USC Institute for Creative Technologies

University of Southern California

Tutorial

Σ The Sigma Cognitive Architecture/System

Paul S. Rosenbloom
[Abram Demski & Volkan Ustun also here]

8.1.2014

The work depicted here was sponsored by the U.S. Army, the Air Force Office of Scientific Research and the Office of Naval Research. Statements and opinions expressed do not necessarily reflect the position or the policy of the United States Government, and no official endorsement should be inferred.

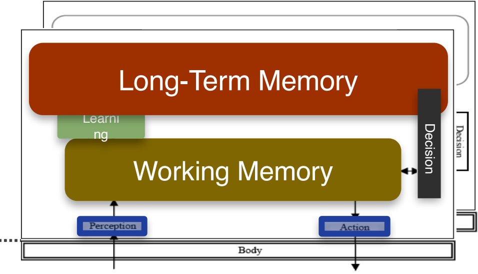
Goal of this Tutorial

- Not an introduction to programming in Sigma
  - Also not a hands on tutorial
- Goal is instead to provide a deeper insight into Sigma:
  - What it is about
  - How it works
  - What it is capable of

First public tutorial on Sigma

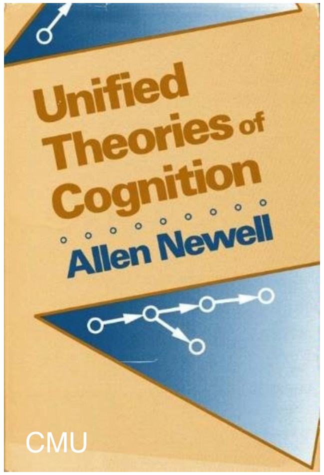
Feel free to ask questions at any time

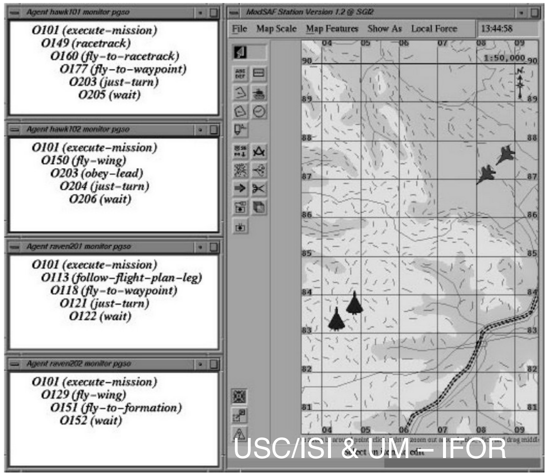
- Will mix lecture, live demonstration and Q&A
  - Sigma 34 on fairly slow machine (old MacBook Air)
  - Sigma 35 on more appropriate machine runs 2-3 times faster



Cognitive Architecture

- Fixed structure underlying *mind* (& thus intelligent behavior)
  - Defines mechanisms for memory, reasoning, learning, interaction, ...
  - Specifies how mechanisms interact
  - Supports acquisition and use of knowledge and skills





- Related to AGI architectures, intelligent agent & robot architectures, AI languages, whole brain models, …

Overall Desiderata for the Sigma (Σ) Architecture

- A new breed of cognitive architecture that is
  - *Grand unified*
    - Cognitive + key non-cognitive (perceptuomotor, affective, …)
  - *Functionally elegant*
    - Broadly capable yet simple and theoretically elegant
  - *Sufficiently efficient*
    - Fast enough for anticipated applications
- For virtual humans (& intelligent agents/robots) that are
  - Broadly, deeply and robustly *cognitive*
  - *Interactive* with their physical and social worlds
  - *Adaptive* given their interactions and experience
- For integrated models of natural minds

Hybrid: Discrete + Continuous
Mixed: Symbolic/Relational + Probabilistic/Statistical

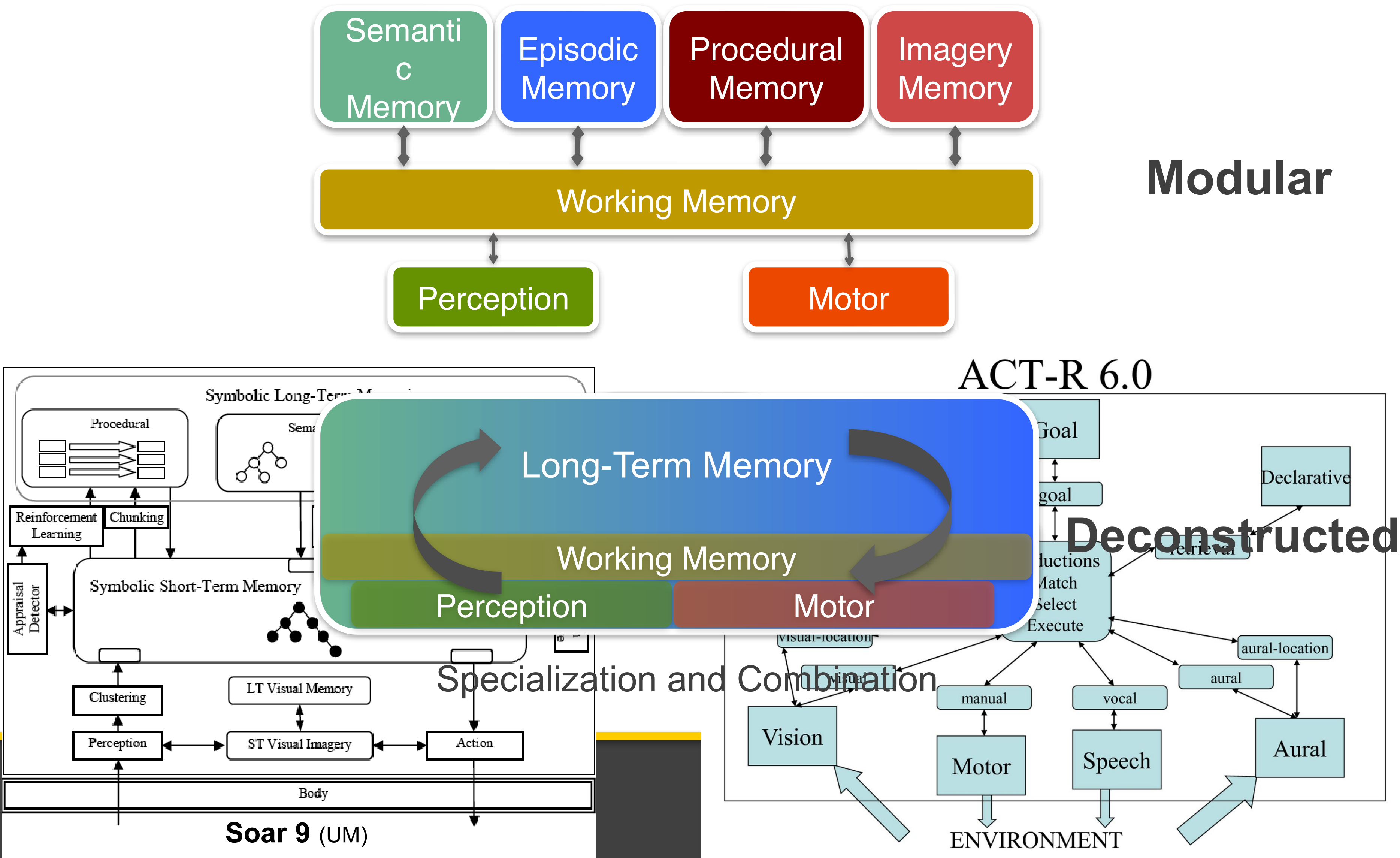
More on Functional Elegance

- Can the diversity of intelligent behavior arise from the interactions among a small general set of mechanisms?
  - *Cognitive Newton's laws*
  - *Elementary cognitive particles* ➔ *Periodic table of behaviors*
  - *Cognitive axioms* ➔ *"Proofs" of behavioral theorems*

Akin to *Universal AI* (Hutter) in spirit, but not necessarily as minimal

- Given a small set of general mechanisms how many requisite behaviors can be produced?
  - Discovering "proofs" of intelligent behaviors
  - *Deconstructing* intelligent behaviors in terms of cognitive mechanisms
- Towards deeper theories with greater explanatory reach
  - Discovering a sufficient small general set of cognitive mechanisms
  - Discovering how they can yield the breadth of intelligent behavior

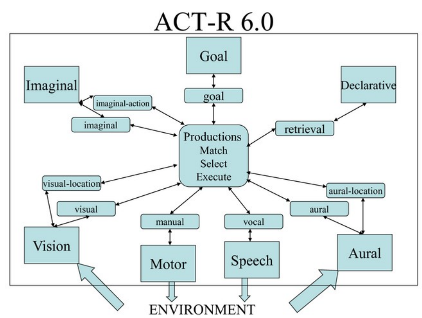
Modular versus Deconstructed (Functionally Elegant) Approaches to Cognitive Architecture

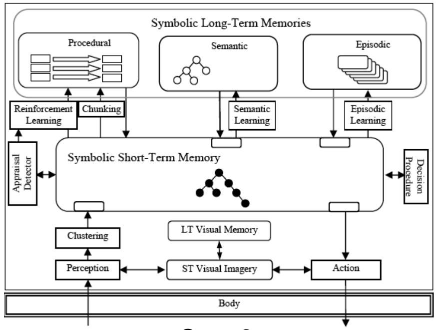


Graphical Architecture Hypothesis

Key to success is *blending what has been learned from over three decades of independent work* in **cognitive architectures** and **graphical models**

Cognitive Architectures

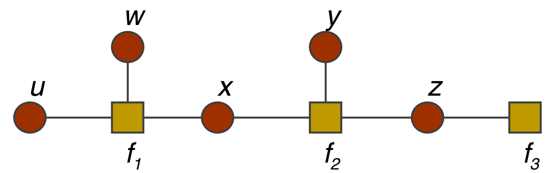




Soar 9

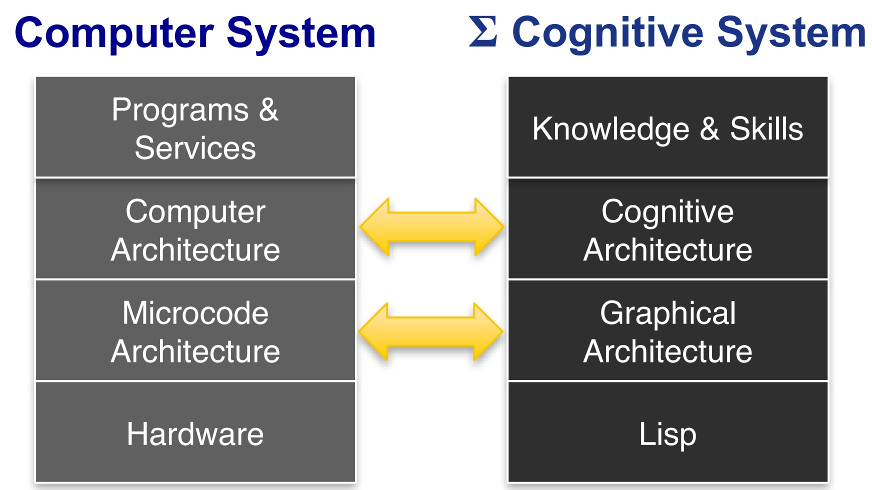
Graphical Models

$f(u,w,x,y,z) = f_1(u,w,x)f_2(x,y,z)f_3(z)$



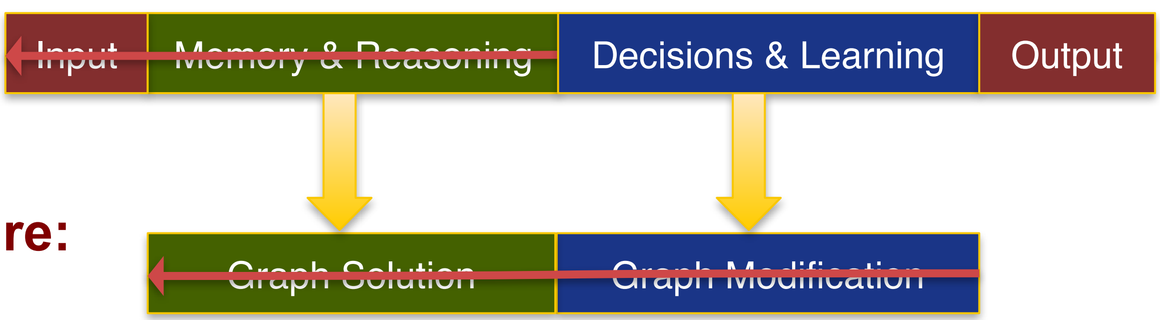
The Structure of Sigma

- Constructed in layers
  - In analogy to computer systems



Cognitive Architecture:

Predicates
Conditionals
Control structure



Graphical Architecture:

Graphical models
Piecewise-linear functions
Gradient-descent learning

Predicates

Conditionals

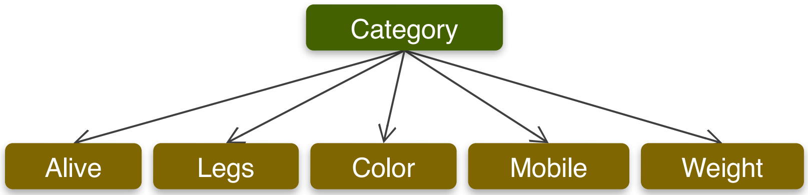
Control Structure

COGNITIVE ARCHITECTURE

Initial Example Tasks

- Transitive closure: Next(*a*, *b*) & Next(*b*, *c*) ➔ Next(*a*, *c*)
  - **Given** Next(i1, i2) and Next(i2, i3), yield Next(i1, i3)

- Naïve Bayes classifier
  - Given cues, retrieve/predict object category & missing attributes
    - E.g., **Given** *Alive*=T & *Legs*=4 **Retrieve** *Category*=Dog, *Color*=Brown, *Mobile*=T, *Weight*=67



When both, unique are “function of” universal

Predicates

- Specify relations among typed arguments
  - Defined via a *name, typed arguments* and other optional attributes
  - `(predicate 'concept :arguments '((id id) (value type %)))`
- Types may be *symbolic* or *numeric* (*discrete* or *continuous*)
  - `(new-type 'id :constants '(i1 i2 i3))`
  - `(new-type 'type :constants '(walker table dog human))`
  - `(new-type 'color :constants '(silver brown white))`
  - `(new-type 'i04 :numeric t :discrete t :min 0 :max 5)`
    - Discrete [0, 5) => 0, 1, 2, 3, 4
  - `(new-type 'weight :numeric t :min 0 :max 500)`
    - Continuous [0, 500) => [0, 500-ε]

Symbolic

Discrete Numeric

Continuous Numeric

- Predicates may be *open* or *closed* world
  - Whether unspecified values are assumed false (0) or unknown (1)
  - `(predicate 'concept2 :world 'closed :arguments '((id id) (value type !)))`
- Arguments may be *universal* or *unique* (*distribution* or *selection*)
  - `(predicate 'next :world 'closed :arguments '((id id) (value id)))`

Pure rules: Closed and universal
Pure probabilities: Open and unique

Predicate Memories

- Each predicate induces a segment of *working memory* (WM)
  - Closed-world predicates *latch* their results for later reuse while open-world predicates only maintain results while supported
    - *Selection predicates* latch a specific choice rather than whole distribution
      - Best, probability matching, Boltzmann, expected value, …
- Perception predicates induce a segment of the *perceptual buffer*
  - Input is latched in perceptual buffer until changed `:perception t`
- Predicates may also include an optional (piecewise linear) *function*
  - *Long-term memory* (LTM) for predicate

```
(predicate 'concept-color :arguments '((concept type) (color color %))
           :function '((.95 walker silver) (.05 walker brown)
                       (.05 table silver) (.95 table brown)
                       (.05 dog silver) (.7 dog brown) (.25 dog white)
                       (.5 human brown) (.5 human white)))
```

P(color | concept)

- With *episodic memory*, also get LTM for history of predicate's values

Conditionals

- Structure *long-term memory* (LTM) and *basic reasoning*
  - Deep blending of traditional rules and probabilistic networks
- Comprise a *name*, *predicate patterns* and an optional *function*
  - Patterns may include *constant tests* and *variables* (in parentheses)
    - `(tetromino (x (x)) (y 1) (present true))`
    - [Constant tests have been generalized to *piecewise-linear filters*]
  - Patterns may be *conditions*, *actions* or *condacts*
  - As with predicate functions, conditional functions are *piecewise linear*

```
(conditional 'trans
   :conditions ’((next (id (a)) (value (b)))
                 (next (id (b)) (value (c))))
   :actions '((next (id (a)) (value (c)))))
```

```
(conditional 'acceptable
   :conditions '((state (state (s)))
                 (operator (id (o)) (state (s))))
   :actions '((selected (state (s)) (operator (o))))
              :function .1)
```

```
(conditional 'concept-color*join
   :conditions '((object (state (state)) (id (id))))
   :condacts '((concept (id (id)) (value (concept)))
               (color (id (id)) (value (color)))
               (concept-color (concept (concept)) (color (color))))
```

Conditionals (Rules)

- *Conditions* and *actions* embody traditional rule semantics
  - Conditions: Access information in WM
  - Actions: Suggest changes to WM
- Multiple actions for the same predicate must *combine* in WM
  - Traditional parallel rule system uses *disjunction* (*or*): A v B
  - Sigma uses multiple approaches depending on nature of predicate
    - For a universal predicate, uses *maximum*: Max(A, B)
    - For a normalized distribution, uses *probabilistic or*: P(A v B)
      - $= P(A) + P(B) - P(AB) \approx P(A) + P(B) - P(A)P(B)$
        - Assumes independence since doesn't have access to P(AB)
    - For an unnormalized distribution, uses *sum*: P(A) + P(B)

```
(conditional 'acceptable
   :conditions '((state (state (s)))
                 (operator (id (o)) (state (s))))
   :actions '((selected (state (s)) (operator (o))))
   :function .1)
```

```
(conditional 'trans
   :conditions '((next (id (a)) (value (b)))
                 (next (id (b)) (value (c))))
   :actions '((next (id (a)) (value (c)))))
```

Conditionals (Probabilistic Networks)

- *Condacts* embody (bidirectional) constraint/probability semantics
  - Access WM and suggest changes to it (combining multiplicatively)
- *Functions* relate/constrain/weight combinations of values of specified variables (or are constant if no variables specified)
- Functions traditionally part of conditionals in Sigma, but now preferably specified as part of predicates, unless constant
  - Was effectively specifying a pseudo-predicate in conditionals

```
(conditional 'concept-color
    :conditions '((object (state (state)) (id (id))))
    :condacts '((concept (id (id)) (value (concept)))
                (color (id (id)) (value (color))))
    :function-variable-names '(concept color)
    :function '((.95 walker silver) (.05 walker brown)
                (.05 table silver) (.95 table brown)
                (.05 dog silver) (.7 dog brown) (.25 dog white)
                (.5 human brown) (.5 human white)))
```

```
(predicate 'concept-color :arguments '((concept type) (color color %))
    :function '((.95 walker silver) (.05 walker brown)
                (.05 table silver) (.95 table brown)
                (.05 dog silver) (.7 dog brown) (.25 dog white)
                (.5 human brown) (.5 human white)))

(conditional 'concept-color*join
    :conditions '((object (state (state)) (id (id))))
    :condacts '((concept (id (id)) (value (concept)))
                (color (id (id)) (value (color)))
                (concept-color (concept (concept)) (color (color))))
```

Pattern types and functions can be mixed arbitrarily in conditionals

Piecewise Linear Functions

- Unified representation for *continuous*, *discrete* and *symbolic* data
- At base have multidimensional continuous functions
  - One dimension per variable, with multiple dimensions providing *relations*
  - Approximated as *piecewise linear* over *arrays/tensors of regions*
- *Discretize domain* for discrete distributions (& symbols)
- *Booleanize range* (and add symbol table) for symbols

  Color(O_1, Brown) & Alive(O_1, T)
- Dimensions/variables are *typed*

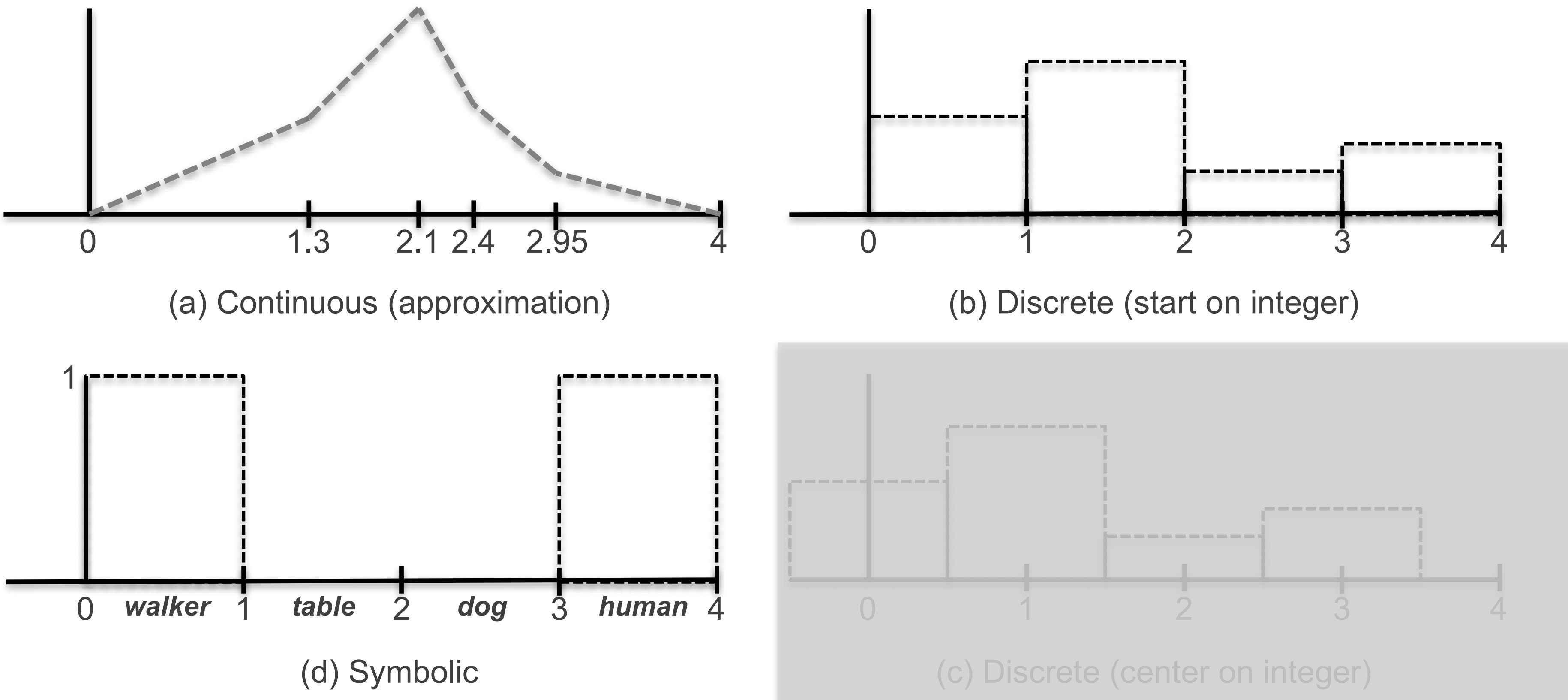
O_1	Brown	Silver	White
T	1	0	
F	0		

P(*legs* | *concept*)	Walker	Table	…
1	0	0	…
2	0	0	…
3	0	.1	…
4	1	.9	…

P(*weight* | *concept*)	Walker	Table	…
[1,10>	.01*w*	.001*w*	…
[10,20>	.2-.01*w*	"	…
[20,50>	0	.025-.00025*w*	…
[50,100>	"	"	…

Analogous to implementing digital circuits by restricting an inherently continuous underlying substrate

Piecewise Continuous) Functions



(a) Continuous (approximation)

(b) Discrete (start on integer)

(d) Symbolic

(c) Discrete (center on integer)

Unique variables: Distribution over which element of domain is valid (like random variables)

Universal variables: Any or all elements of the domain can be valid (like rule variables)

The Eight Puzzle

- Classic sliding tile puzzle
- Represented symbolically in standard AI systems
  - **LeftOf**(*cell*$_{11}$, *cell*$_{21}$), **At**(*tile*$_1$, *cell*$_{11}$), etc.
- Typically solved via some weak search method

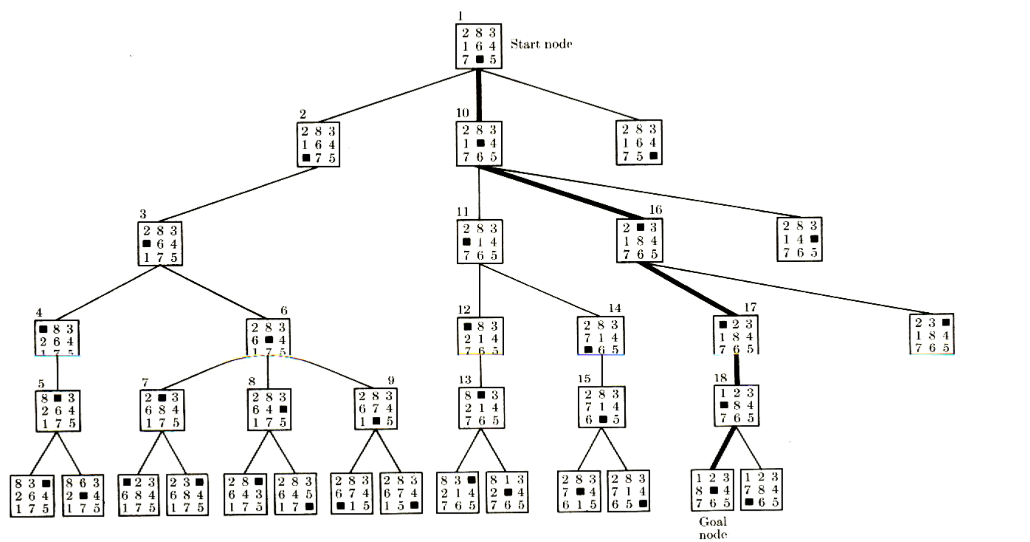
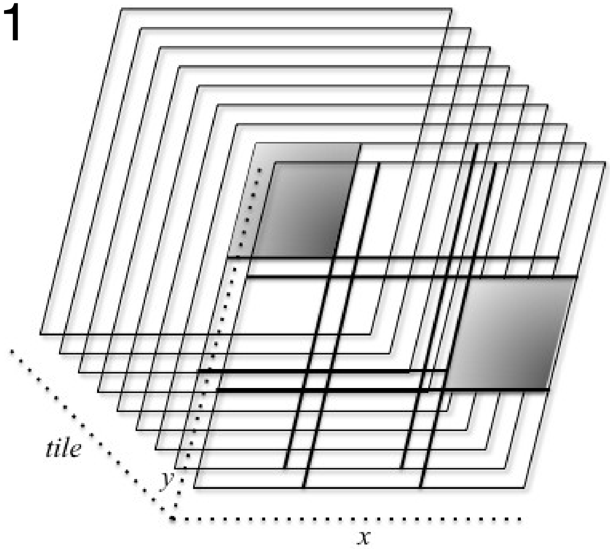
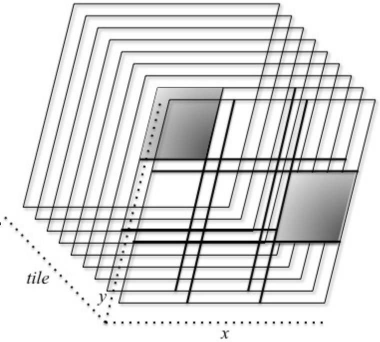


FIG. 3-5 *The tree produced by a depth-first search.*

Hybrid Representation of Eight Puzzle Board

- Instead represent as a 3D function
  - Continuous spatial *x* & *y* dimensions
    - `dimension[0-3)`
  - Discrete *tile* dimension (an *xy* plane)
    - `tile[0:9)`
  - Region of plane with tile has value 1
    - All other regions have value 0

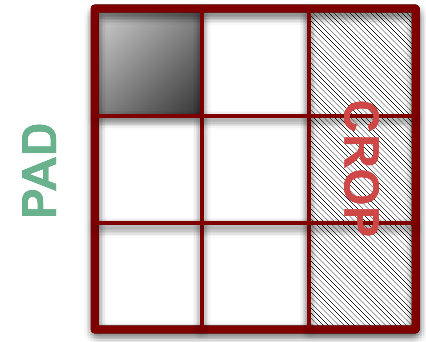




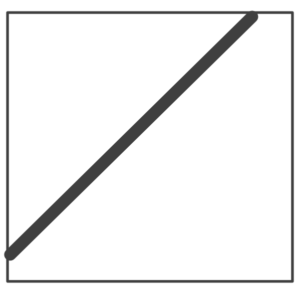
How to Slide a Tile

- Offset boundaries of regions along a dimension

```
CONDITIONAL Move-Right
   Conditions: (selected state:s operator:o)
               (operator id:o state:s x:x y:y)
               (board state:s x:x y:y tile:t)
               (board state:s x:x+1 y:y tile:0)
   Actions: (board state:s x:x+1 y:y tile:t)
            (board state:s x:x y:y tile:0)
```



- Special purpose optimization of a *delta function*



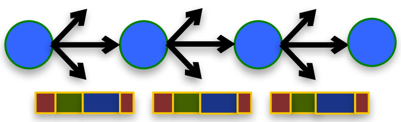
2	0	1	0
1	1	0	0
0	0	0	0
y/x	*0*	*1*	*2*

Control Structure: (Soar-like) Nesting of Layers

- **A *reactive* layer**
  - One (internally parallel) graph/cognitive cycle

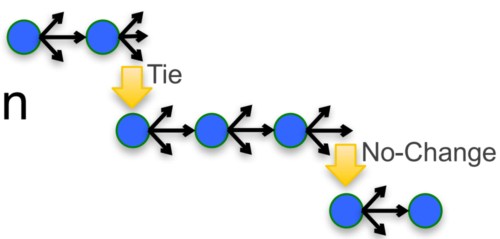
Which acts as the inner loop for

- **A *deliberative* layer**
  - Serial selection and application of operators



Which acts as the inner loop for

- **A *reflective* layer**
  - Recursive, impasse-driven, meta-level generation



- The layers differ in
  - Time scales
  - Serial versus parallel
  - Controlled (System 2) versus automatic (System 1)

Reactive Layer

One Decision/Cognitive Cycle

Input	Memory & Reasoning	Decisions & Learning	Output

- Perceive into perceptual buffer (for perception predicates)
  - Ideally/ultimately just raw signal
- Process conditionals to update distributions in WM
  - Accomplishes both long-term memory access and basic reasoning
    - For both cognitive and sub-cognitive (e.g., perceptual) processing
  - Doesn't make decisions or learn
- Decide by choosing one set of values for the selection arguments in each selection predicate

```
(predicate 'concept2 :world 'closed :arguments '((id id) (value type !)))
```

- Latch WM distributions and selections (for closed-world predicates)
- Learn for predicate and conditional functions (when enabled)
- Execute output commands

Deliberative Layer

The Problem Space Computational Model

Blocks World
Problem Space

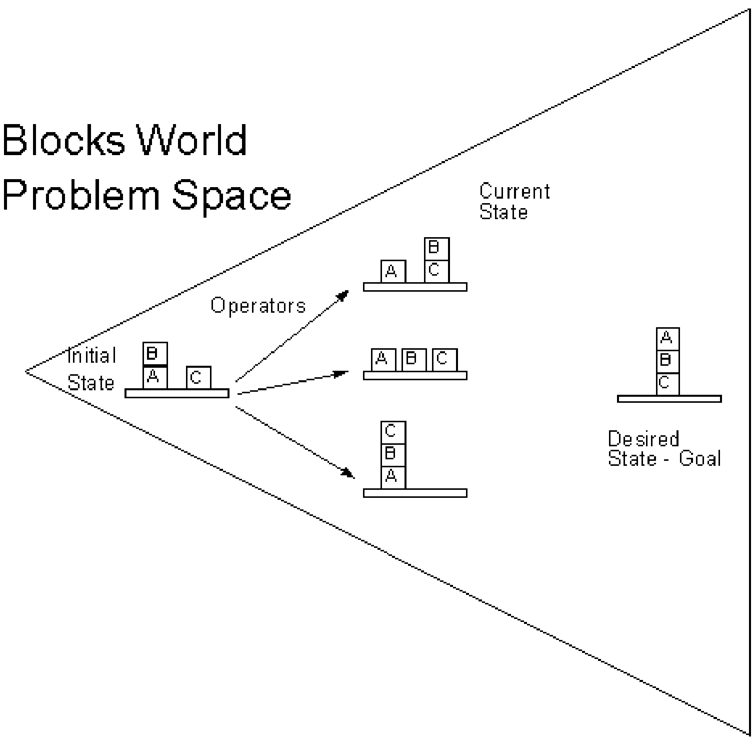


Figure 3.1: A problem space

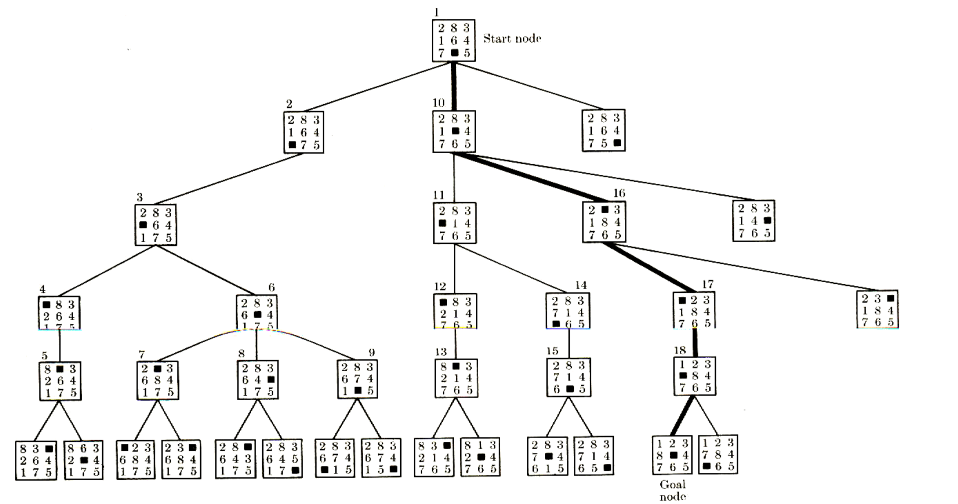


FIG. 3-5 *The tree produced by a depth-first search.*

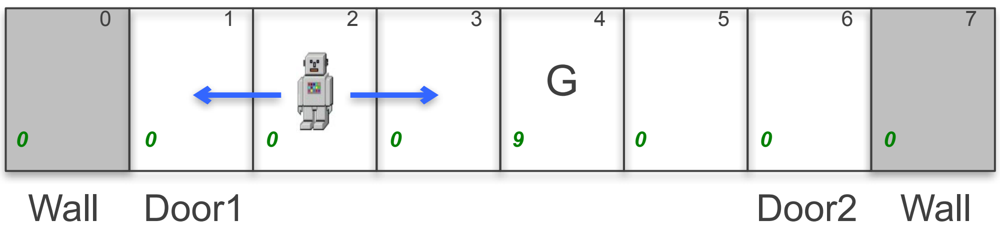
Follows path determined by knowledge

- Knowledge-intensive or algorithmic behavior
- Best, probability matching, Boltzmann, …

Doesn't actually do combinatoric search

- *Requires reflection*

New Task: Simulated Robot in 1D Corridor



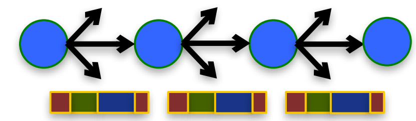
- Determine location in corridor
- Map corridor
- Learn to go to goal location in corridor
- Learn to model action effects

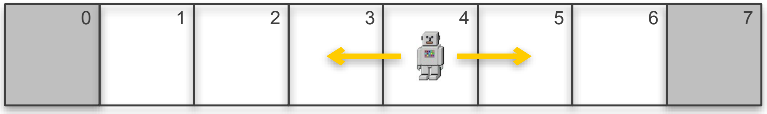
SLAM

RL

Deliberative Layer

The Problem Space Computational Model





- States
  - Closed-world predicates with `state` argument

```
(predicate 'location :world 'closed
  :arguments '((state state) (x location !)))
(predicate 'board :world 'closed
 :arguments '((state state) (x dimension) (y dimension) (tile tile !)))
```

state predicates

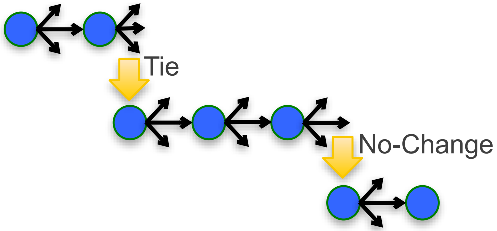
- Operators
  - A *type* for (internal) actions
  - Specified via `init-operators` or `init`
- Operators selected for states via `selected` predicate

```
(predicate ’selected :world ’closed :select ’best
  :arguments '((state state) (operator tile !)))
```

- Operators apply to states – via conditionals – to yield new states
  - Assumed done, and removed, on change to a unique state predicate

Reflective Layer

Impasses and Subgoaling (Meta-Levels)



- Impasses occur for problems in operator selection
  - *None*: No operator acceptable (i.e., none with a non-zero rating)
  - *Tie*: More than one operator has the same best rating
    - And the rating is not 1 (*best*)
  - *No-change*: An operator remains selected for >1 decision
- Impasses yield subgoals (meta-levels, reflective-levels, …)
  - Confusingly, these levels are called states (modeled after Soar)
    - The state argument in predicates is thus actually for levels
    - There are no unique symbols designating distinct states at a level
- Subgoal flushed when impasse goes away
  - Or when a change occurs higher in hierarchy

Typical Processing

- *Tie* impasses for selecting operators
- *No-change* impasses for implementing complex (multi-step) operators
- Can combine for *search*
  - 0. Tie among task operators
  - 1. No-change on evaluation operators
  - 2. Simulate operator to see how good it is

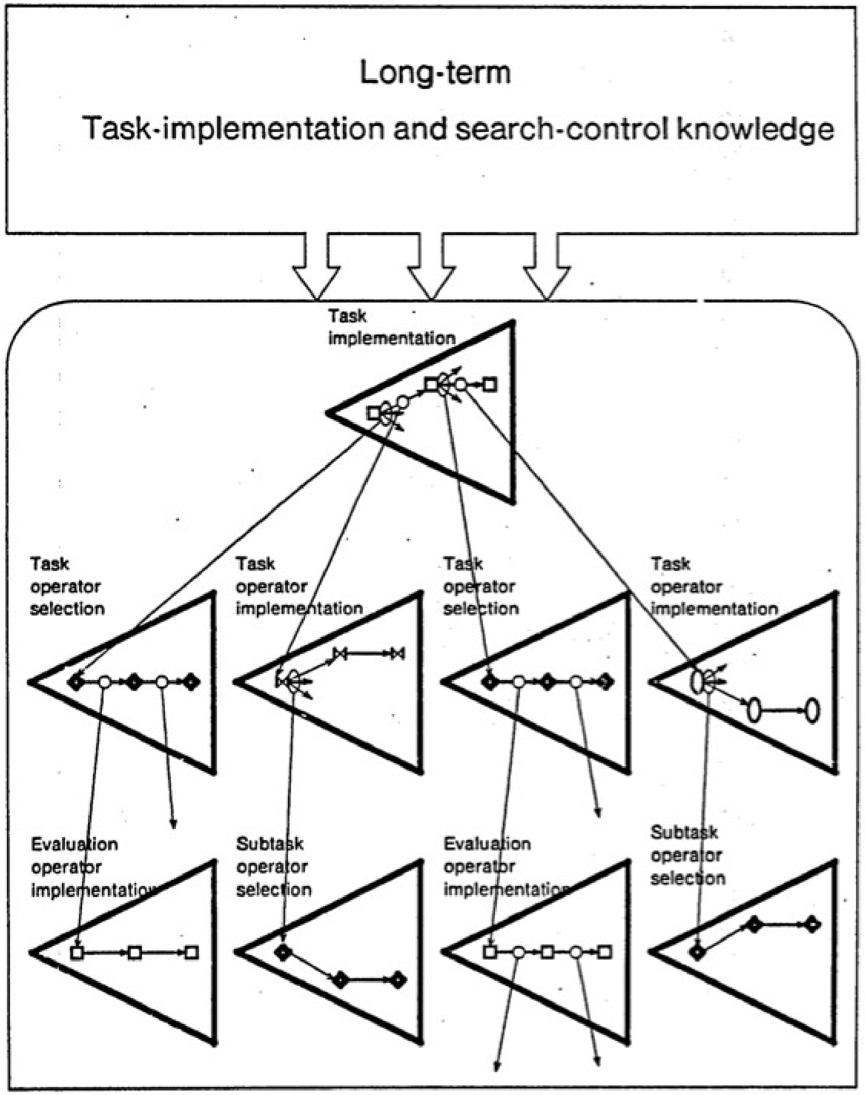
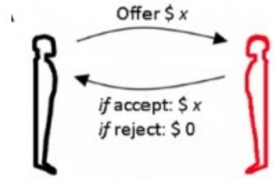


Figure 1-2: The tree of subgoals and their problem spaces.

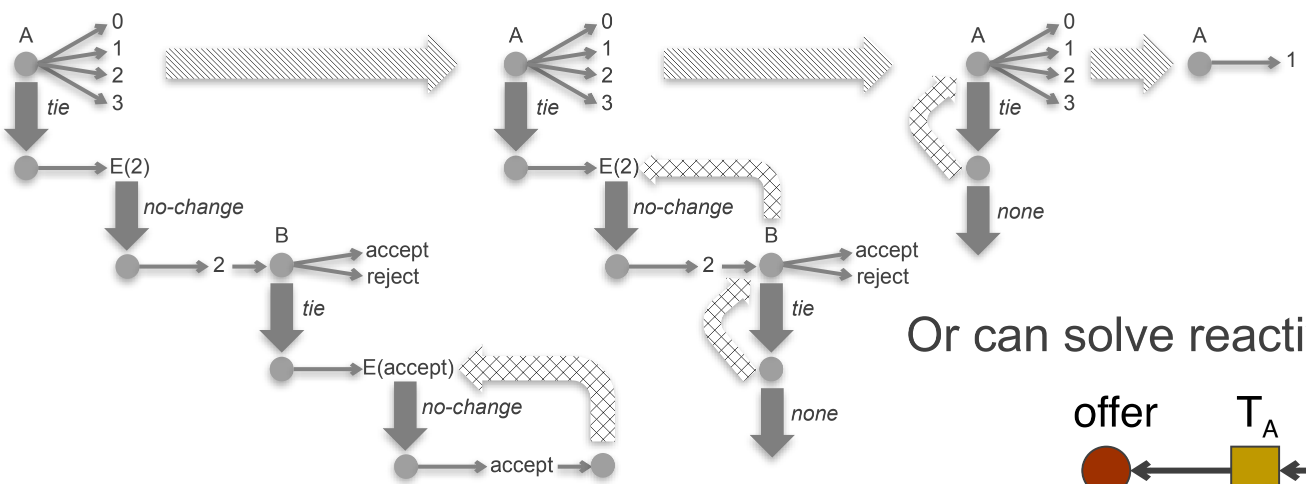
Ultimatum Game



- Multiagent game (Use `init` to specify *multiagent type*)
  - A offers to keep 0, 1, 2 or 3 out of a total of 3, with rest to B
  - B accepts or rejects offer
    - If B accepts, A gets offer and B gets (3 – offer)
    - If B rejects, both get nothing

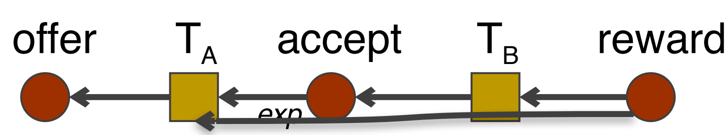
Solves implicit POMDP

Softmax model of B's choice (from reward)



Automatize?

Or can solve reactively via explicit POMDP:



Graphical models

Piecewise-linear functions

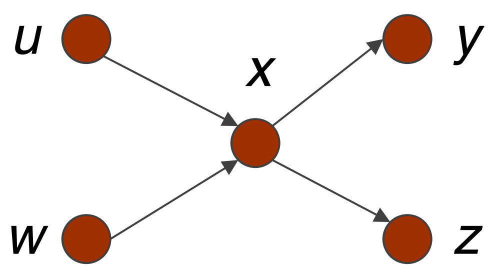
Gradient-descent learning

GRAPHICAL ARCHITECTURE

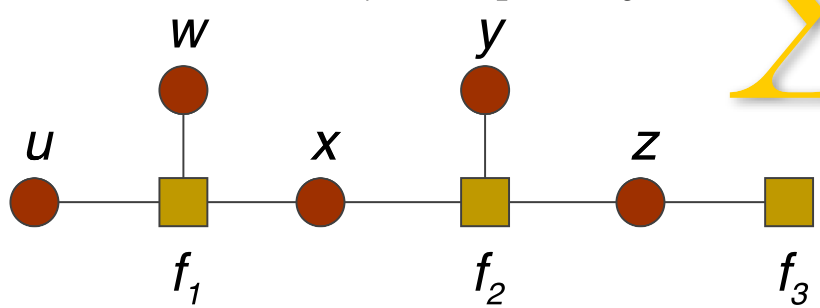
Graphical Models

- Efficient computation over multivariate functions by leveraging forms of independence to decompose them into products of simpler subfunctions
  - Bayesian/Markov networks, Markov/conditional random fields, factor graphs

$p(u,w,x,y,z) = p(u)p(w)p(x|u,w)p(y|x)p(z|x)$



$f(u,w,x,y,z) = f_1(u,w,x)f_2(x,y,z)f_3(z)$



- Typically solve via message passing (e.g., *summary product*) or sampling
  - Can support mixed and hybrid processing
  - Several neural network models map onto them
- Yield broad range of state-of-the-art capability from a uniform base
  - Across *symbols*, *probabilities* & *signals* via uniform representation & reasoning algorithm
    - (Loopy) belief propagation, forward-backward algorithm, Kalman filters, Viterbi algorithm, FFT, turbo decoding, arc-consistency, production match, …

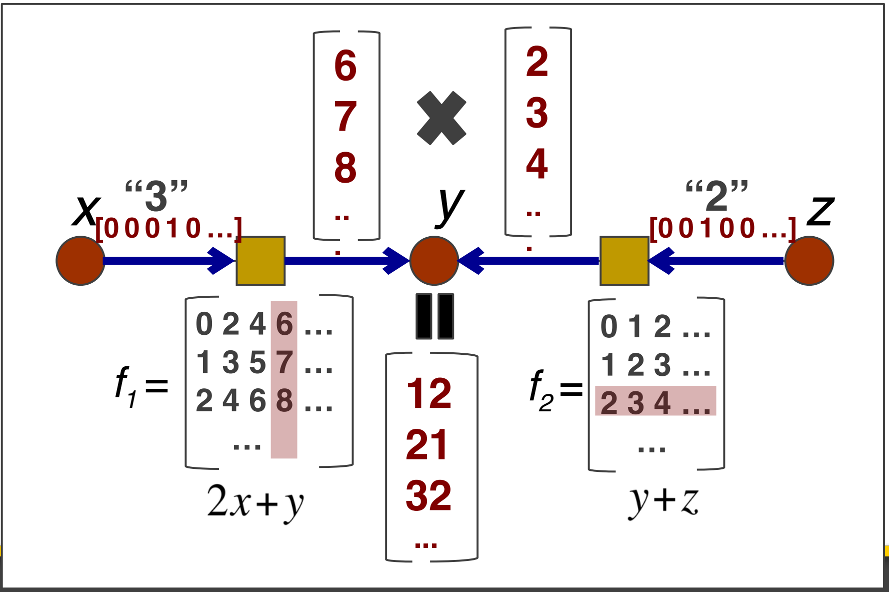
➔ **major potential for satisfying all three desiderata**

(Factor Graphs and) Summary Product Algorithm

- Compute variable marginals (*sum-product*/*integral-product*) or mode of entire graph (*max-product*)
- Pass messages on links and process at nodes
  - Messages are distributions over link variables (starting w/ *evidence*)
  - At variable nodes messages are combined via *pointwise product*
  - At factor nodes do products, and summarize out unneeded variables:

$$m(y) = \int_x m(x) \times f_1(x,y)$$

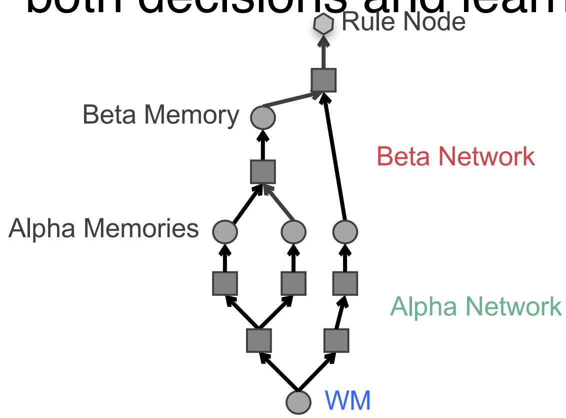
$$f(x,y,z) = y^2 + yz + 2yx + 2xz$$
$$= (2x+y)(y+z) = f_1(x,y) f_2(y,z)$$



In Sigma, both functions and messages are piecewise linear

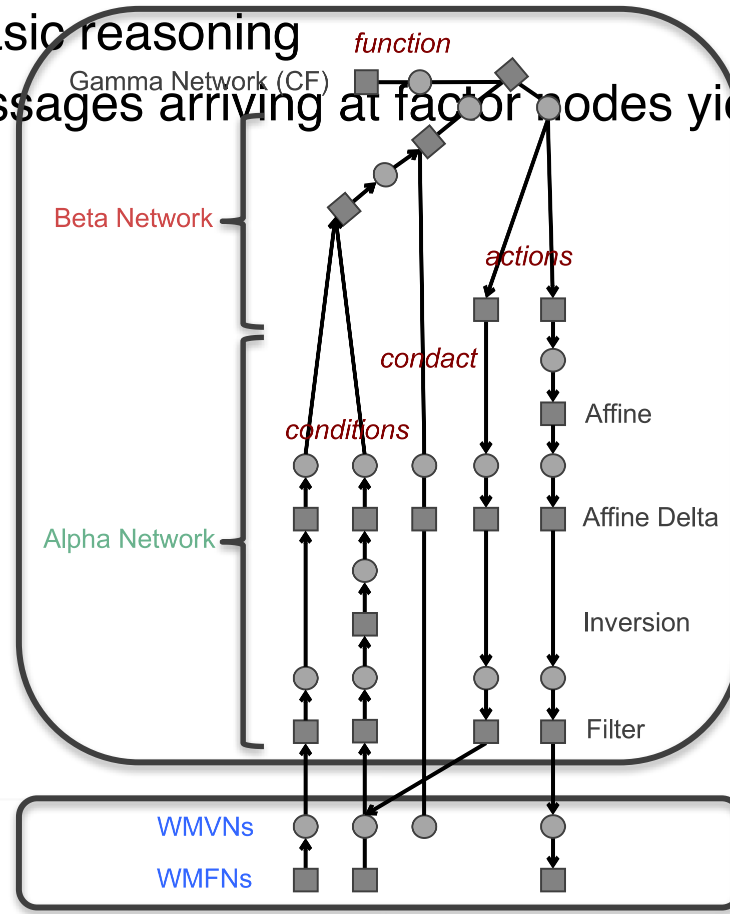
Relationship Back to Cognitive Architecture

- Predicates and conditionals compile into portions of factor graph
  - *Graph solution* via passing of piecewise-linear messages yields both long-term memory access and basic reasoning
  - *Graph modification* based on messages arriving at factor nodes yields both decisions and learning



Rete for rule match

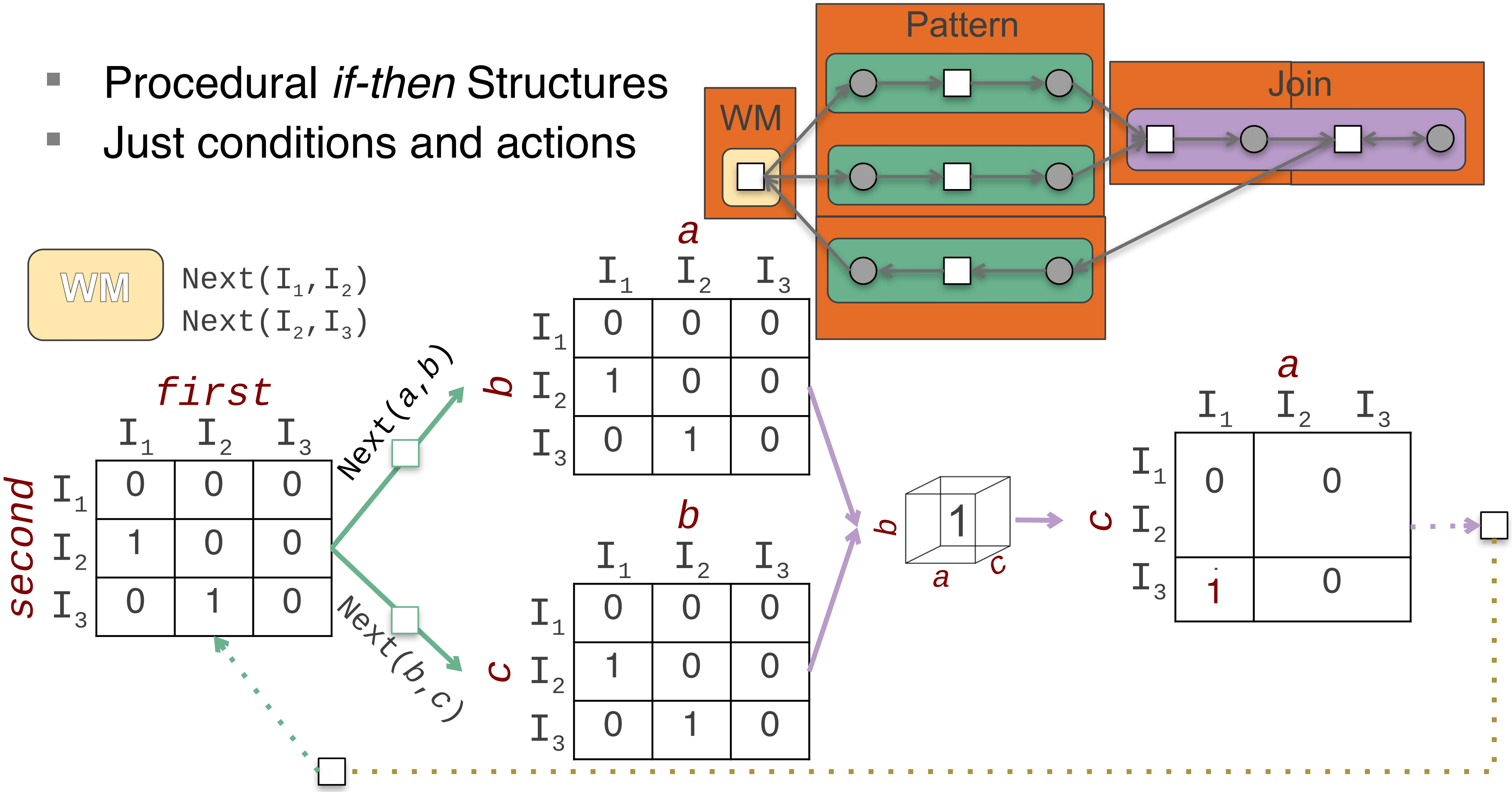
C1 & C2 & C3 ➔ A1 & A2



Procedural Memory (Rules)

```
CONDITIONAL Transitive
   Conditions: Next(a,b)
               Next(b,c)
   Actions: Next(a,c)
```

- Procedural *if-then* Structures
- Just conditions and actions

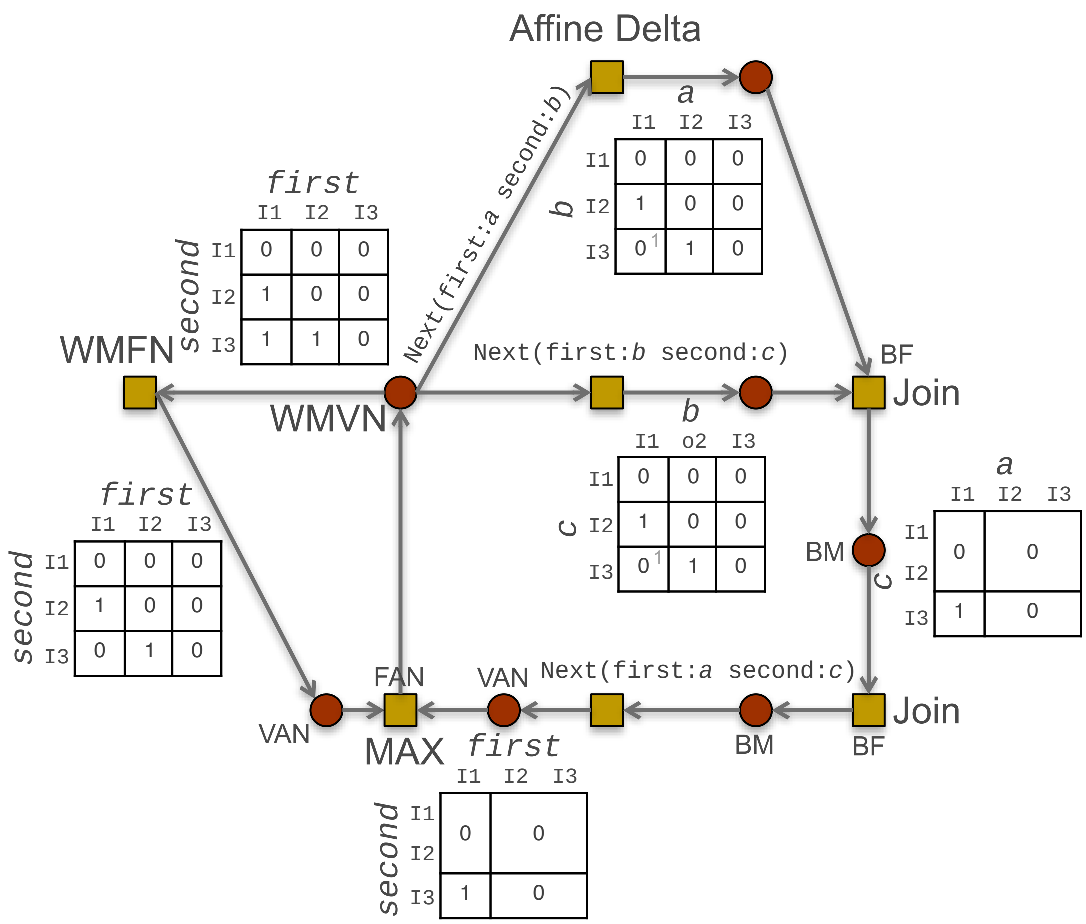



```
(type 'ID :constants '(I1 I2 I3))
(predicate 'Next '((first ID) (second ID)) :world 'closed)
```

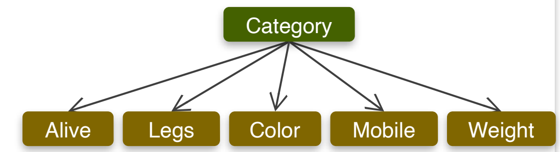
Procedural Memory (Rules)

In More Detail

CONDITIONAL *Transitive*
Conditions: Next(*a*,*b*)
Next(*b*,*c*)
Actions: Next(*a*,*c*)



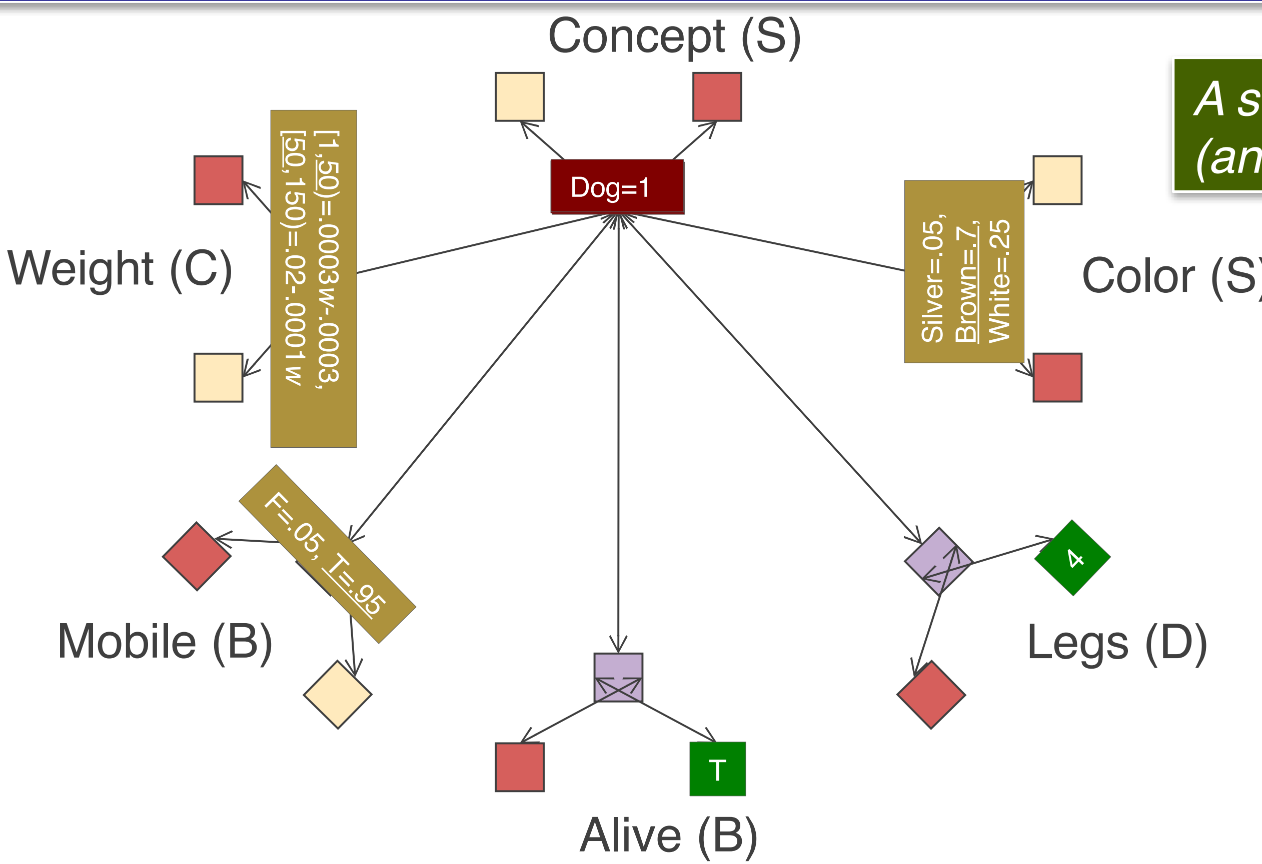
Semantic Memory (Classifier)



Naïve Bayes classifier

Given cues, **retrieve/predict** object category and missing attributes

E.g., **Given** *Alive*=T & *Legs*=4 **Retrieve** *Category*=Dog, *Color*=Brown, *Mobile*=T, *Weight*=50



A subset of factor nodes (and no variable nodes)

Function

WM

Join

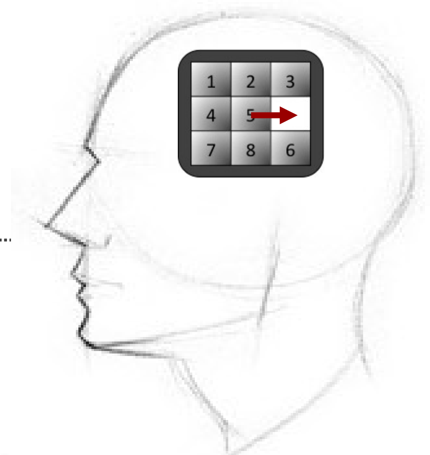
B: Boolean

S: Symbolic

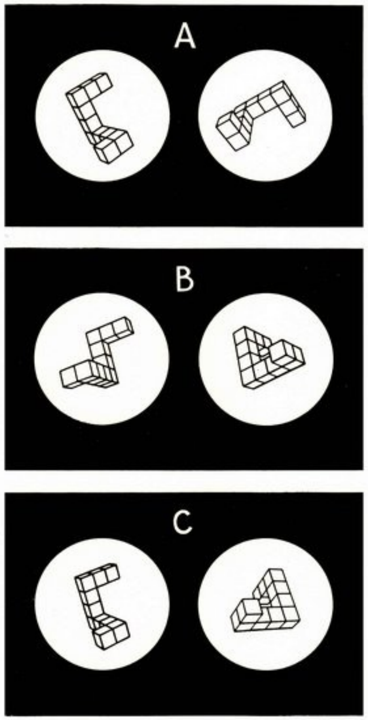
D: Discrete

C: Continuous

Imagery Memory (Mental Imagery)



- How is spatial information represented and processed in minds?
  - Add and delete objects from images
  - Aggregate combinations into new objects
  - Translate, scale and rotate objects
  - Extract implied properties for further reasoning
- In a symbolic architecture either need to
  - Represent and reason about images symbolically
  - Connect to an imagery component (as in Soar 9)
- In Sigma, use its standard mechanisms
  - Continuous, discrete and hybrid representations
  - *Affine transform nodes* that are special purpose optimizations of general factor nodes



Special purpose optimization of standard factor node that operates on variables/dimensions & their region boundaries

Affine Transforms

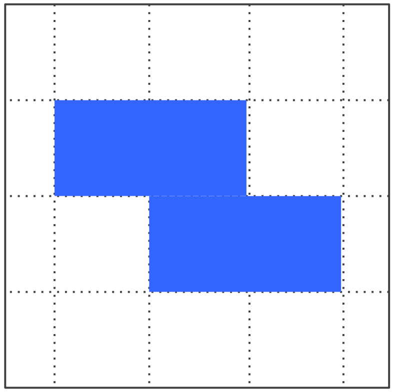
- Translation: Addition (offset)
  - Negative (e.g., y + -3.1 or y – 3.1): Shift to the left
  - Positive (e.g., y + 1.5): Shift to the right

- Scaling: Multiplication (coefficient)
  - <1 (e.g. ¼ × y): Shrink
  - >1 (e.g. 4.37 × y): Enlarge
  - -1 (e.g., -1 × y or -y): Reflect
  - *Requires translation as well to scale around object center*

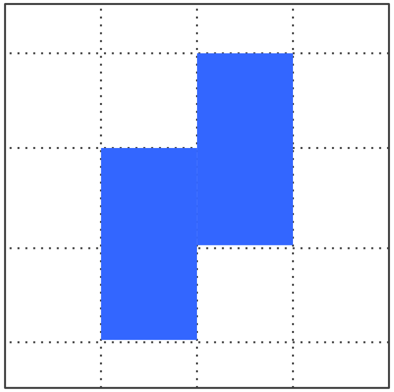
- Rotation (by multiples of 90°): Swap dimensions
  - $x \rightleftarrows y$
  - *In general also requires reflections and translations*

Yields a form of primitive mental arithmetic

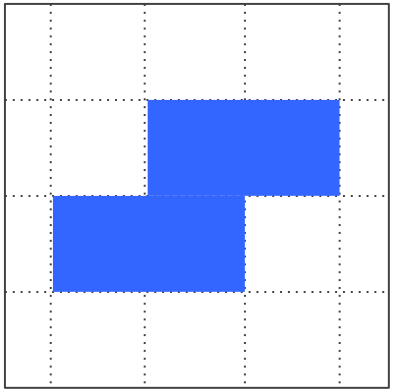
Transform a *Z Tetromino* (via Affine Nodes)



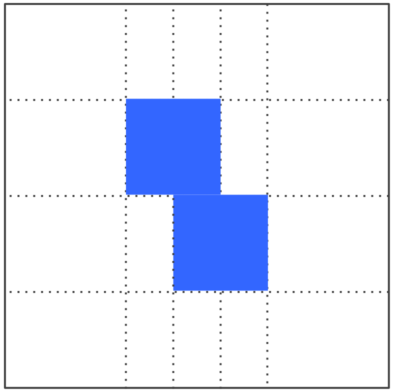
```
CONDITIONAL Rotate-90-Right
    Conditions: (tetromino x:x y:y)
    Actions: (tetromino x:4-y y:x)
```



```
CONDITIONAL Reflect-Horizontal
    Conditions: (tetromino x:x y:y)
    Actions: (tetromino x:4-x y:y)
```



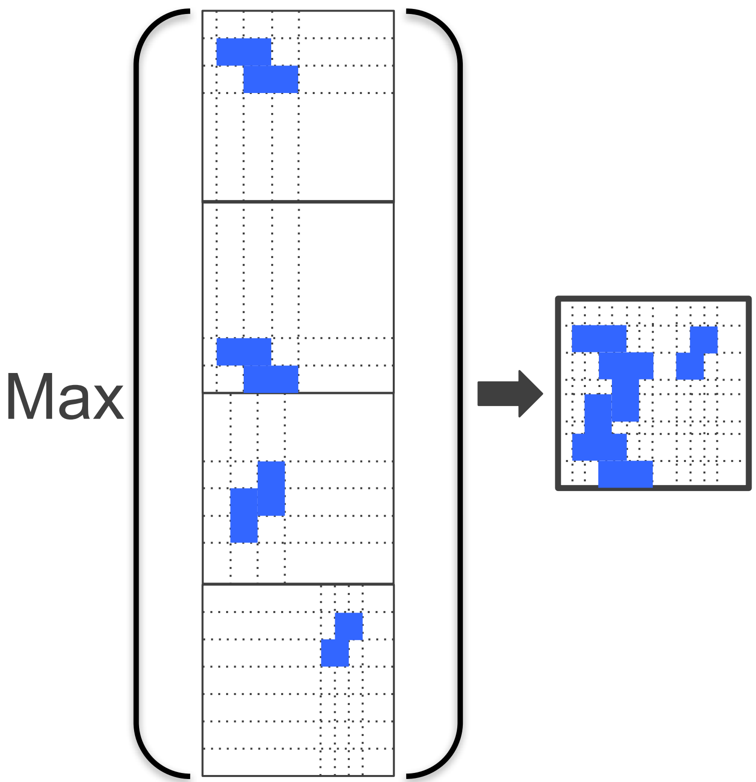
```
CONDITIONAL Scale-Half-Horizontal
    Conditions: (tetromino x:x y:y)
    Actions: (tetromino x:x/2+1 y:y)
```



Composition and Extraction

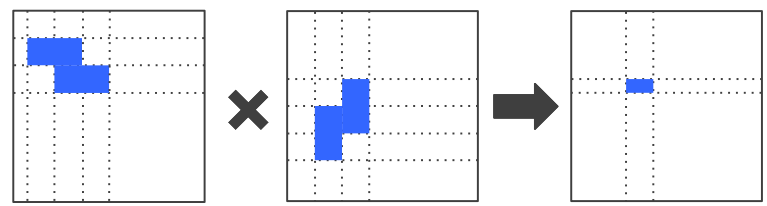
Object Composition

```
CONDITIONAL Union
   Conditions: (Image object:o x:x y:y)
   Actions: (Composite x:x y:y)
```



Overlap Detection

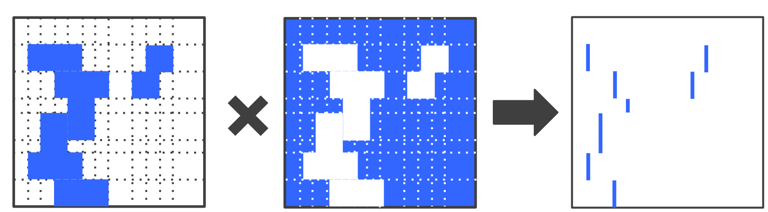
```
CONDITIONAL Ovelap-0-1
   Conditions: (Image object:0 x:x y:y)
               (image object:1 x:x y:y)
   Actions: (Overlap overlap:0 x:x y:y)
```



Edge Extraction

negated condition

```
CONDITIONAL Left-Edge
   Conditions: (Union x:x y:y)
               (Union – x:x-.0001 y:y)
   Actions: (Left-Edge x:x y:y)
```

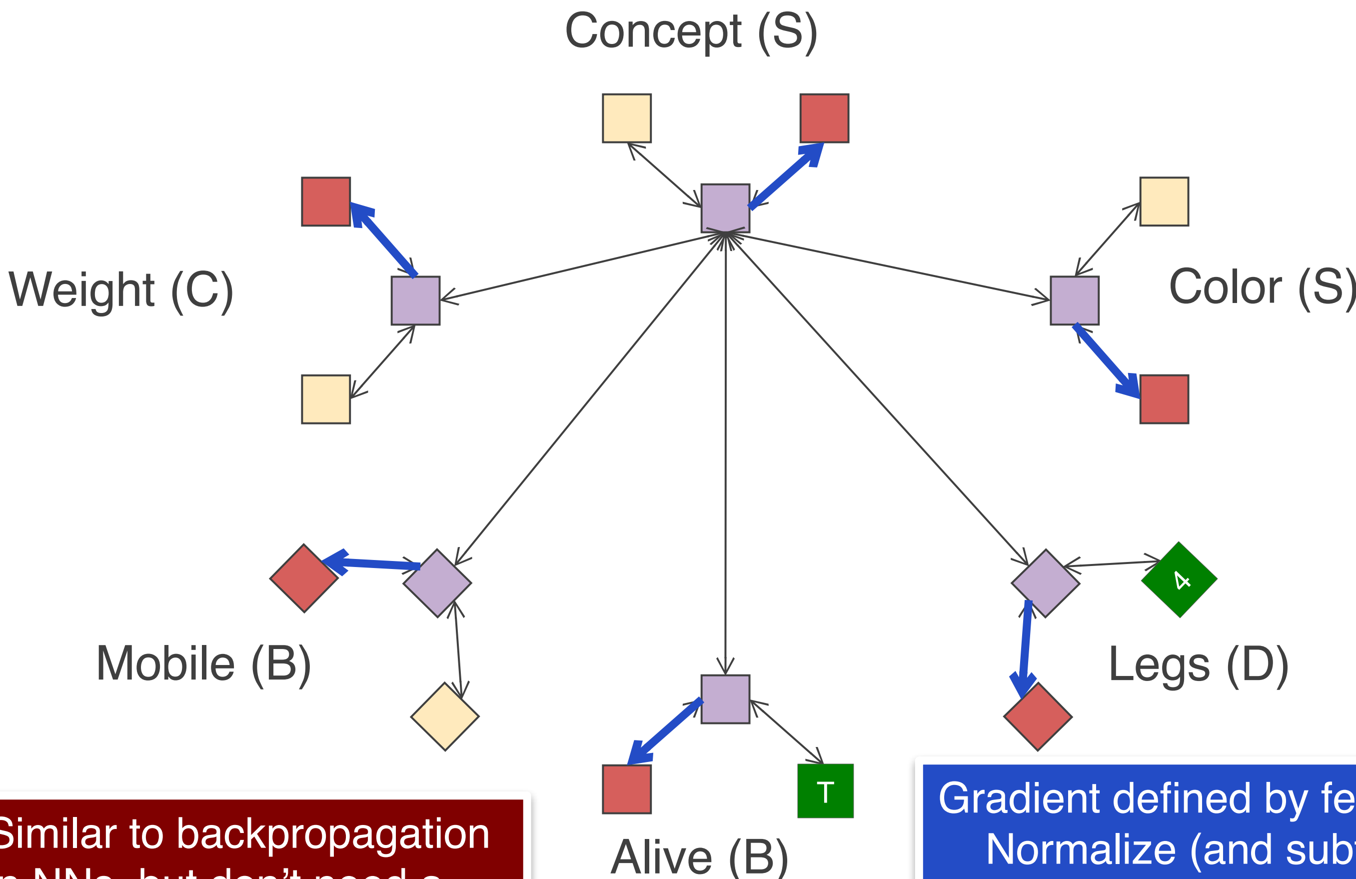


DECISIONS & LEARNING

Decisions at Working Memory Factor Nodes (WMFNs)

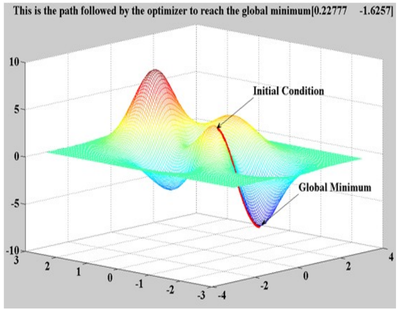
- Choice of *best* alternative at the cognitive level is computed as a side effect of MAX summarization over messages arriving at WMFN nodes
  - As MAX is computed, maximal (sub)regions are tracked for *argmax*
- Choice of *expected value* involves EV summarization
- Choice by *probability matching* involves a variant of INTEGRAL summarization
  - Can also transform function before summarization to yield variations such as Boltzmann/softmax selection

Learning at Function Factor Nodes (FFNs)



Gradient descent

Local, incremental search for optimal weights



Similar to backpropagation in NNs, but don't need a separate backprop phase

Gradient defined by feedback to function node
- Normalize (and subtract out average)

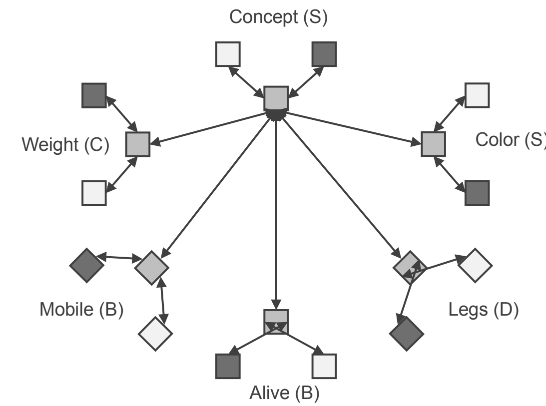
Multiply by learning rate

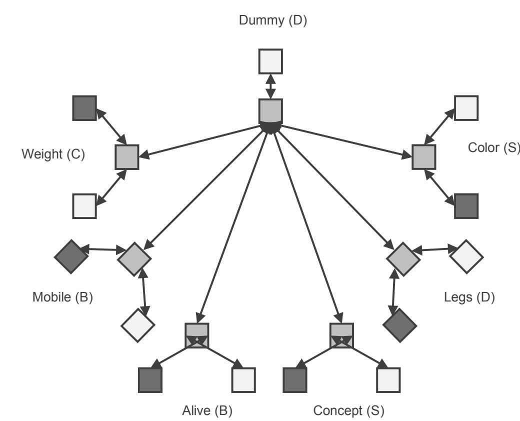
Add to function, smooth and normalize

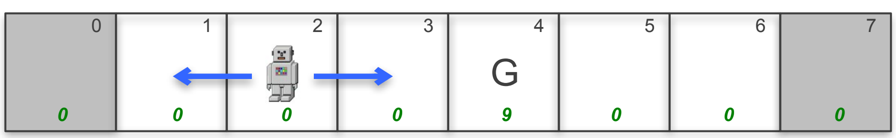
Only function/parameter learning, not structure learning

Learning Examples

- Tools to learn naïve-Bayes classifiers from data
  - Separate train and test sets
  - Supervised or unsupervised
  - Specifiable number of training cycles
- Episodic learning
  - Episodes (values of state predicates at decision time)
- Reinforcement learning learns to predict:
  - Rewards at states
  - Projected future values of states
  - Q values for operators at states
  - (optional) Models of the actions used





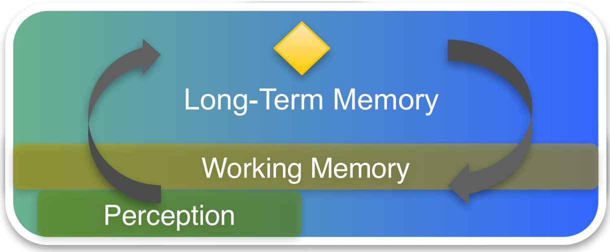


Episodic Memory

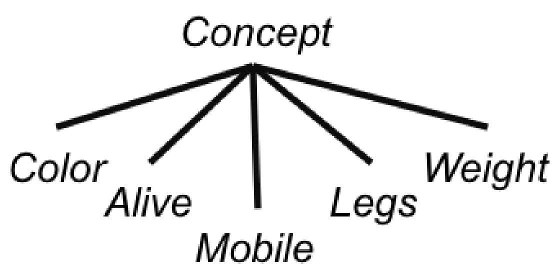
- A core competency in cognition
  - Back at least to Tulving (1983) in psychology
  - Back at least to Vere & Bickmore (1990) in AI
- Spans ability to
  - Store history of what has been experienced
    - Autobiographical and temporal
  - Selectively retrieve and reuse information from past episodes
  - Replay fragments of past history
- Not yet pervasive in cognitive architectures
  - But see work in Soar, Icarus, ACT-R, ..
- General relationship to CBR and IBL

How Episodic Memory and Learning Works in Sigma

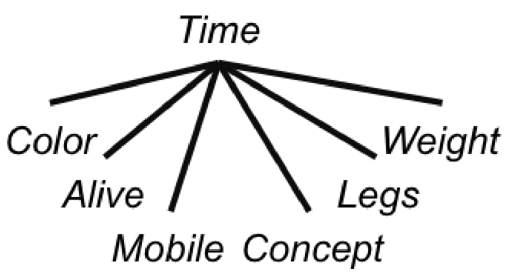
- Episode: Distributions over state predicates at decision time
- Three key processes
  - Learning a new episode
  - Selecting best previous time
  - Retrieving features from selected time



- Naïve Bayes classifier over distributions (like SM) but
  - Time acts as the category
  - MAP/max-product used to retrieve single episode coherently



Semantic

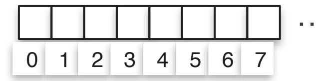


Episodic

```
Conditional Legs-Time*Retrieve
  Conditions: Time*Episodic(value:t)
  Condacts: Legs*Episodic(value:l)
  Function(t,l): Legs-Time*Learn
```

Time as a Category

- Modeled in Sigma as a discrete numeric type



  - Automatically incremented once per cognitive/decision cycle
- Must distinguish *past* from *present*
  - Episode learning depends on *present*
  - Episode selection depends on comparing *past* and *present*
  - Episode retrieval depends on *past*
    - With results then being distinguishable from *present*
- Use related but different predicates & working memory buffers
  - `Time` vs. `Time*Episodic`, `Concept` vs. `Concept*Episodic`, …
- Use one conditional per episodic process per feature
  - Appropriately considering *past* vs. *present* as necessary
  - *Tying* functions together to share what is learned
- Episodic predicates and conditionals generated automatically from *state predicates* such as `Legs`

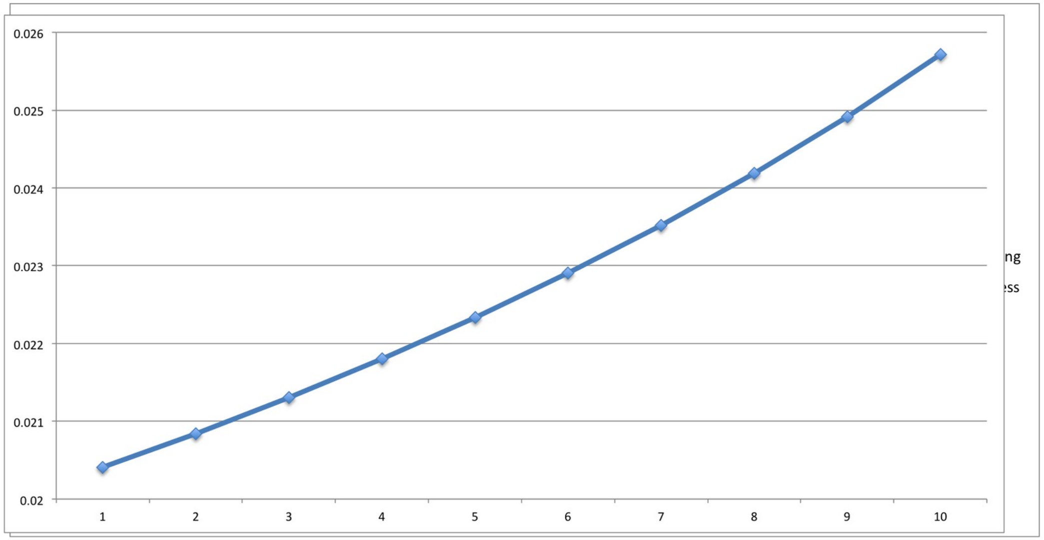
Time as a Function

- Category prior – `Time*Episodic` – for episodic classifier
  - Learning at each cycle (w/ normalization) yields exponential "decay"
  - Episodic selection automatically provides feedback to adjust
    - Implicitly takes into consideration frequency and recency

```
Conditional Time*Access
  Condacts: Time*Episodic(value:t)
  Function(t): Time*Learn

Conditional Time*Learn
  Condacts: Time(value:t)
  Function(t): Time*Learn

Conditional Legs-Time*Select
  Conditions: Legs(value:l)
  Condacts: Time*Episodic(value:t)
  Function(t,l): Legs-Time*Learn
```

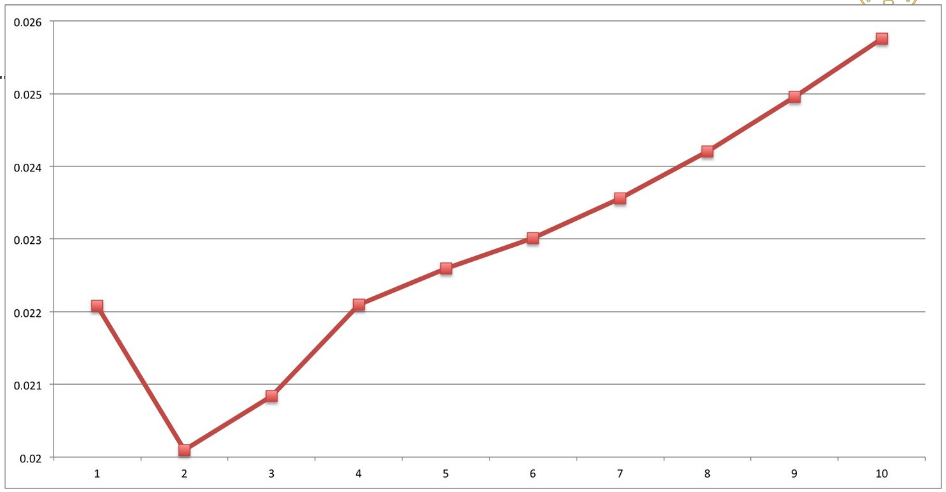


Mimics *base-level activation*!

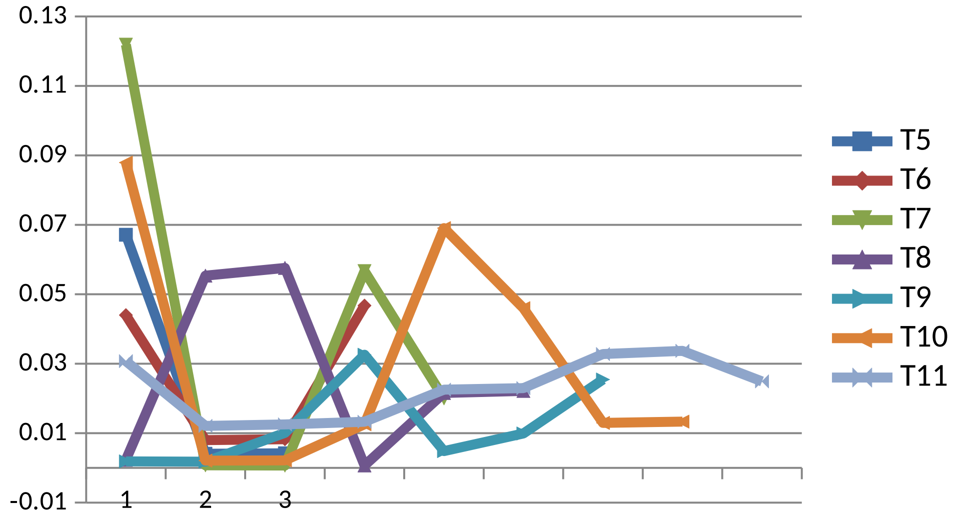
Results

	Concept	*Color*	*Alive*	*Mobile*	*Legs*	*Wgt.*
T1	walker	silver	false	true	4	10
T2	human	white	true	true	2	150
T3	human	brown	true	true	2	125
T4	dog	silver	true	true	4	50

	Queries	Best
T5	*Concept*=walker	T1
T6	*Color*=silver	T4
T7	*Alive*=false, *Legs*=4	T1
T8	*Alive*=false, *Legs*=2	T3
T9	*Concept*=dog, *Color*=brown	T4
T10	*Concept*=walker, *Color*=silver, *Alive*=true	T1
T11	*Alive*=false	T8



- Trades off partial match across multiple cues with temporal prior
- Retrieves all features from single best episode when they exist
- Can replay a sequence deliberately
- Works for more complexly structured tasks too



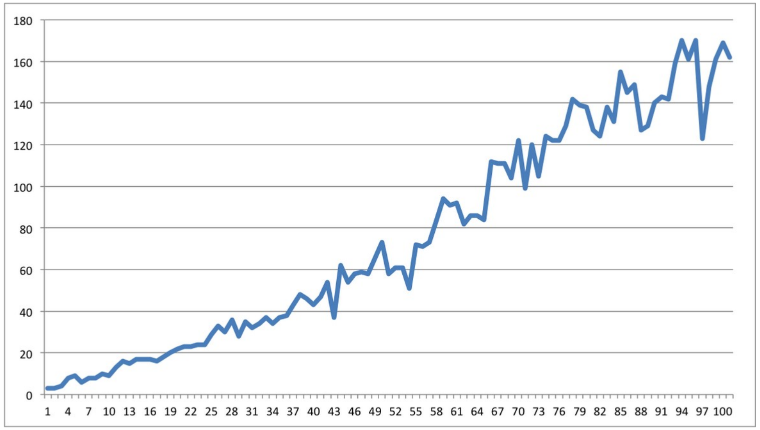
Efficiency

+: Piecewise-linear functions track only changes in memories

	T1	T2–T3	T4
walker	.85	.05	.05
table	.05	.05	.05
dog	.05	.05	.85
human	.05	.85	.05

–: Reprocess entire episodic memory every cycle

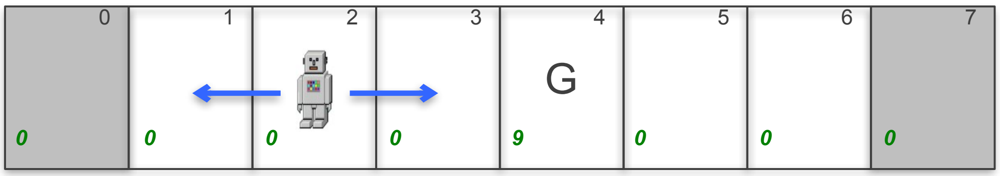
- A function is reprocessed in its entirety if any region in it changes



Time (msec) per cycle over trials

- Implies need for some form of *incremental message processing*

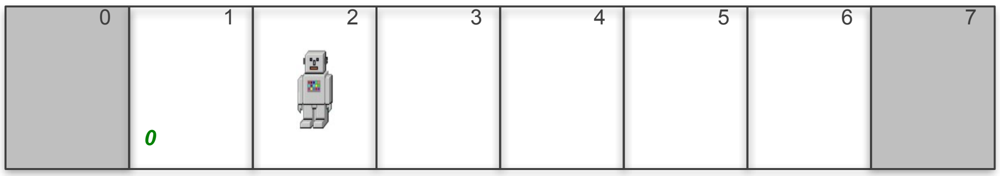
Reinforcement Learning



Learn values of actions for states by backwards propagation of rewards received during exploration:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

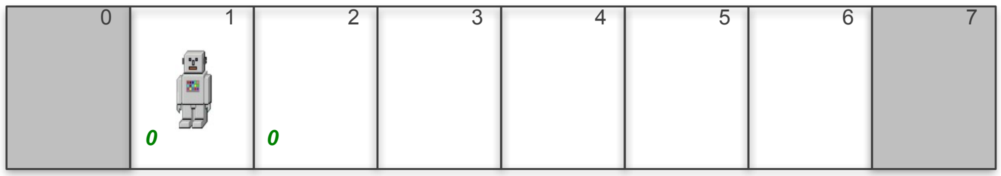
Reinforcement Learning



Learn values of actions for states by backwards propagation of rewards received during exploration:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

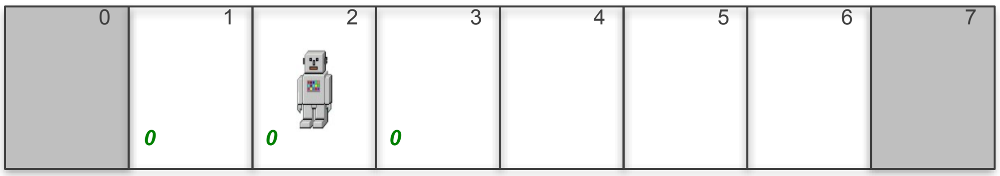
Reinforcement Learning



Learn values of actions for states by backwards propagation of rewards received during exploration:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

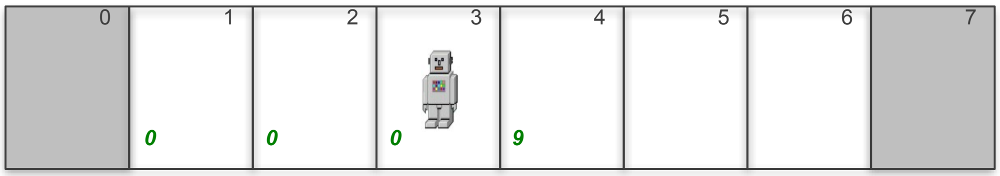
Reinforcement Learning



Learn values of actions for states by backwards propagation of rewards received during exploration:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

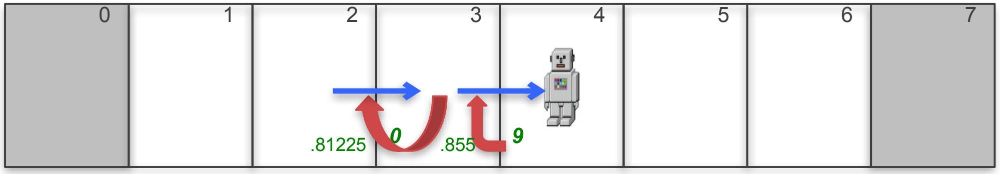
Reinforcement Learning



Learn values of actions for states by backwards propagation of rewards received during exploration:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Reinforcement Learning

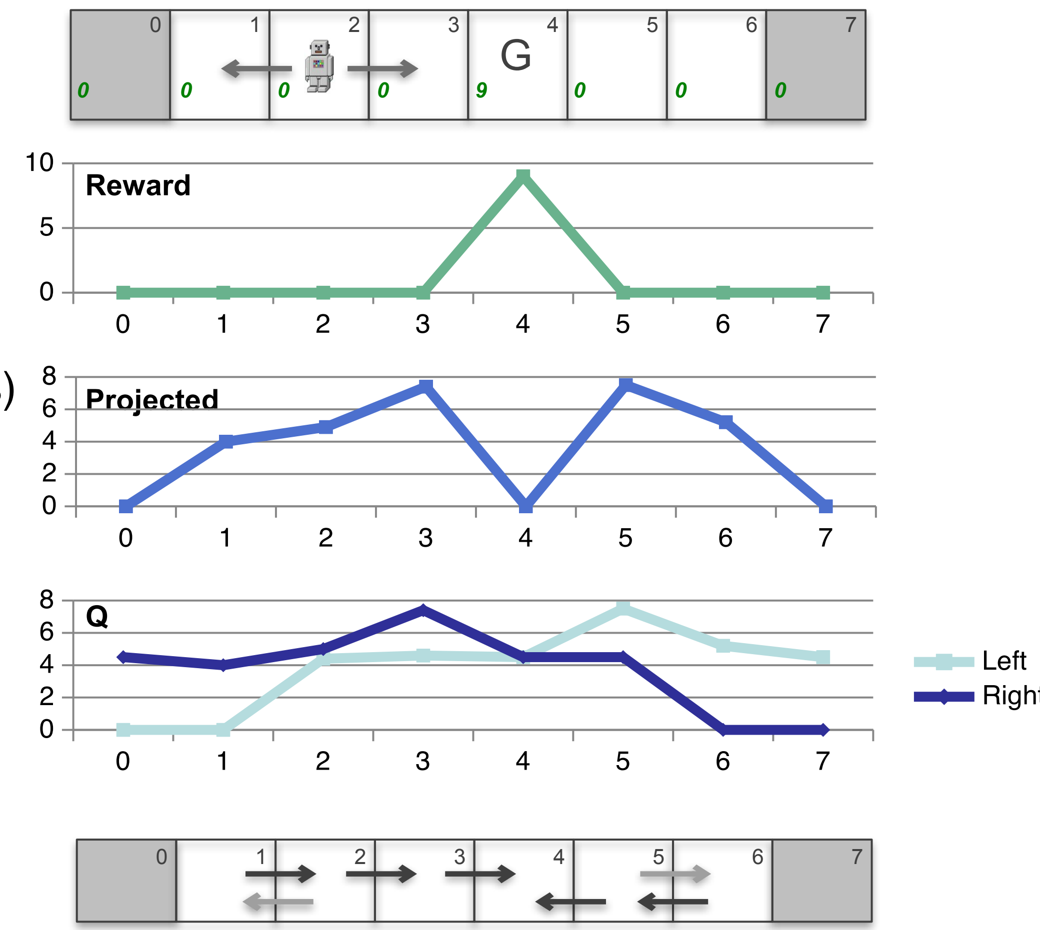


Learn values of actions for states by backwards propagation of rewards received during exploration:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Deconstructing RL in Sigma

- Knowledge:
  - Initial uniform predictors for:
    - Current reward (*R*)
    - Projected future reward (*P*)
    - Action preferences (*Q*)
  - Regression (backup) knowledge
  - Action models (predict next states)
- Supervised learning of:
  - Current reward (*R)*
  - Projected future reward (*P)*
  - Action preferences (*Q*)
- Add *Diachronic cycles* to also learn action models



Graphs are of expected values, but learning is actually of full distributions

Template-Based Structure Creation

- From specifications of core state predicates automatically generate additional types, predicates and conditionals as needed for various forms of learning
- *Synchronic prediction*
  - Map learning in SLAM
  - Acoustic function learning in speech HMM
- *Diachronic prediction*
  - Learning action models in RL
  - Transition function learning in speech HMM
- *Episodic learning*
- *Reinforcement learning*

SUMMARY

Basic User Functions

- Initializing
  - System: `init`
  - Operators: `init-operator`
- Programming
  - Type: `new-type`
  - Predicate: `predicate`
  - Conditional: `conditional`
- Input
  - Evidence: `evidence`
  - Perception: `perceive`
- Executing
  - Messages: `r`
  - Decisions: d
  - Trials: `t`
- Printing
  - Types: `pts`
  - Predicates: `pps`, `ppfs`
  - Conditionals: `pcs`, `pcfs`
  - Functions: `pplm`, `parray`
  - Working memory, `pwm` , `ppwm`
- Graph: `g`
- Debugging
  - Recompute message: `debug-message`
  - Print alpha memories: `pam`
- Learning: `learn`

Overall Progress on Sigma

- Memory [ICCM 10]
  - Procedural (rule)
  - Declarative (semantic/episodic) [CogSci 14]
  - Constraint
  - Distributed vectors [AGI 14a]
- Problem solving
  - Preference based decisions [AGI 11]
  - Impasse-driven reflection [AGI 13]
  - Decision-theoretic (POMDP) [BICA 11b]
  - Theory of Mind [AGI 13, AGI 14b]
- Learning [ICCM 13]
  - Concept (supervised/unsupervised)
  - Episodic [CogSci 14]
  - Reinforcement [AGI 12a, AGI 14b]
  - Action/transition models [AGI 12a]
  - Models of other agents [AGI 14b]
  - Perceptual (including maps in SLAM)
- Mental imagery [BICA 11a; AGI 12b]
  - 1-3D continuous imagery buffer
  - Object transformation
  - Feature & relationship detection
- Perception
  - Object recognition (CRFs) [BICA 11b]
  - Isolated word recognition (HMMs)
  - Localization [BICA 11b]
- Natural language
  - Question answering (selection)
  - Word sense disambiguation [ICCM 13]
  - Part of speech tagging [ICCM 13]
- Graph integration [BICA 11b]
  - CRF + Localization + POMDP
- Optimization [ICCM 12]

Some of these are still just beginnings

Current and Near Future Topics

- Scaling up knowledge and learning
- Continuous speech understanding, and its integration with language and cognition
- Distributed vector representations and their role in (integrating) speech, language and cognition
- Emotion/affect and its relationship to the architecture
- Learning of models of others
- Lower architectural levels
- Adaptive virtual humans

Broad Set of Capabilities from Space of Variations

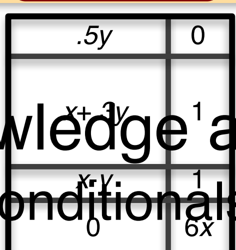
Highlighting *Functional Elegance* and *Grand Unification*

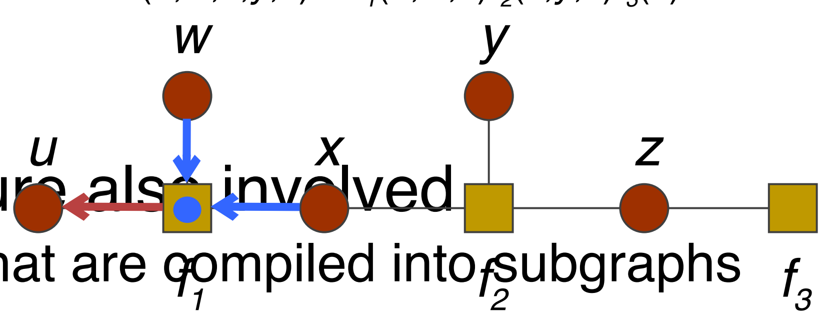
- Rule memory
- Preference-based decisions
- Episodic memory
- POMDP-based decisions
- Semantic memory
- Localization
- Mental imagery
- Edge detectors

Closed vs. open world functions
Universal vs. unique variables
Discrete vs. continuous variables
Boolean vs. numeric function values

Uni- vs. bi-directional links
Max vs. sum summarization
Long- vs. short-term memory
Product vs. affine factors

$f(u,w,x,y,z) = f_1(u,w,x)f_2(x,y,z)f_3(z)$





- Knowledge above architecture also involved
  - Conditionals and predicates that are compiled into subgraphs

Piecewise Continuous Functions ***Factor graphs w/ Summary Product***

Fundamental Questions about Sigma

- Can full range of capabilities be provided in this manner?
- Can it all be sufficiently efficient for real time behavior?
- What are the functional gains?
- Can the human mind (and brain) be modeled?

Wrapping Up

- Sigma website is http://cogarch.ict.usc.edu
  - Most papers on Sigma can be found through there
  - New papers on which I'm an author usually appear online sooner at http://cs.usc.edu/~rosenblo/pubs.html
- Full use of Sigma requires a non-free version of LispWorks
  - The free version imposes heap-size limits that are problematic for anything other than small programs
  - We will soon have a version without the graph, regression and parallel processing interfaces that should run in any version of Lisp
- Sigma is open source (simplified BSD license)
  - We are not yet distributing it openly because of a lack of appropriate documentation, but we are beginning to make progress on this
  - We will consider special requests in the interim