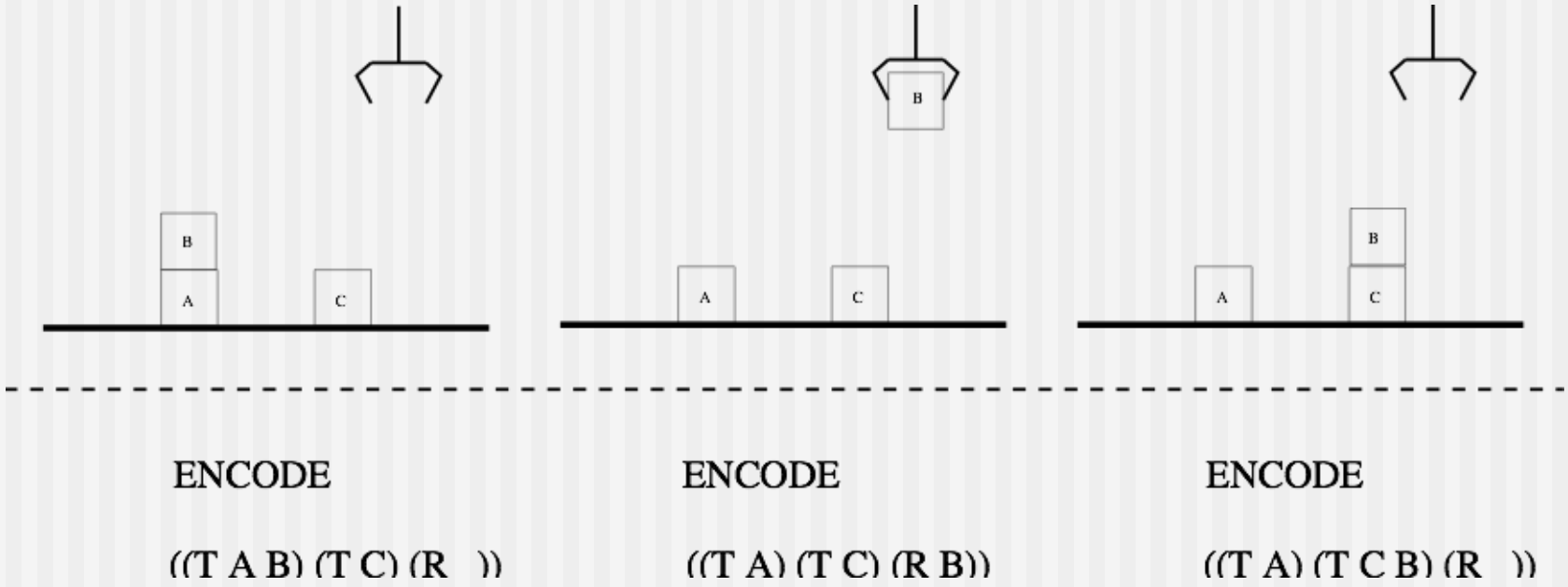
ADDEND2 FOUR

SUM SEVEN)

Knowledge

- Domain knowledge can be divided into two categories:
  - (a) basic problem space knowledge: definitions of the state representation, the “legal move” operators, their applicability conditions, and their effects
  - (b) control knowledge, which gives guidance on choosing what to do, such as heuristics for solving problems in the domain

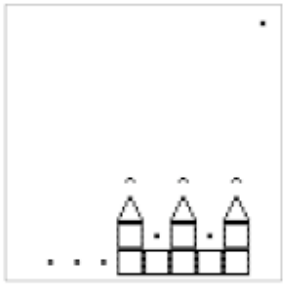
Representation



- To ease comprehension for a human reader, additional pictograms like two dimensional binary sketches are often used beside the symbolical representation

(a) (b) (c)

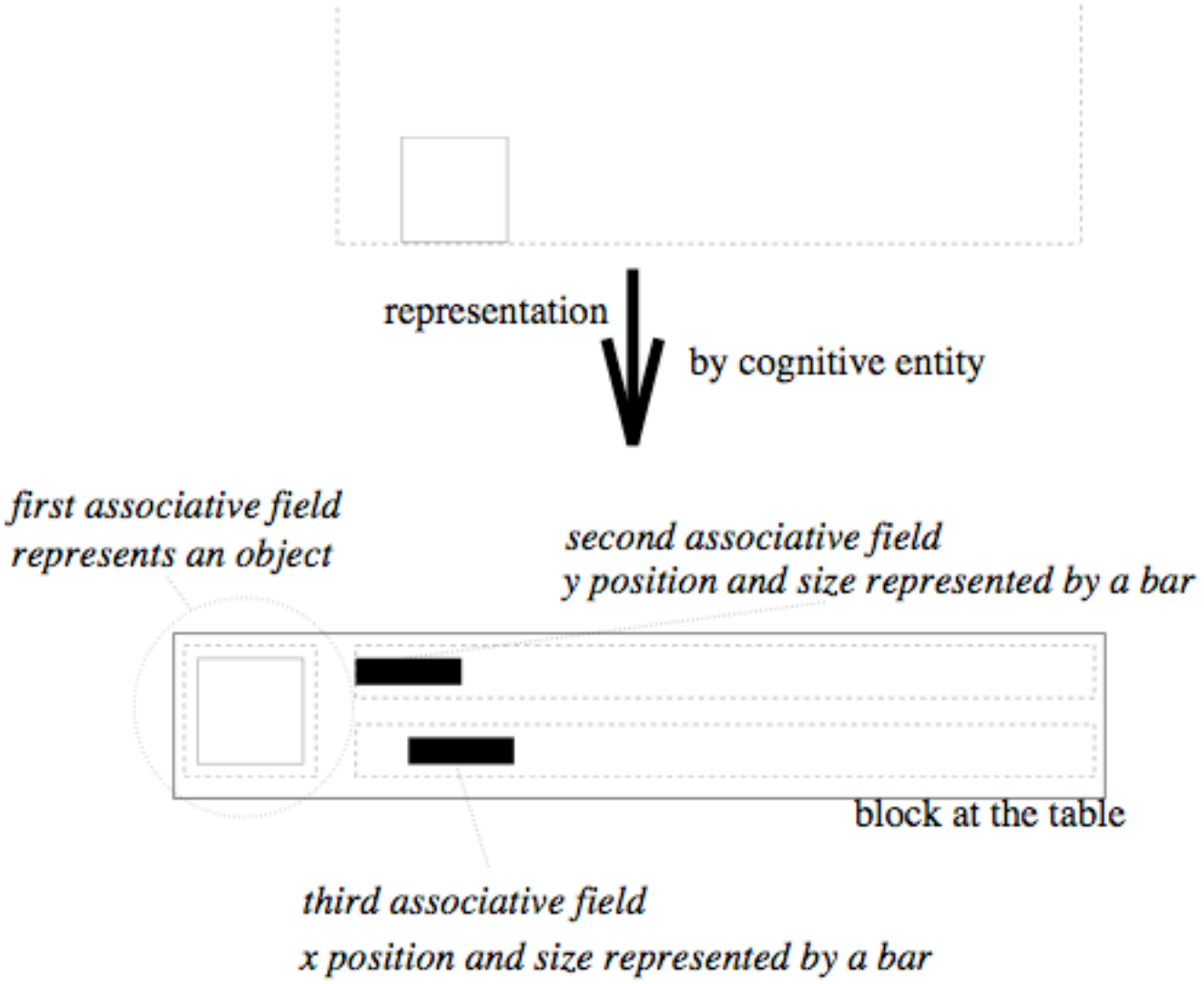
Similarity

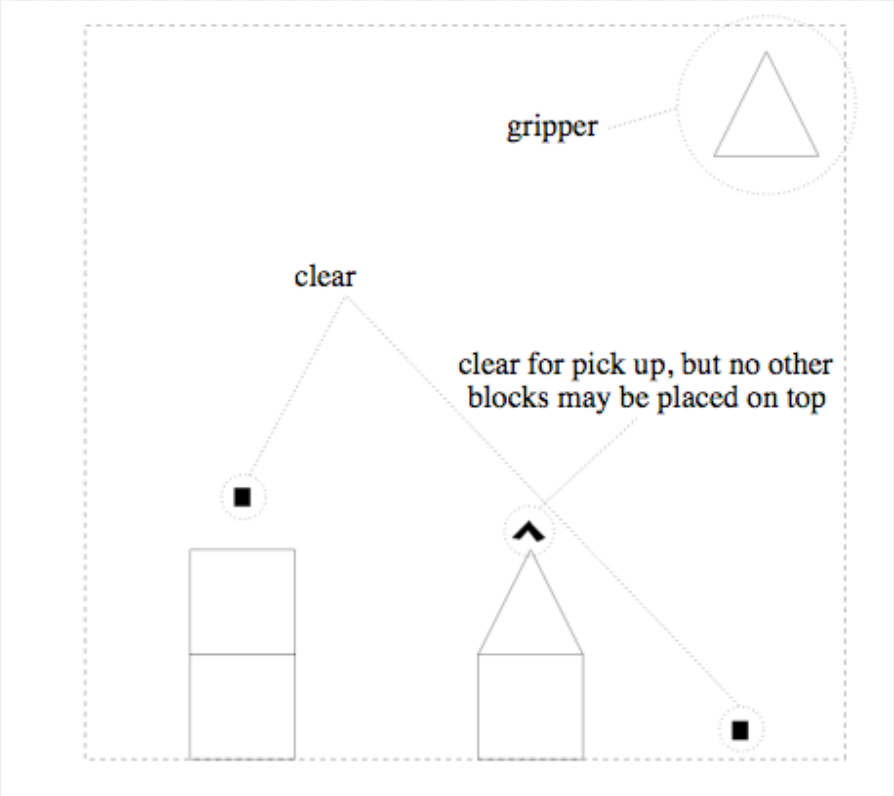


of tower to (a) 0.81

of tower to (b) 0.7

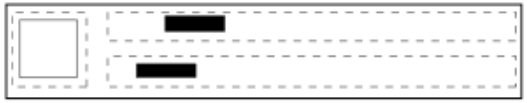
of tower to (c) 0.7


representation
by cognitive entity
first associative field
represents an object
second associative field
y position and size represented by a bar
block at the table
third associative field
x position and size represented by a bar




gripper holding a pyramid

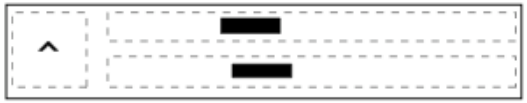

block at the table


block on other block

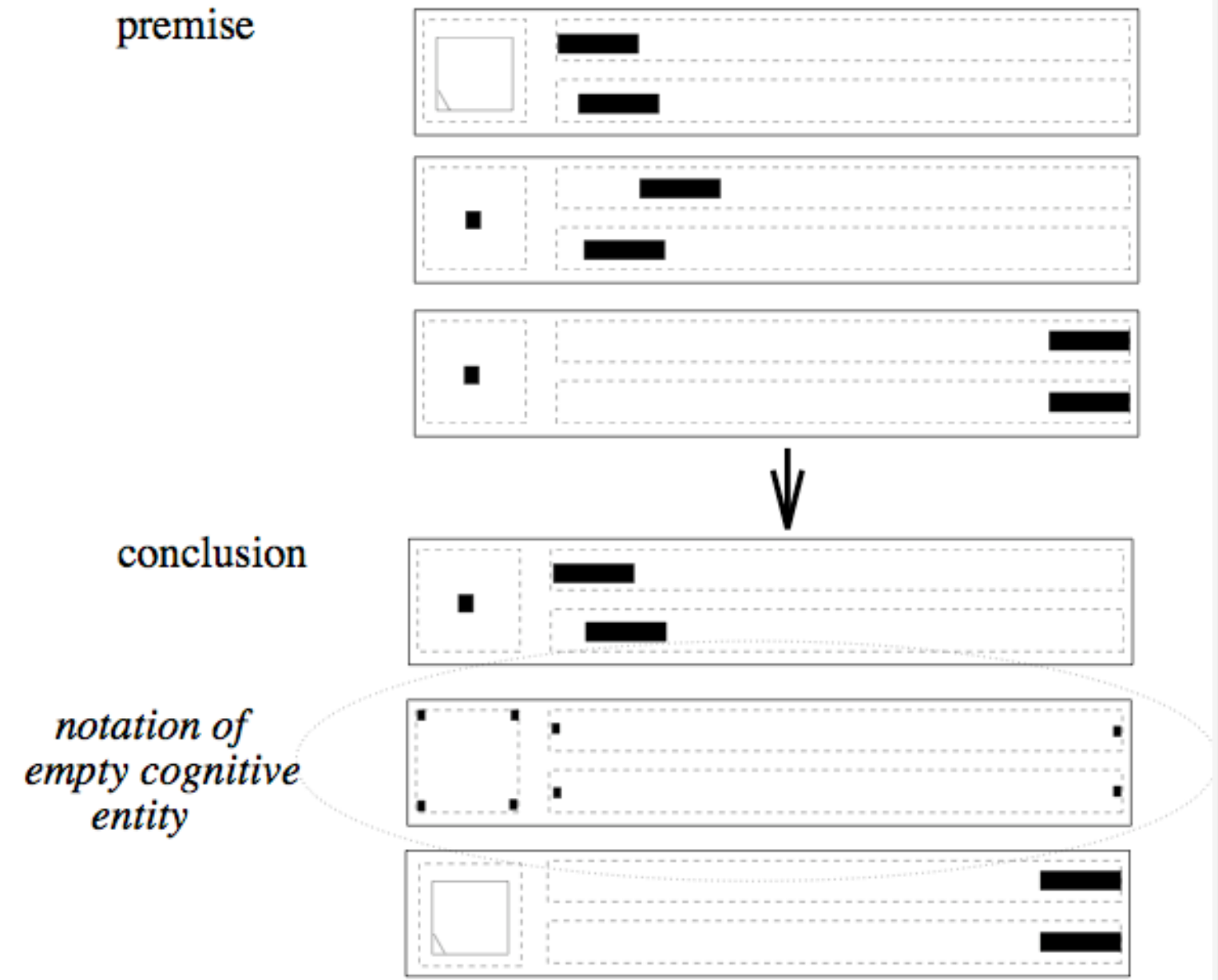

clear position

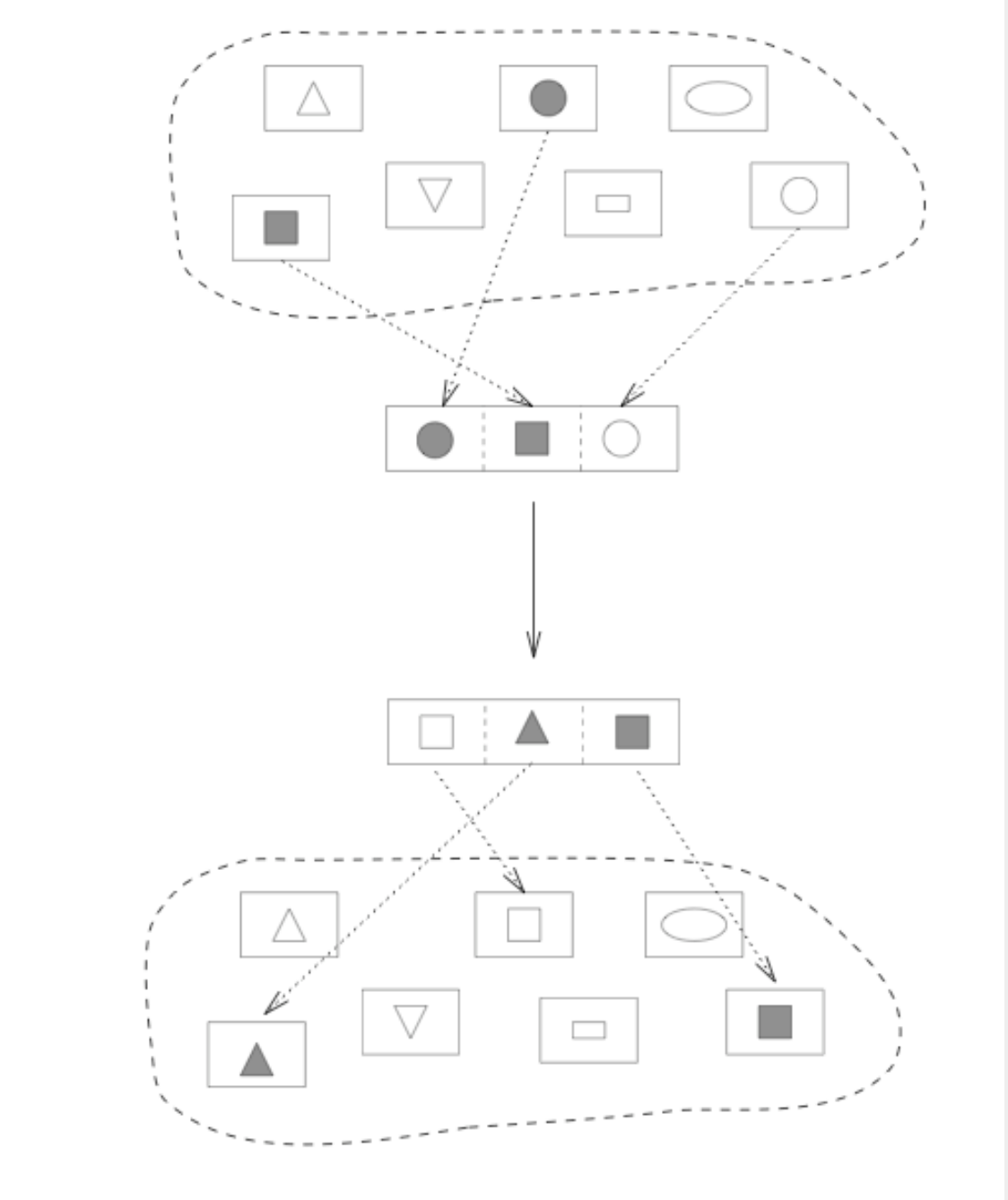

block at the table

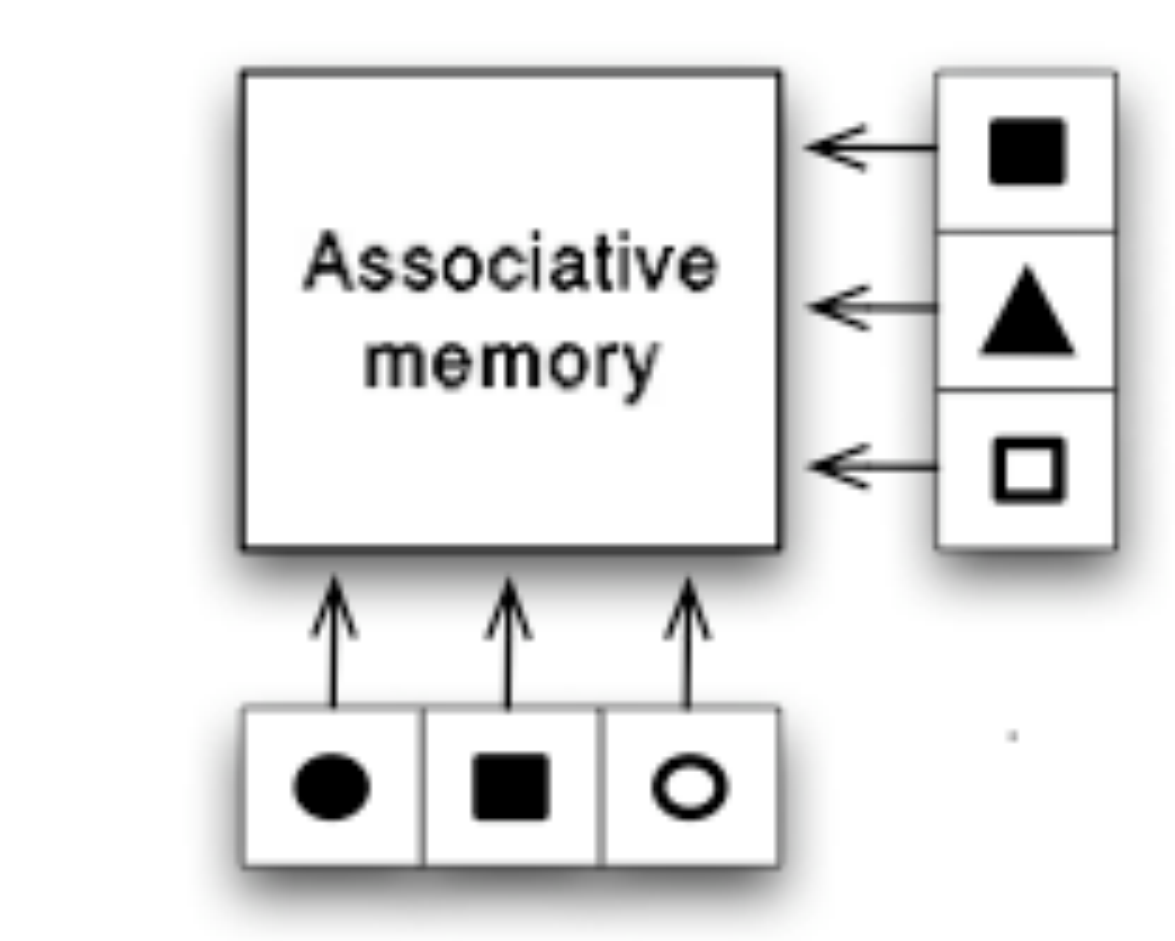

pyramid on other block

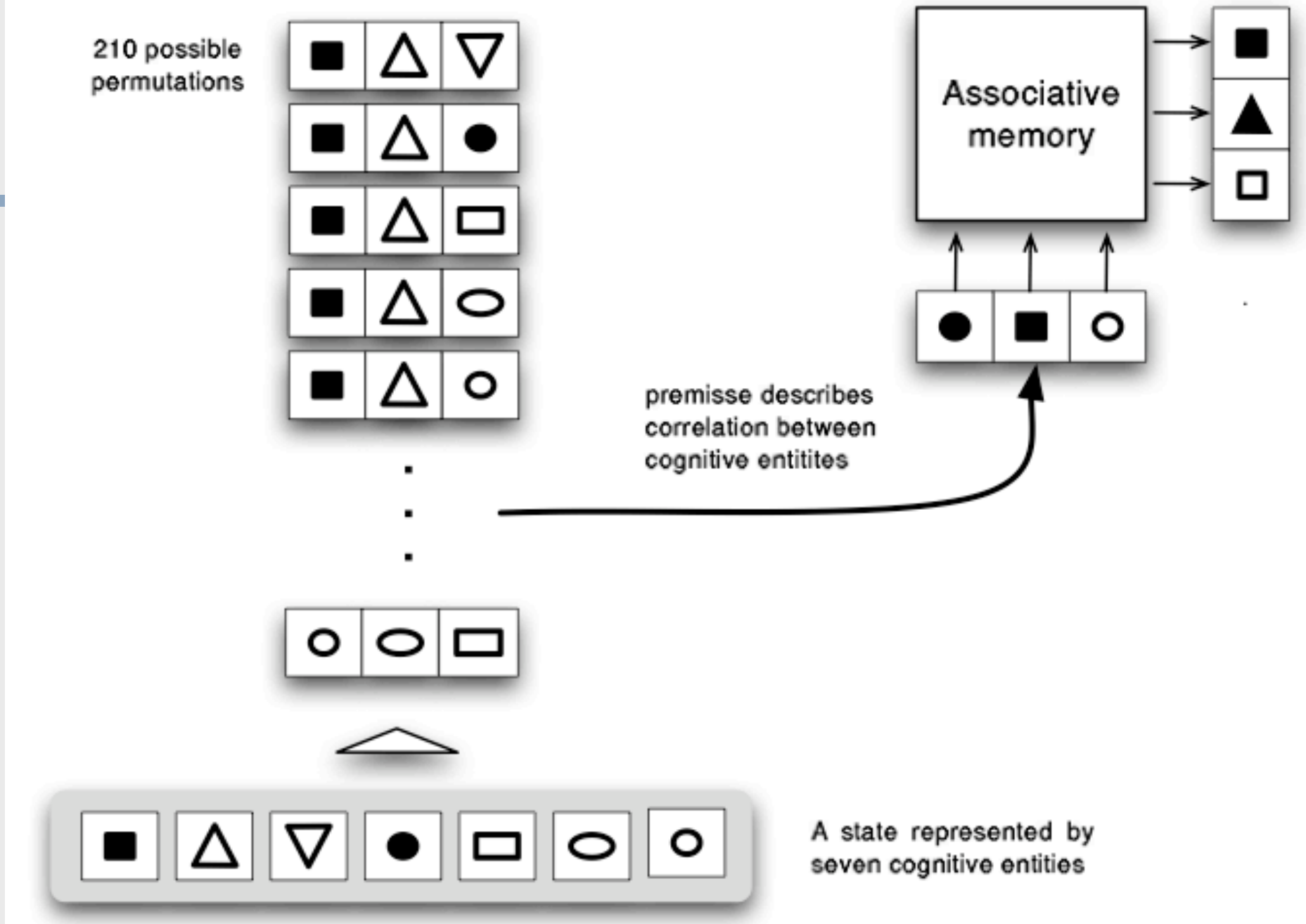

clear position

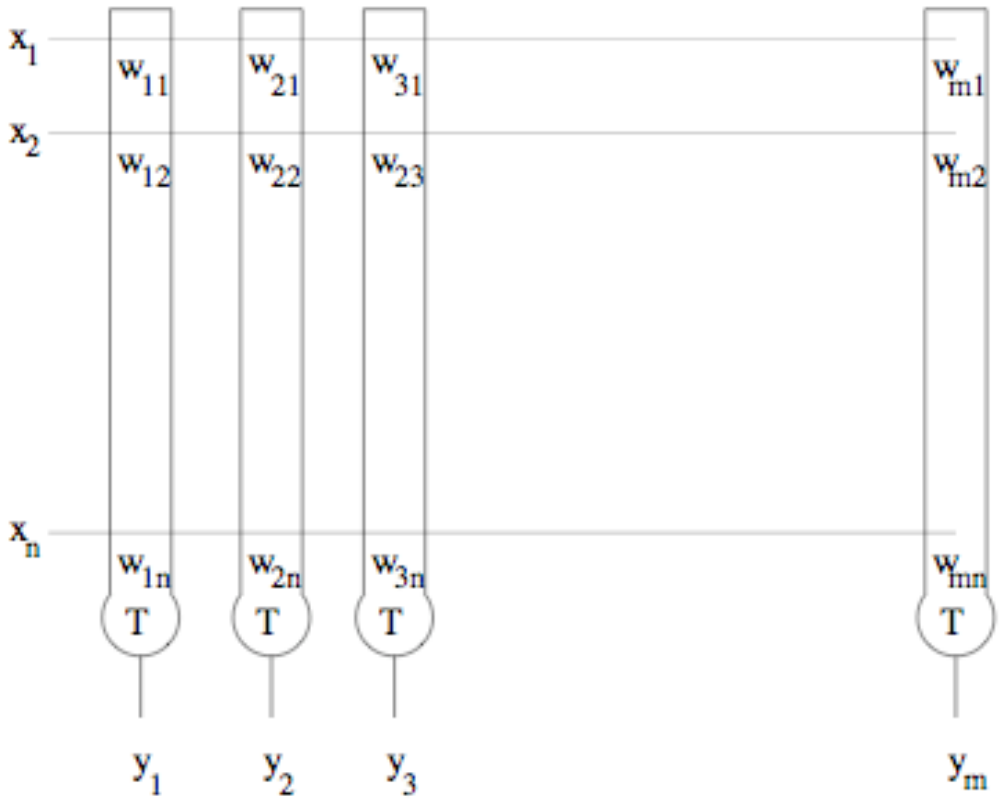

clear position

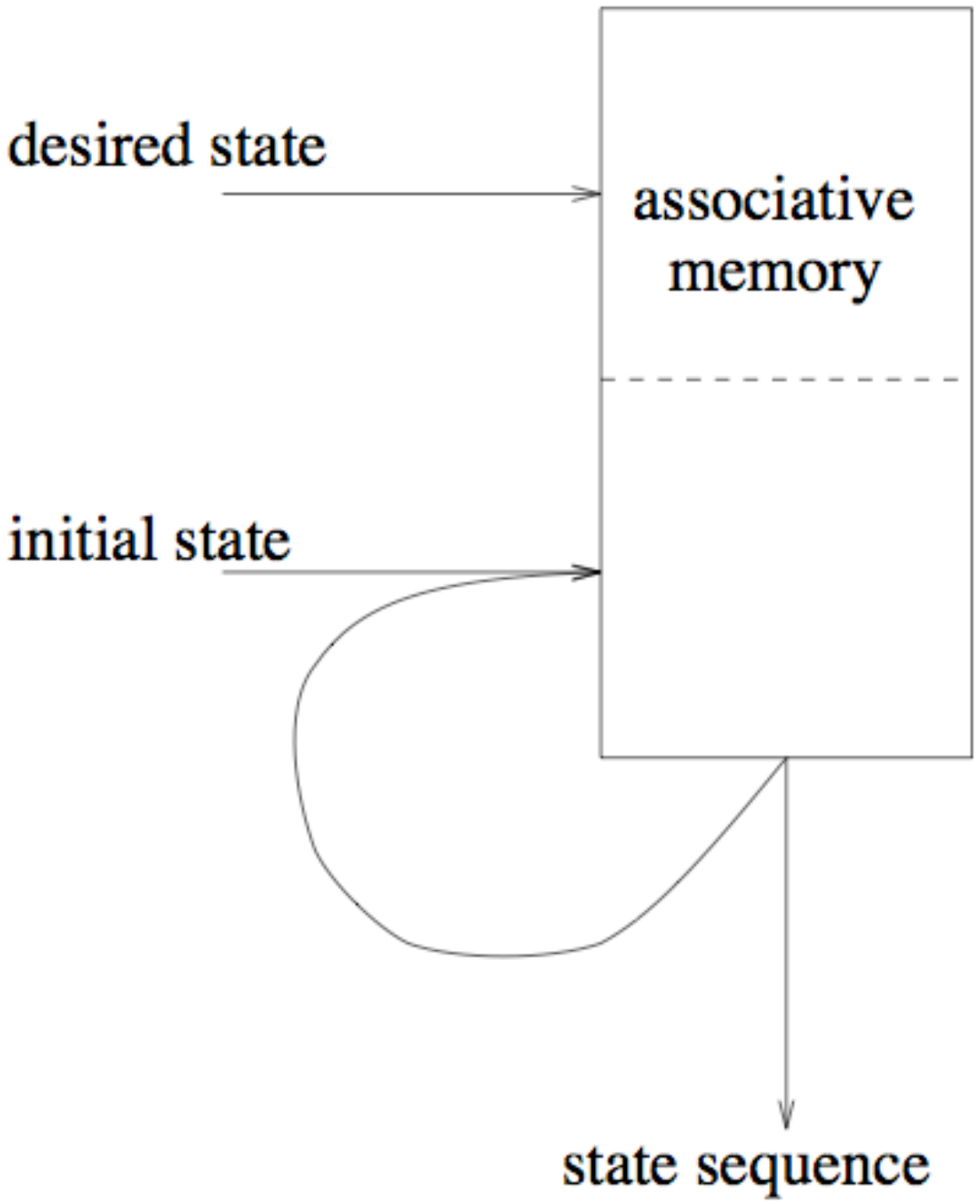

premise
conclusion
notation of empty cognitive entity

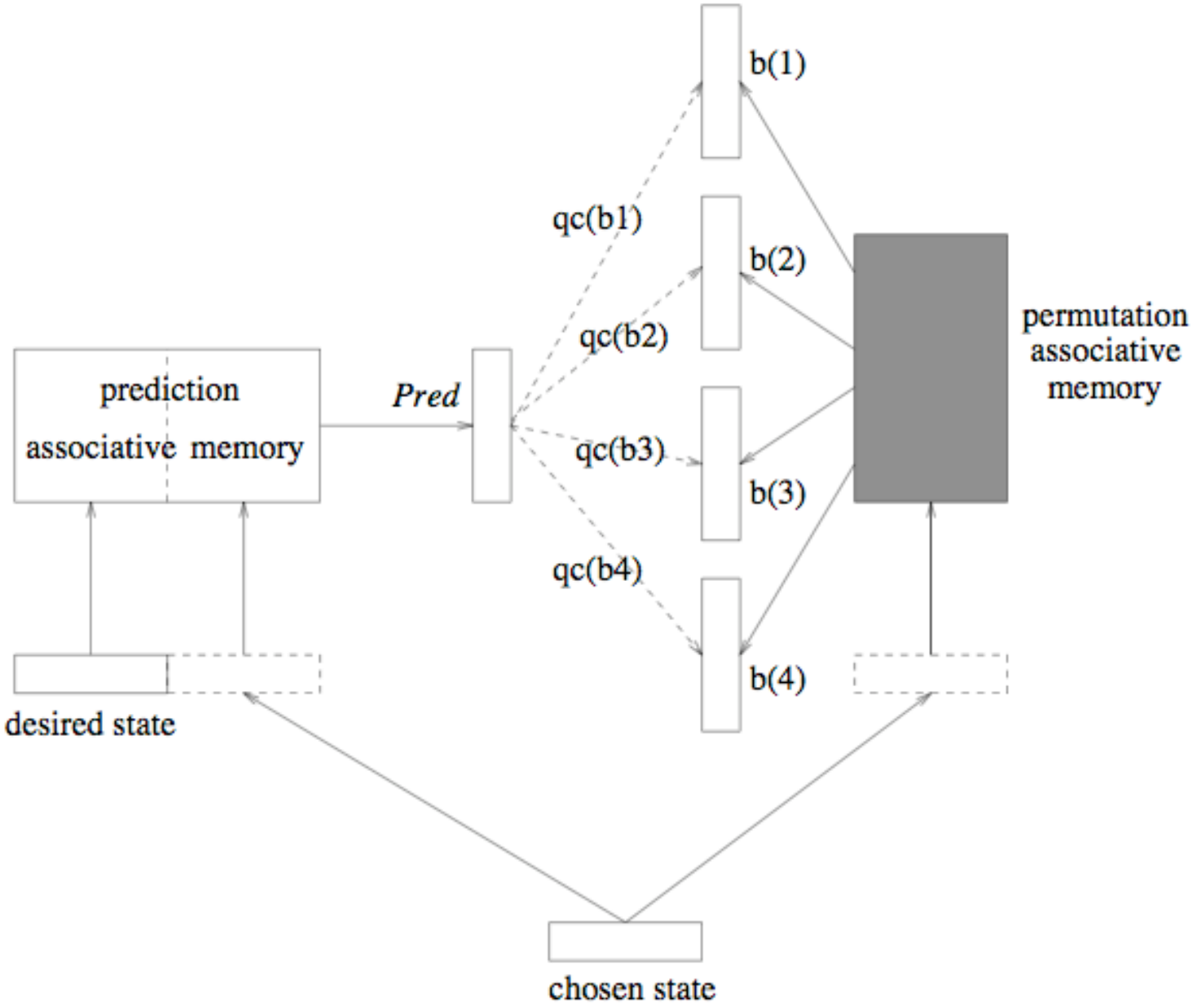


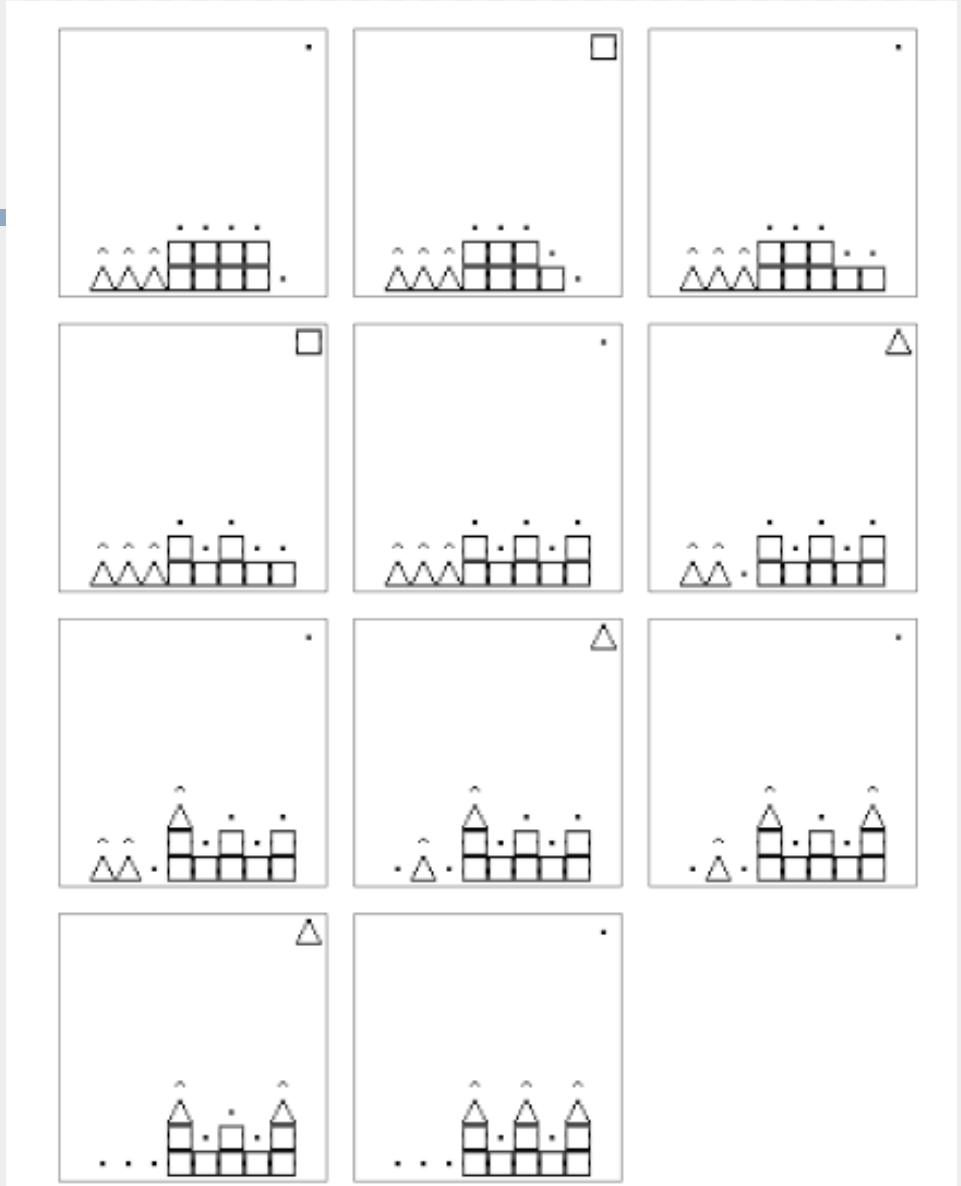

Associative
memory

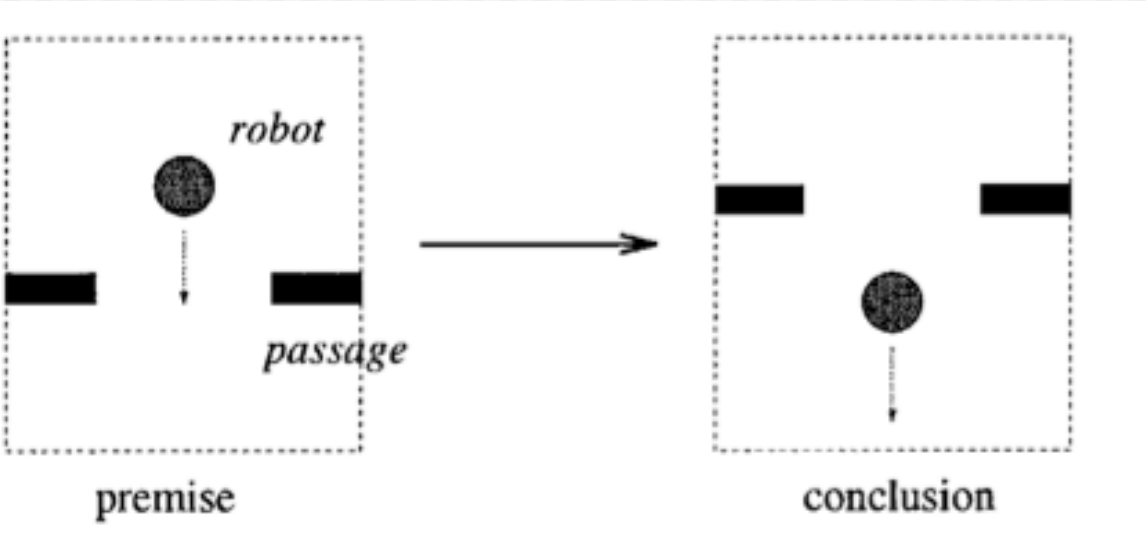

210 possible permutations
Associative memory
premisse describes correlation between cognitive entitites
A state represented by seven cognitive entities

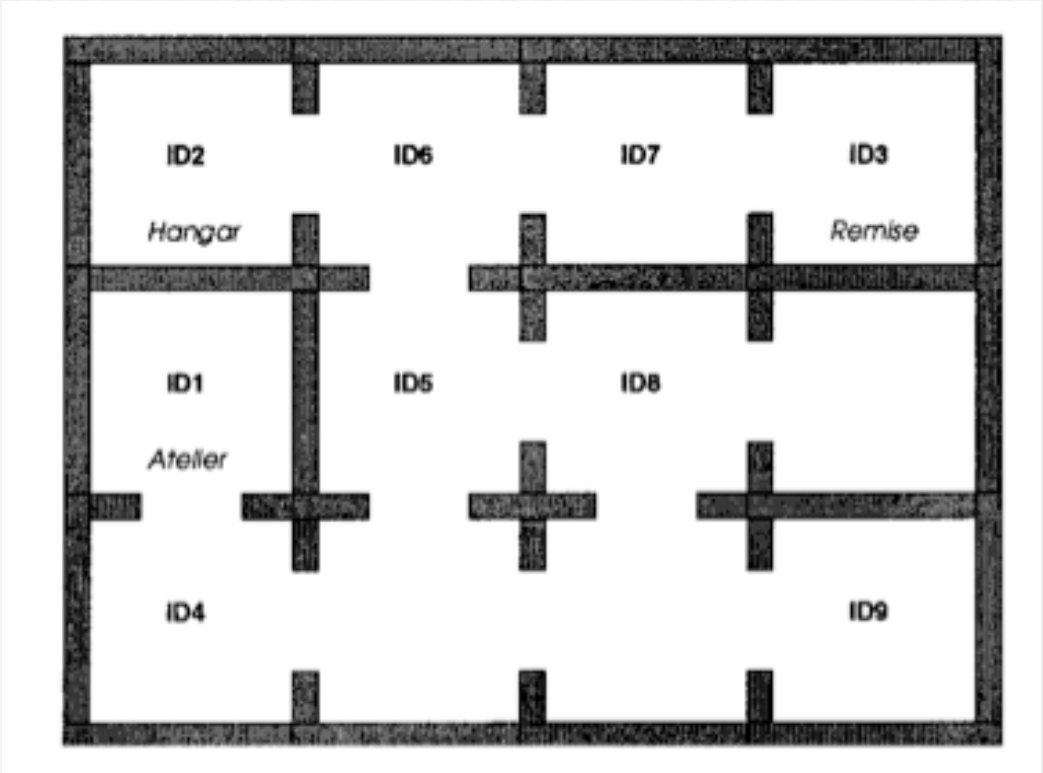

x1
x2
xn
w11
w21
w31
wm1
w12
w22
w23
wm2
w1n
w2n
w3n
wmn
T
T
T
T
y1
y2
y3
ym

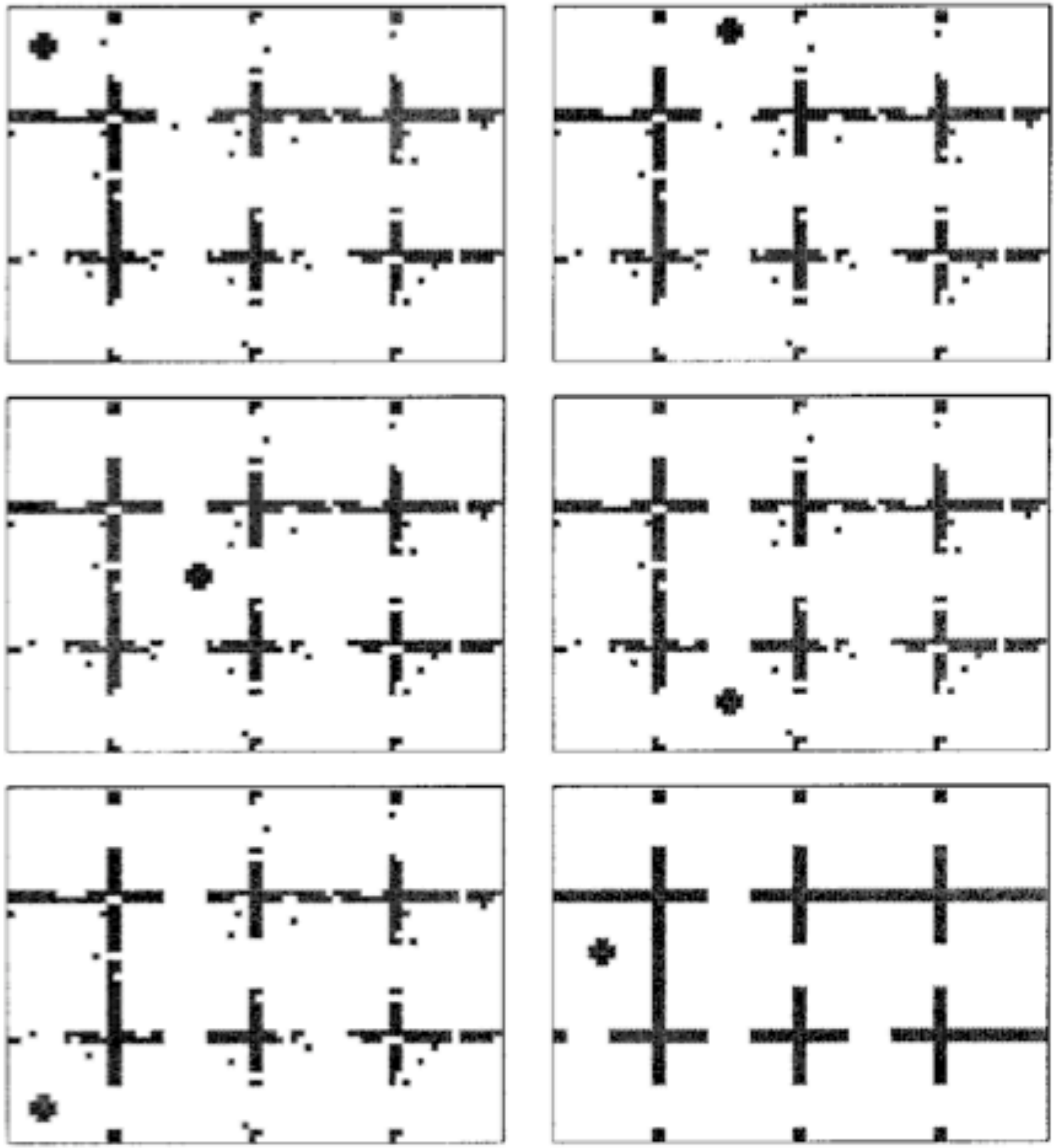

desired state
associative
memory
initial state
state sequence


b(1)
qc(b1)
b(2)
qc(b2)
permutation
associative
memory
prediction
associative memory
Pred
qc(b3)
b(3)
qc(b4)
b(4)
desired state
chosen state




robot
passage
premise
conclusion


ID2
ID6
ID7
ID3
Hangar
Remise
ID1
ID5
ID8
Atelier
ID4
ID9



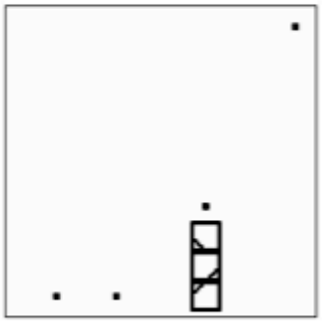


Fig. 11. The desired state, CBA tower.

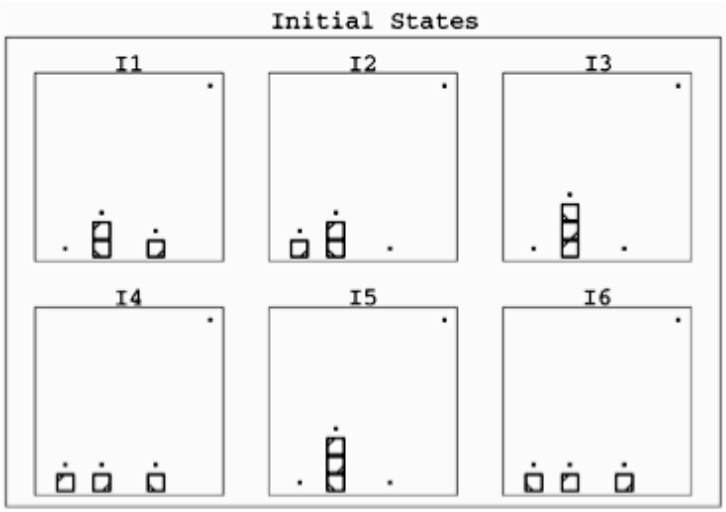
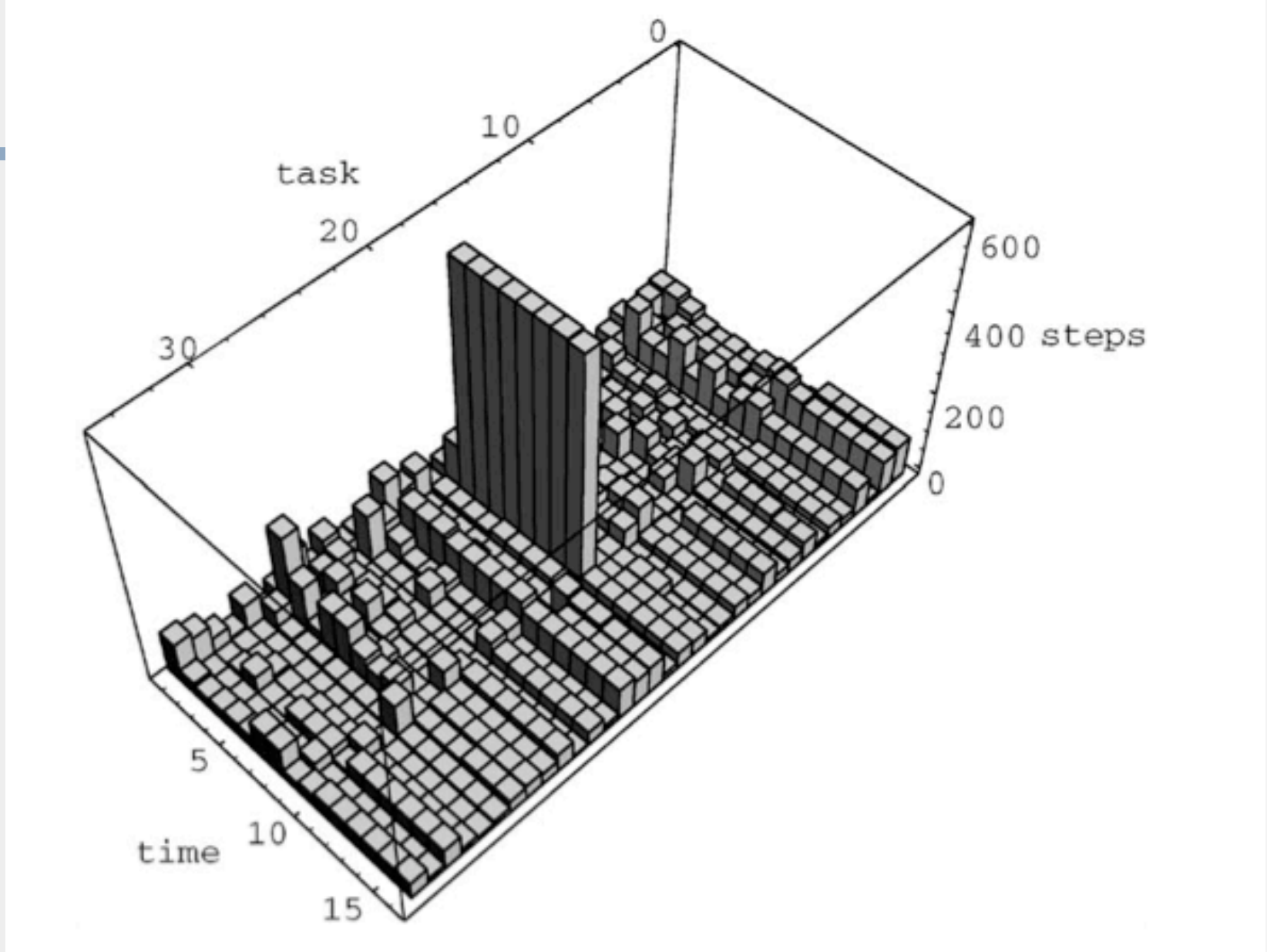
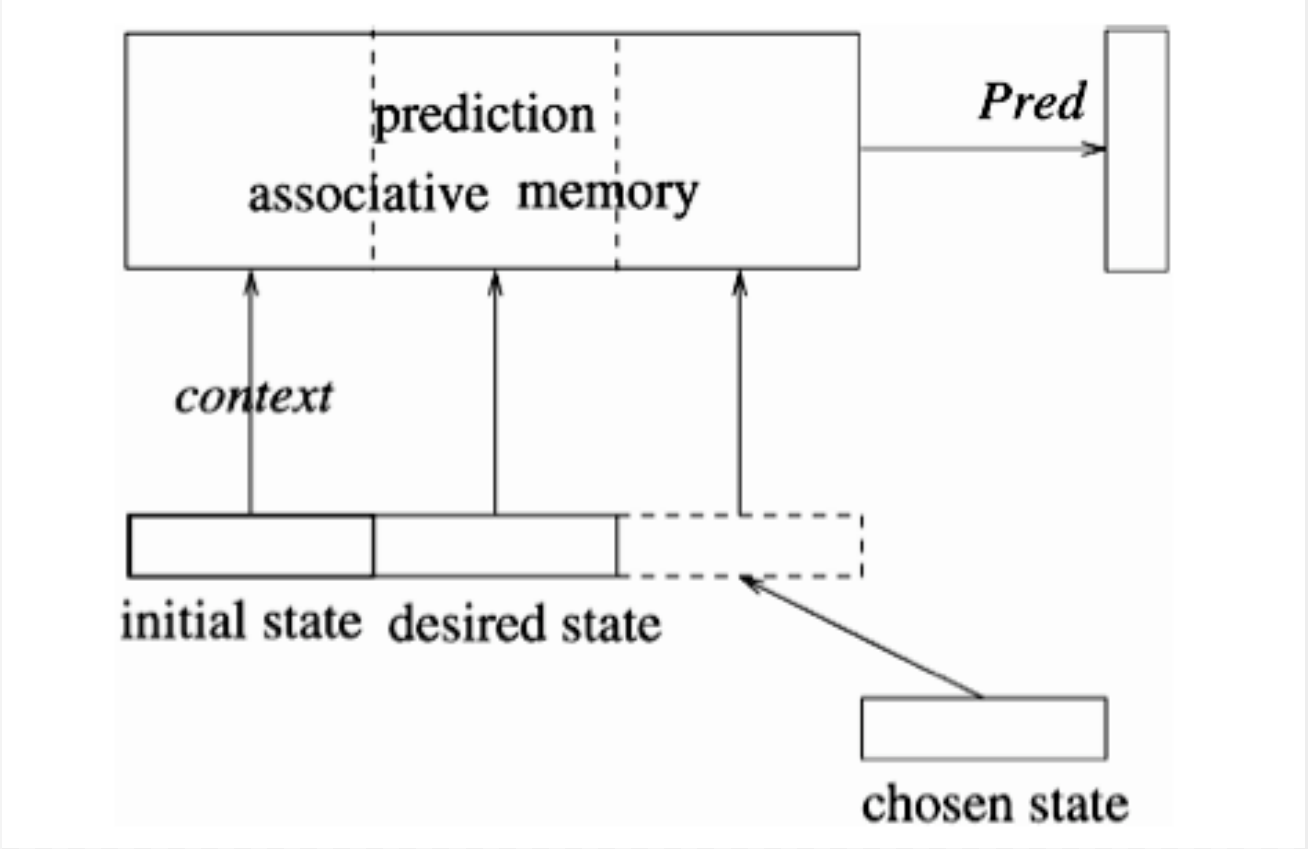
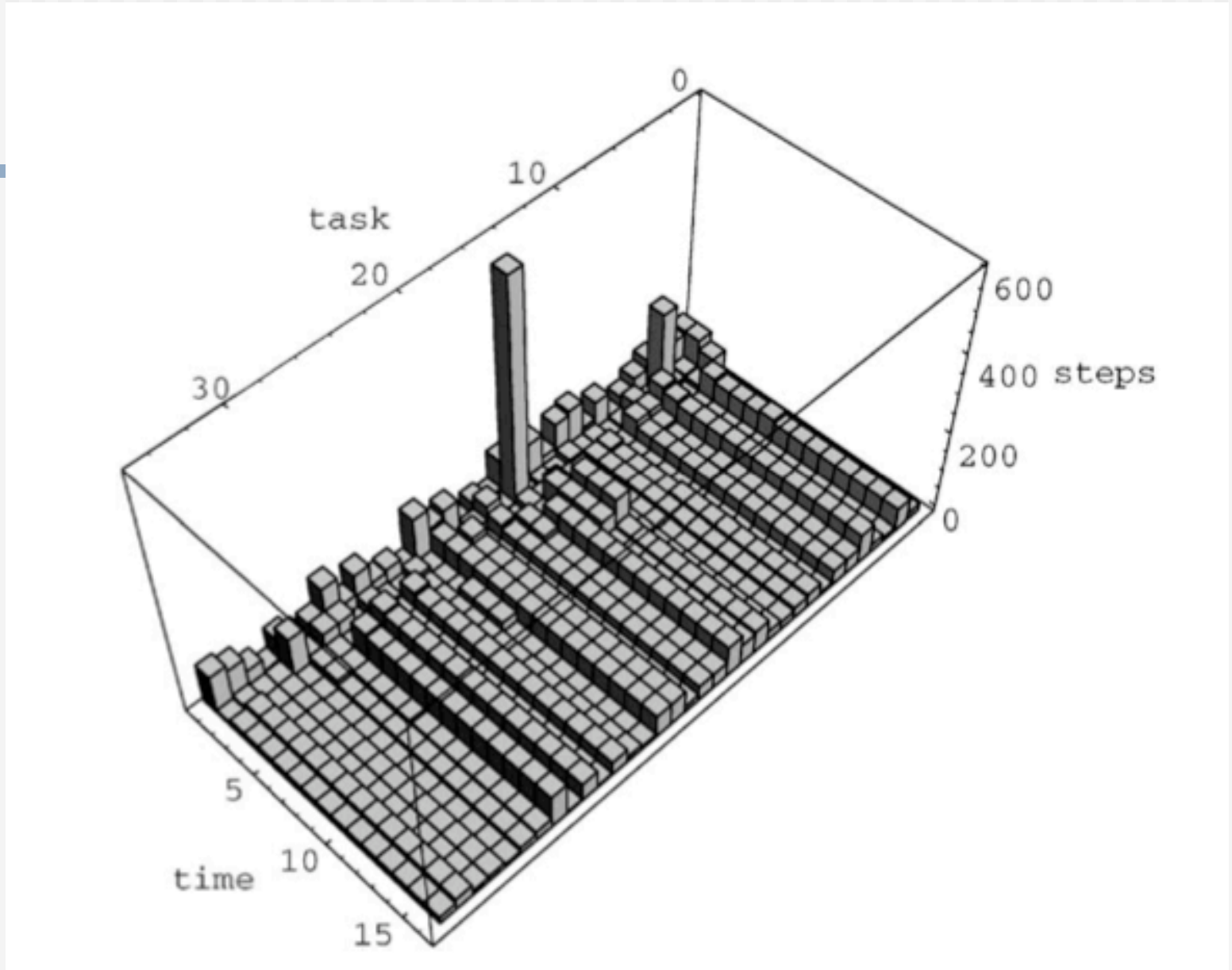


Fig. 12. The first six initial states of 34 different initial states.


0
10
task
20
30
600
400 steps
200
0
5
10
time
15


prediction
associative memory
Pred
context
initial state
desired state
chosen state


0
10
task
20
30
600
400 steps
200
0
5
10
time
15

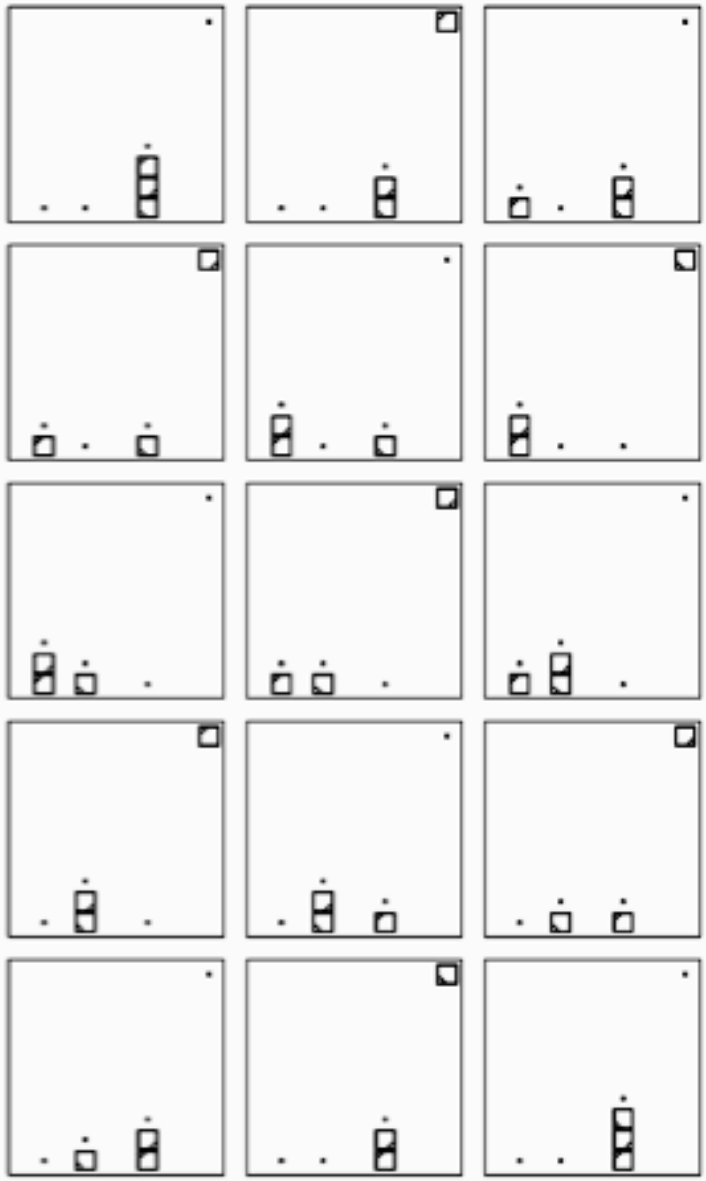


Fig. 18. Planning of the task unstack the ABC tower and stack the CBA tower. The best possible sequence of moves was found after perfect learning. Initial state ABC tower, desired state CBA tower. Planning sequence is shown line by line from left to right. Fourteen steps were needed.

- Learning Chunks (with experience)
- Learning with a teacher with associative memory
- Learning by experience with associative memory