USC Institute for Creative Technologies

University of Southern California

Tutorial

Σ The Sigma Cognitive Architecture/System

Paul S. Rosenbloom & Volkan Ustun

5.9.2016

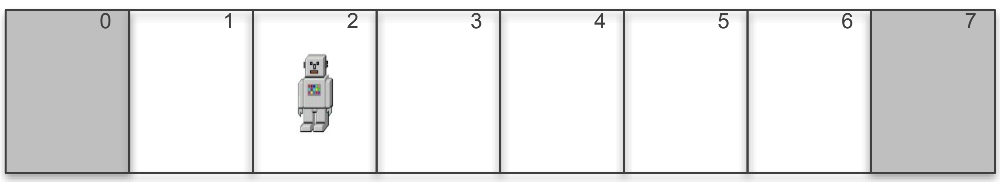
The work depicted here was sponsored by the U.S. Army, the Air Force Office of Scientific Research and the Office of Naval Research. Statements and opinions expressed do not necessarily reflect the position or the policy of the United States Government, and no official endorsement should be inferred.

First hands-on tutorial on Sigma

Goal of this Tutorial

Feel free to ask questions at any time

- Describe and provide insight into Sigma:
  - What it is about
  - How it works
  - What it is capable of
- Much of it from perspective of virtual agents



- Mixed presentation, demonstration, and hands on
  - Execution but not programming
- Complements what can be found in papers

Outline

- Introduction
  - The basics of Sigma
- Hands on
  - Sigma as the mind of an agent on a grid
  - A sequence of random walks of increasing functionality
- Additional topics
  - Rule memory (& mapping to graph), mental imagery, distributed vectors, episodic memory, appraisal & attention, Theory of Mind (& multiagent systems), and interactive adaptive virtual humans
- Summary

Setup Instructions

WIFI Setup
Ssid: waterfront
User: aamas   pwd: aamas2016

- To participate directly in the hands-on portion of this tutorial you will need to have LispWorks installed
  - If not, you can still watch the demos as we go through them but won't be able to do the same yourself
- There is a free trial version available for download:
  - http://www.lispworks.com/downloads/index.html
  - It is sufficient for our purposes here, but does have limitations:
    - A limited heap size
    - A limit to five hours per session
    - It is slower
- It may take a while to download, so please start it now
  - Further instructions will be forthcoming during hands-on portion

LispWorks Personal Edition Download Screen Capture

http://www.lispworks.com/downloads/index.html

Downloading LispWorks® Personal Edition

Please provide us with some information about yourself.

Download LispWorks for * Please choose platform

Your name *

Organization

Email address *

Subscribe to the LispWorks user group?
(You will receive a confirmation message, which you should reply to.)

How long is it since you first used Lisp? Please choose one

How did you hear about LispWorks? Please choose one

What will you use LispWorks for? Please choose one

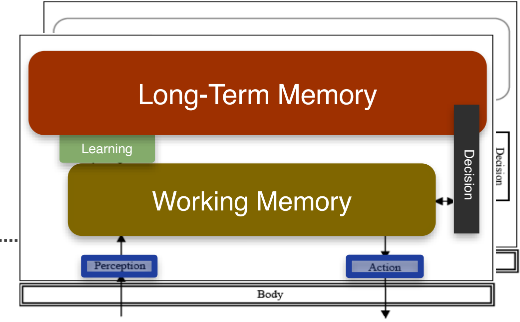
Any other comments you would like to make?

Items marked * are required.

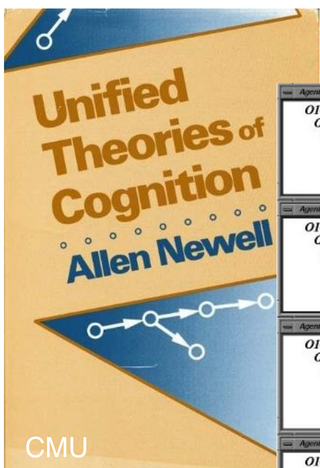
Proceed to Download

INTRODUCTION

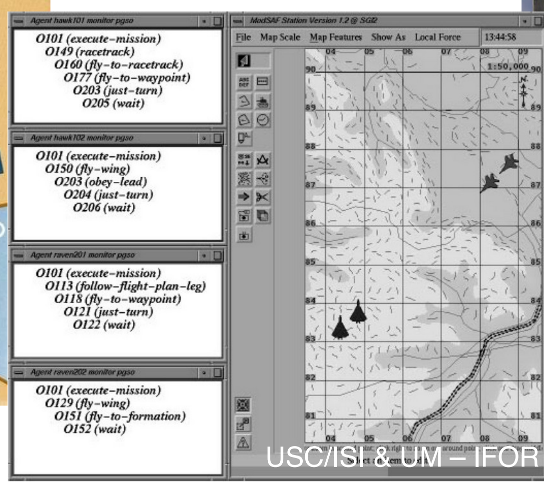
Cognitive Architecture



- Model of the fixed structure of a/the mind
  - Memory, reasoning, learning, interaction, ...
  - Integration across these capabilities
- Supports knowledge and skills above the architecture

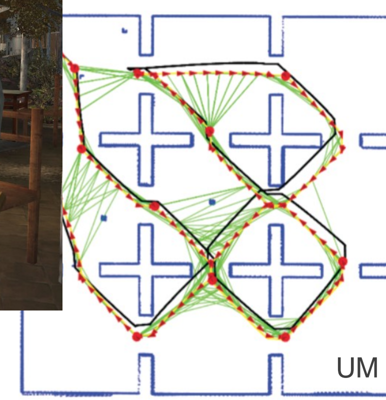


CMU



USC/ISI & UM – IFOR


USC/ICT – SASO


UM

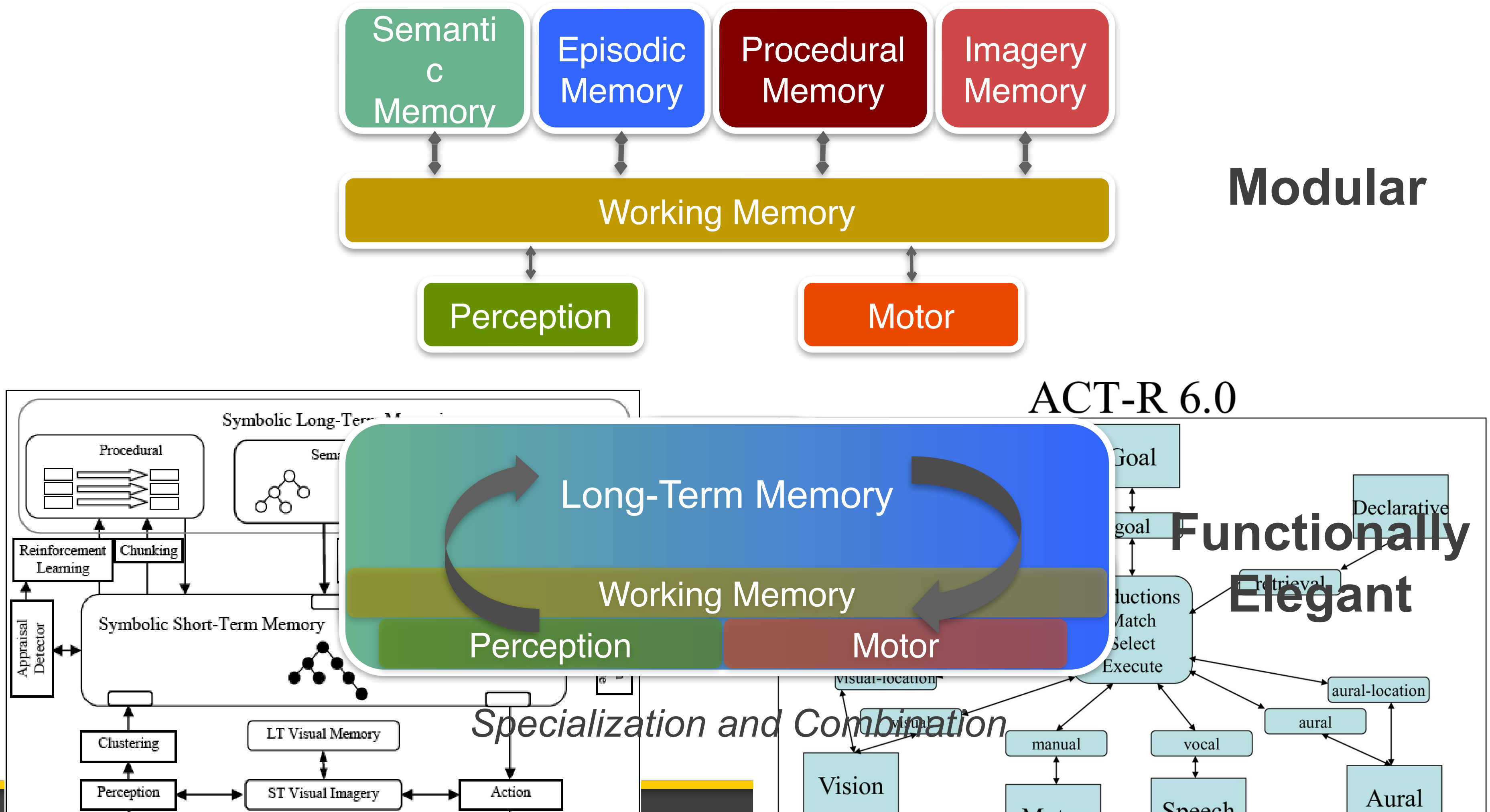
Examples are from Soar, which I co-led for 15 years

Overall Desiderata for the Sigma (Σ) Architecture

- A new breed of cognitive architecture that is
  - *Grand unified*
    - Cognitive + key non-cognitive (perceptuomotor, affective, attentive, …)
  - *Generically cognitive*
    - Spanning both natural and artificial cognition
  - *Functionally elegant*
    - Broadly capable yet simple and theoretically elegant
      - "cognitive Newton's laws"
  - *Sufficiently efficient*
    - Fast enough for anticipated applications
- For virtual humans & intelligent agents/robots that can
  - Think – Broadly, deeply and robustly *cognitive*
  - Behave – *Interactive* with their physical and social worlds
  - Learn – *Adaptive* given their interactions and experience

Hybrid: Discrete + Continuous
Mixed: Symbolic + Probabilistic

Modular versus Functionally Elegant

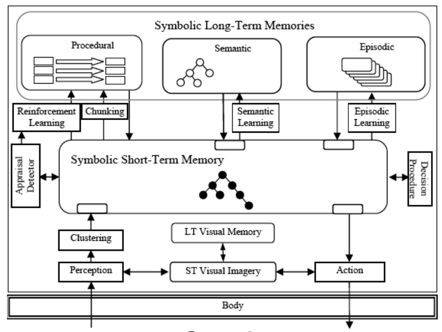


Goal: Advancing *elegance*, *depth* and *breadth*

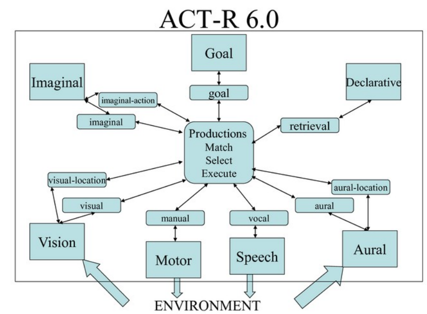
Approach: Graphical Architecture Hypothesis

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

Cognitive Architectures

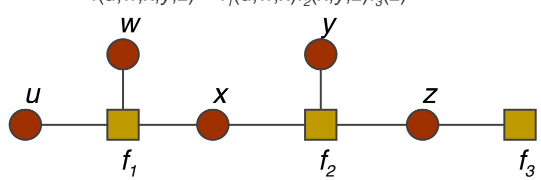


Soar 9



Graphical Models

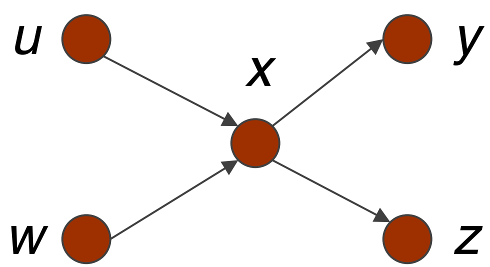
$f(u,w,x,y,z) = f_1(u,w,x)f_2(x,y,z)f_3(z)$



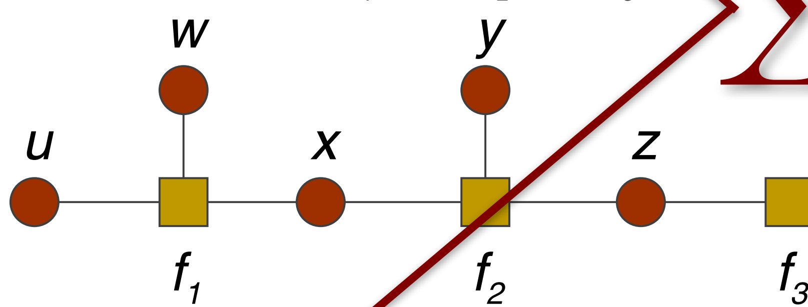
Graphical Models

- 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

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



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



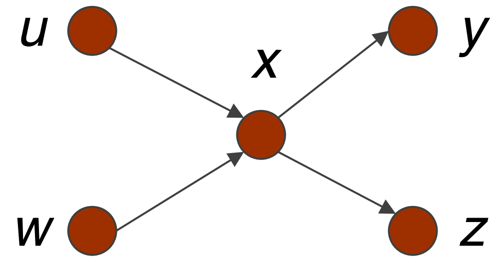
Σ

- Solve typically via some form of message passing or sampling
- State of the art performance across *symbols*, *probabilities* and *signals* from uniform representation and reasoning algorithm
  - (Loopy) belief propagation, forward-backward algorithm, Kalman filters, Viterbi algorithm, FFT, turbo decoding, arc-consistency, production match, …
- Can support mixed and hybrid processing
- Several neural network models map directly onto them

Bayesian Network vs. Factor Graph

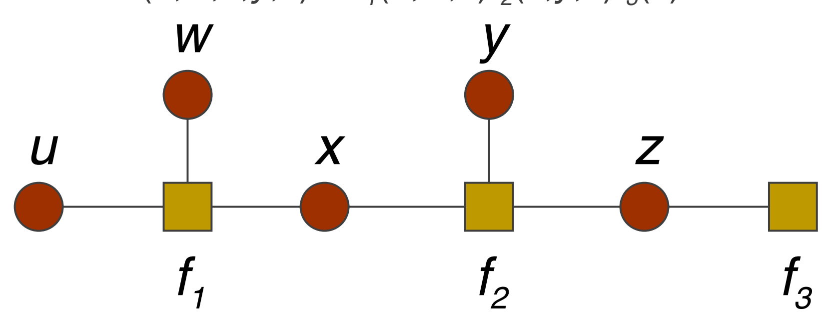
- Bayesian network
  - Directed graph
  - Only variable nodes
  - A function at each node *n*
    - $p(n \mid parents_n)$
  - Decompose probabilities

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



- Factor graph
  - Undirected graph
  - Variable and factor nodes
  - A function at each factor node *n*
    - $f_n(vs_n)$
  - Decompose arbitrary functions

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

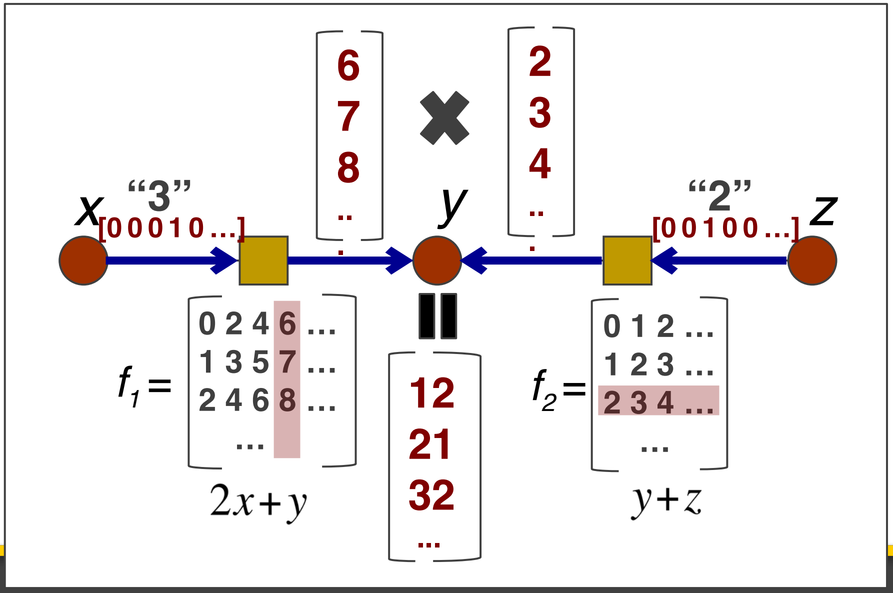


Summary Product Algorithm

- 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

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*

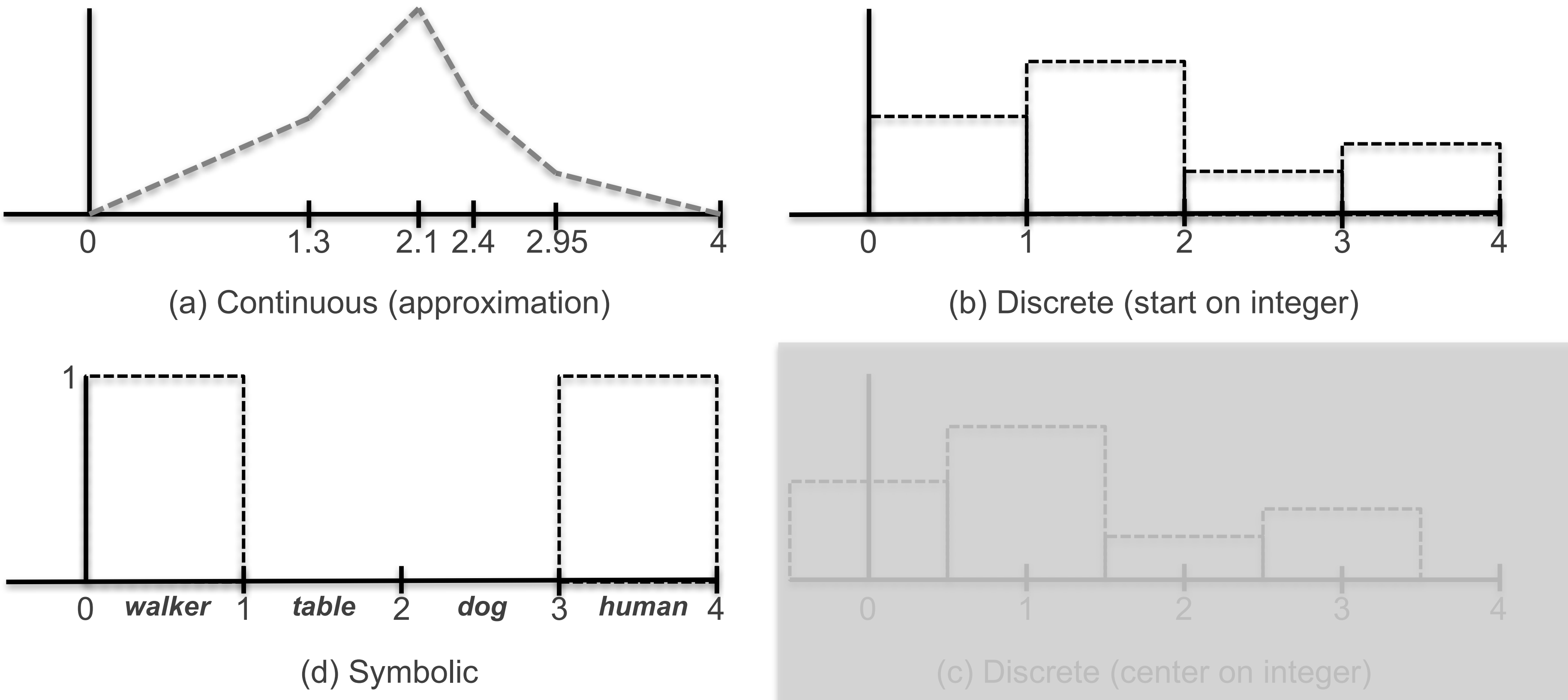
O_1	Brown	Silver	White
T	1	0	
F	0		

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 Linear Functions



(a) Continuous (approximation)

(b) Discrete (start on integer)

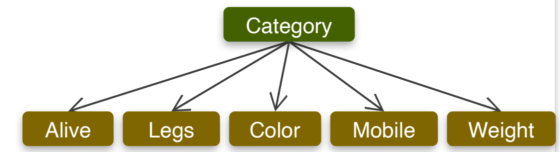
(d) Symbolic

(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)

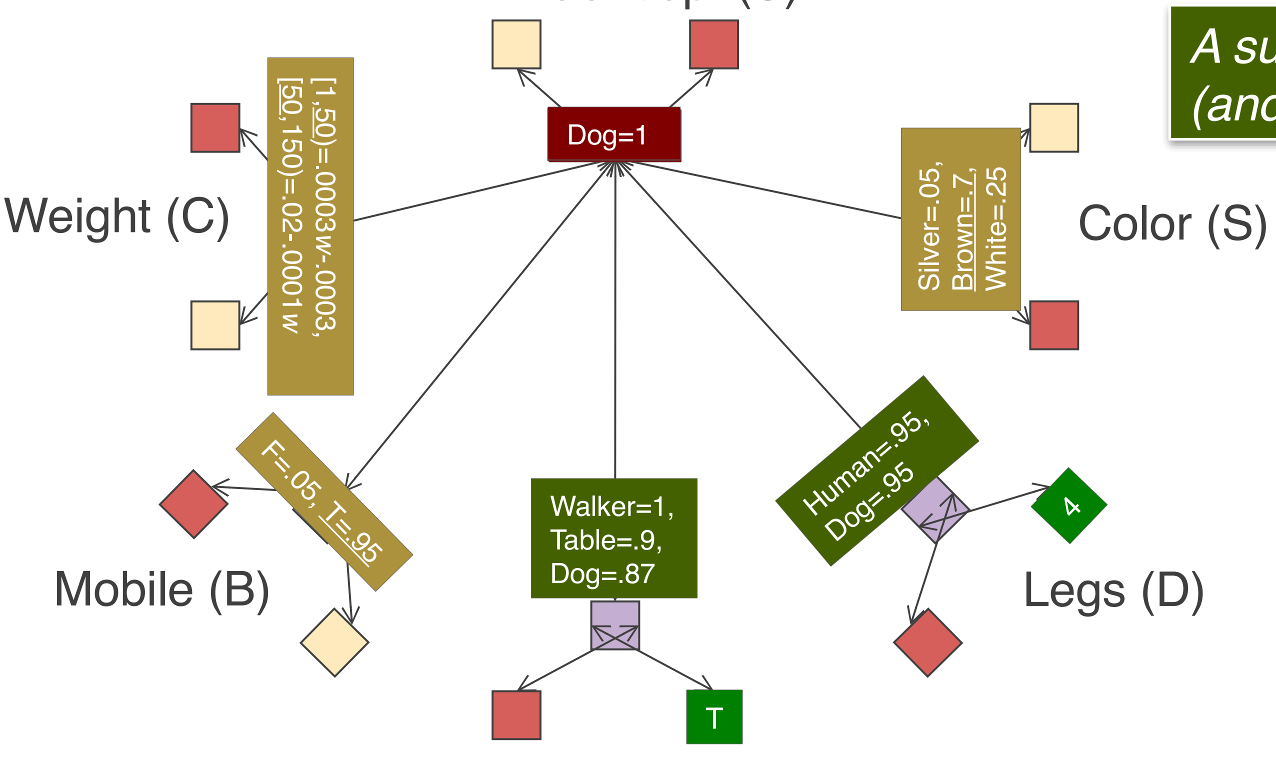
SPA in a Naïve Bayes Classifier



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



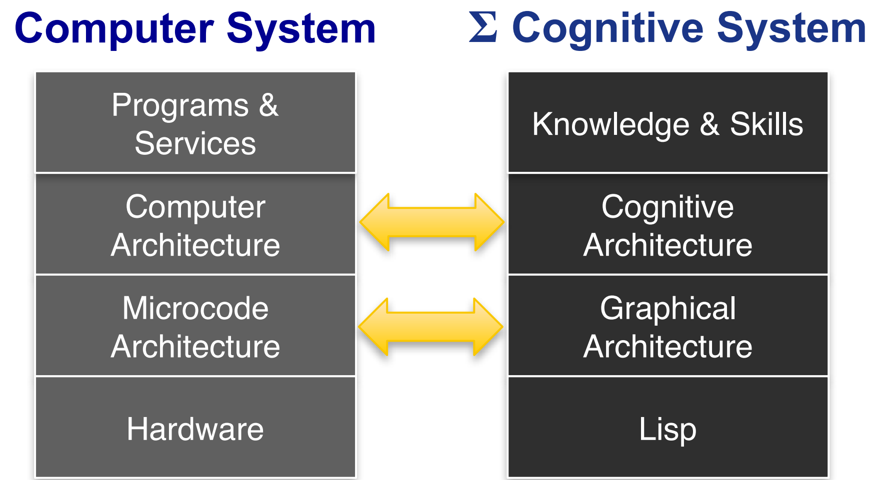
A subset of factor nodes (and no variable nodes)

- Function
- Perception
- Join

B: Boolean
S: Symbolic
D: Discrete
C: Continuous

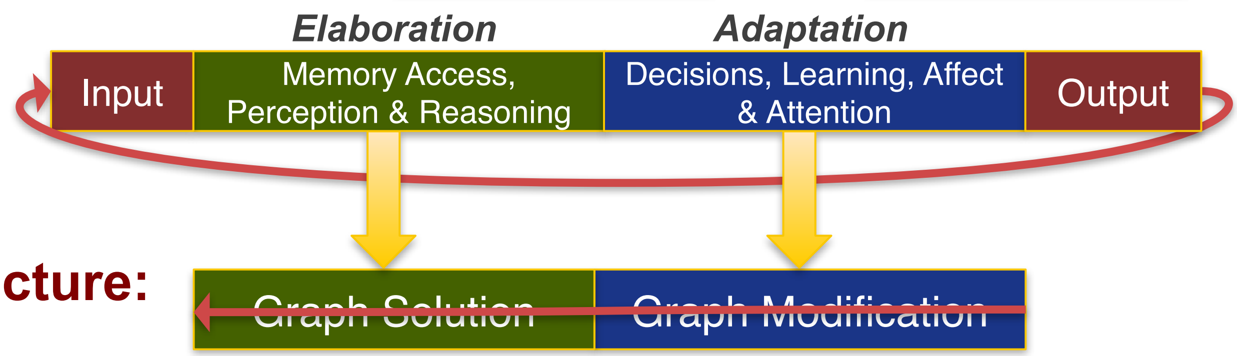
The Structure of Sigma

Σ



Cognitive Architecture:

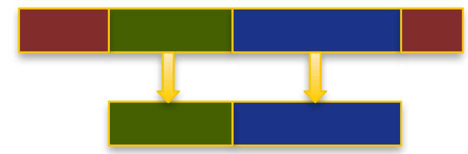
Predicates
Conditionals
Nested tri-level control



Graphical Architecture:

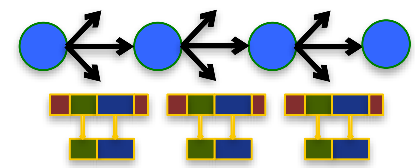
Graphical models
Piecewise linear functions
Gradient-descent learning

(Soar-like) Nested Tri-Level Control



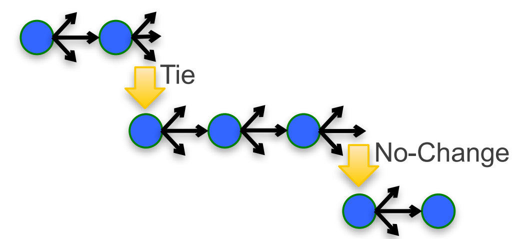
- **A (parallel) *reactive* layer**
  - Single graph/cognitive cycle

  Which acts as the inner loop for



- **A (serial/iterative) *deliberative* layer**
  - Repeated operator selection & application

  Which acts as the inner loop for



- **A (recursive) *reflective* layer**
  - Impasse-driven meta-level processing
- Maps onto bi-/tri-level models in
  - Cognitive Psychology (automatic vs. controlled, System 1 vs. 2, …)
  - Robotics (3T, …)
  - Emotion modeling

Reactive Layer

One Cognitive Cycle

Input	Memory Access, Perception & Reasoning	Decisions, Learning, Affect & Attention	Output

- Perceive into perceptual buffer (for perception predicates)
  - Ideally/ultimately just raw signal
- Process knowledge (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 (where appropriate)
- Latch WM distributions and selections (where appropriate)
- Learn for function parameters (when enabled)
- Update appraisals and their implications (when enabled)
- Execute output commands

Deliberative Layer

The Problem Space Computational Model

Select and apply operators to states

Blocks World
Problem Space

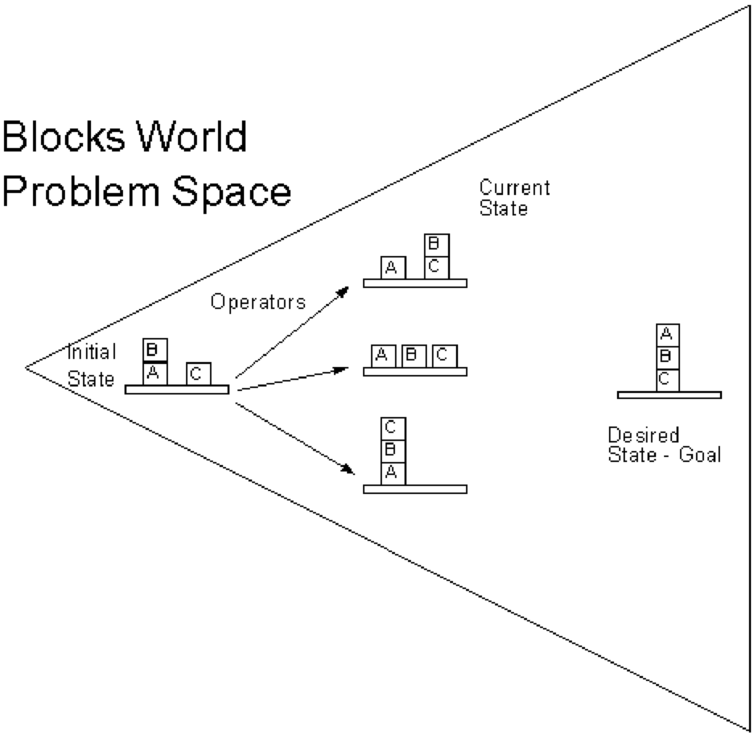


Figure 3.1: A problem space

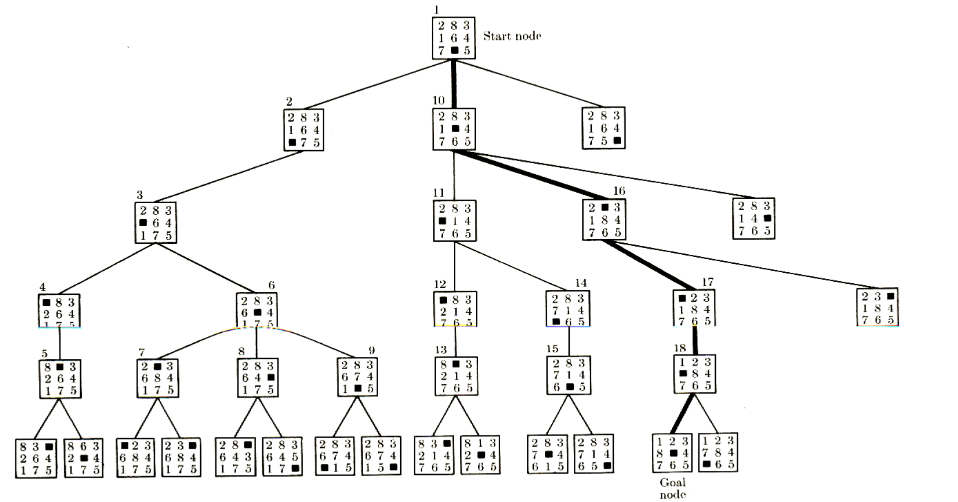


FIG. 3-5 *The tree produced by a depth-first search.*

Follows single path determined by knowledge

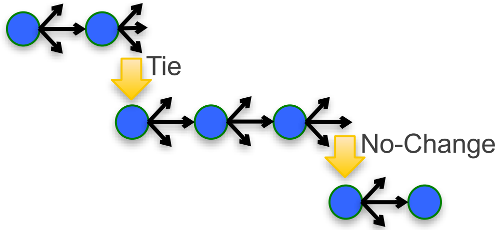
- Knowledge-intensive or algorithmic behavior
- Best, probability matching, Boltzmann, …

Doesn't actually do combinatoric search

- *Requires reflection*

Reflective Layer

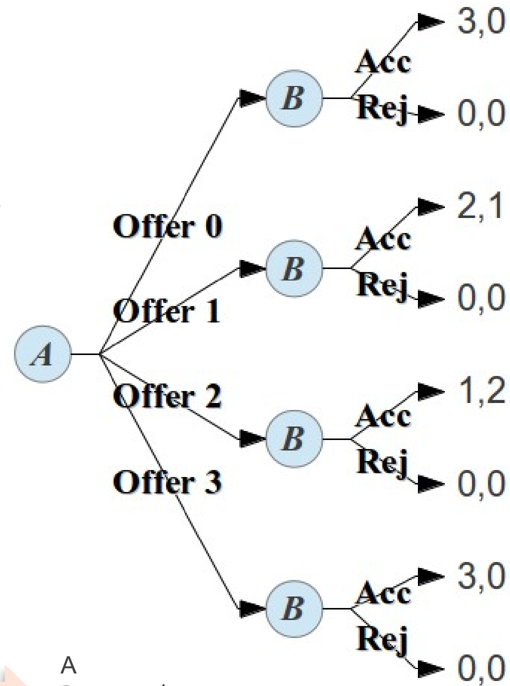
Impasses and Subgoaling (Meta-Levels)

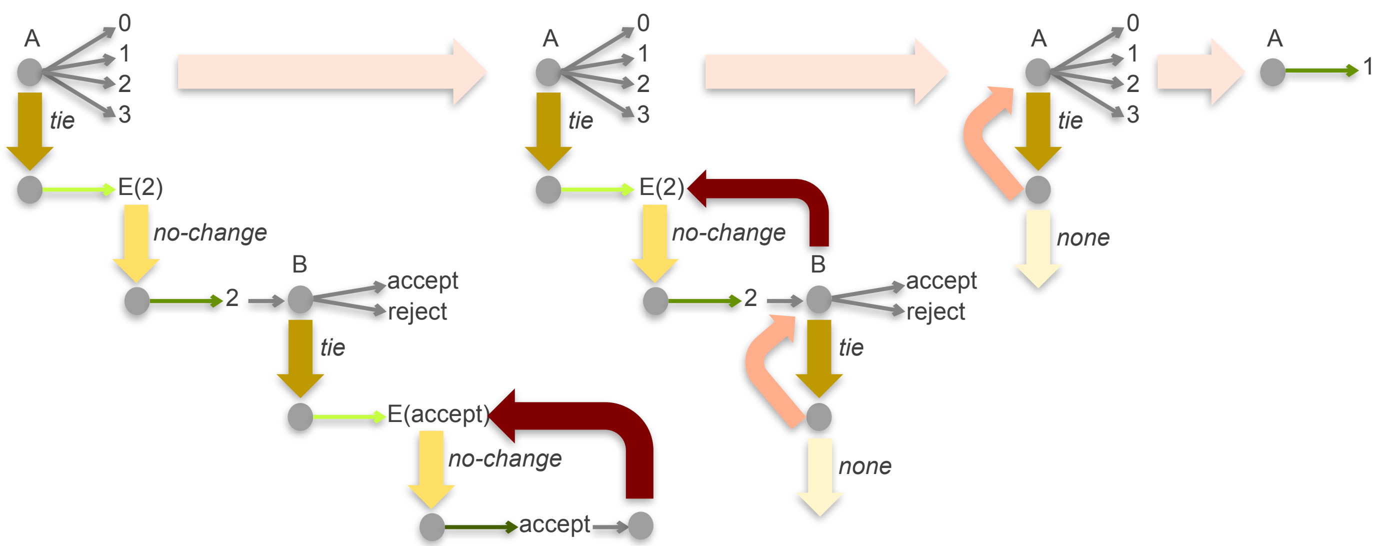


- Impasses occur for problems in operator selection
  - *None*: No operator acceptable (i.e., none with a positive rating)
  - *Tie*: More than one operator has the same best rating
    - And the rating is not 1 (*best*)
  - *No-change*: An operator remains selected for >1 decision
- Impasses yield subgoals (meta-levels, reflective-levels, …)
  - Confusingly, these levels are called states (modeled after Soar)
    - The state argument will see 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

Reflection in the Ultimatum Game

- *A* starts with a fixed amount of money (3)
- *A* decides how much (in 0-3) to offer *B*
- *B* decides whether or not to accept the offer
  - If accepts, each gets resulting amount; else both get 0
- Each has a utility function over money
  - E.g., <.1, .4, .7, 1>





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?

Winner of **Kurzweil Award** at AGI 2011 & 2012

Overall Progress on Sigma

- Memory
  - Procedural (rule) [ICCM 10]
  - Declarative (semantic/episodic) [ICCM 10, CogSci 14]
  - Constraint [ICCM 10]
  - Distributed vectors [AGI 14a]
  - Perceptual [BICA 14a, AGI 15]
  - Neural network [AGI 16]
- 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)
- Efficiency [ICCM 12, BICA 14b]
- Mental imagery [BICA 11a, AGI 12b]
  - 1-3D continuous imagery buffer
  - Object transformation
  - Feature & relationship detection
- Perception
  - Object recognition (CRFs) [BICA 11b]
  - Spoken word recognition (HMMs) [BICA 14a]
  - Localization [BICA 11b]
- Natural language
  - Word sense disambiguation [ICCM 13]
  - Part of speech tagging [ICCM 13]
  - Sentence identification [WS 15]
  - Dialogue [WS 15]
- Affect [AGI 15]
  - Appraisal (expectedness, desirability)
  - Attention (perceptual, cognitive)
- Integration
  - CRF+Localization+POMDP [BICA 11b]
  - Rules+SLAM+RL+ToM+VH [IVA 15, WS 15]
  - SentenceID+Dialogue [WS 15]

HANDS-ON SEGMENT

Online Tutorial

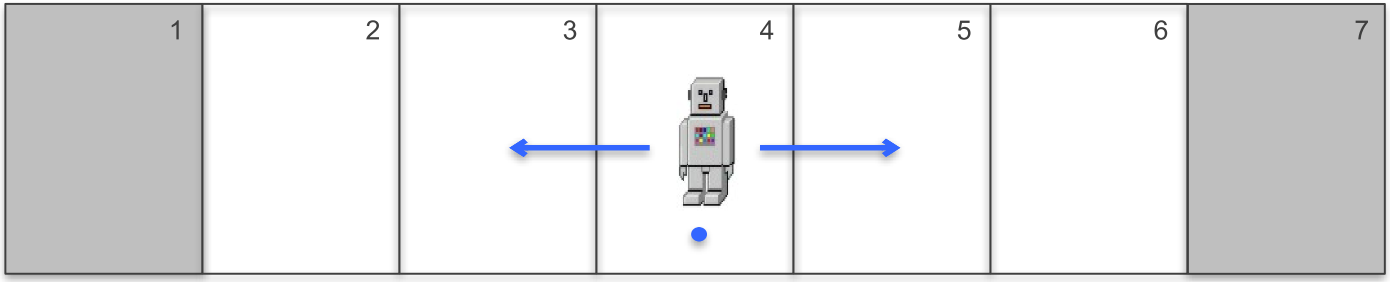
- https://bitbucket.org/sigma-development/tutorial/wiki/Home
  - The URL for the online tutorial
- Sigma source can be downloaded
  - https://bitbucket.org/sigma-development/tutorial/downloads/sigma38-tutorial.lisp
- Start up Lispworks, select 'open' & navigate to the location of sigma38-tutorial.lisp on your filesystem and double click to open.
- From the top menu select buffers -> compile
- All of the sigma functionality & the tutorial code are now loaded into your system

Sigma is Programmed in Common Lisp

- Core data structure is the *list* (of atoms, lists, numbers, etc.)
  - `(a b 5 d)`
  - `(5.22 (hello))`
- Functional programming
  - All activity involves function evaluation
- Function calls are evaluated lists (prefix notation)
  - `(+ (- 3 2) 5)`
  - `(random-walk-1)`
- Function definitions are evaluated lists
  - ```
    (defun factorial (n)
        (if (= n 1) 1 (* n (factorial (1- n)))))
    ```
- Evaluating a list yields a function call unless quoted: `‘(a b)`
- Evaluating an atom yields a binding unless quoted: `‘x`
    ```

A SEQUENCE OF SIGMA AGENTS

Random Walk on 1D Grid



- 1D Grid with eight cells 1-7
- Agent can move one cell to left or right, or stay where is

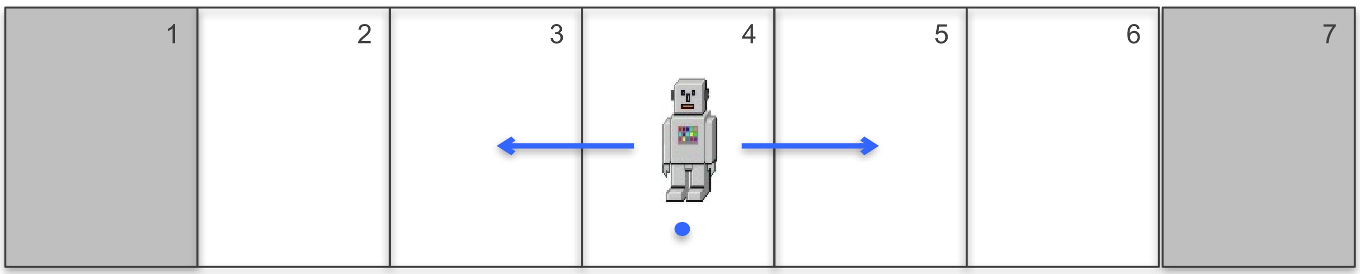
Pedagogical Sequence

1. Operators (+ conditionals)
2. Operator selection
3. Internal action execution (+ types & predicates)
4. Trials
5. External action execution (+ perception & action)
6. Value selection
7. External objects
8. Learning (of maps)
9. Simultaneous Localization and Mapping (SLAM)
10. Semantic memory (& learning)
11. SLAM + semantic memory
12. Action modeling (& templates)
13. Perception modeling
14. Reinforcement learning

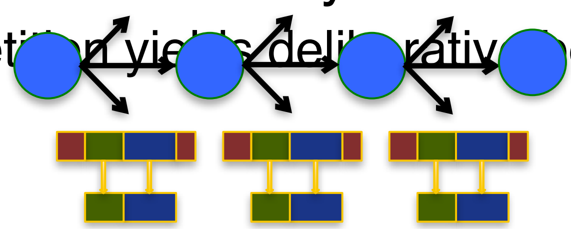
1. OPERATORS (+CONDITIONALS)

Operators

- Actions that can be performed internally or externally
  - In 1D random walk: *left*, *right*, *none*



- Selection followed by execution
  - Repetition yields deliberative behavior



random-walk-1 (Operators and Conditionals)

```
(defun random-walk-1()
  (init '(left right none))
  (conditional 'acceptable
     :actions '((selected (operator left))
                (selected (operator right))
                (selected (operator none))))
  (d 1))
```

- `init` initializes a Sigma model
  - Should be the first function called in a Sigma model
  - Here it also specifies the set of operators that may be considered
- `Selected` frames decision making process
  - Here all operators are specified with same default rating of 1
  - Default `best` decision rule selects randomly among highest rated
- `(d 1)`, or `(decide 1)`, runs one cognitive/decision cycle

Conditionals

- Structure *long-term memory* (LTM) and *basic reasoning*
  - Deep blending of traditional rules and probabilistic networks
- Comprise a *name*, one or more *patterns* and possibly a *function*
- Patterns may be *conditions* and *actions*, as in a rule

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

- Or even just actions that are always to be applicable

```
(conditional 'acceptable
   :actions '((selected (operator left))
              (selected (operator right))
              (selected (operator none))))
```

Always make all three RW operators available for selection

- Patterns may also be *condacts*
  - Support bidirectional reasoning, as needed with probabilities
- Patterns may include *constants* and *variables*

Results of (random-walk-1)

- Call `(pwmb 'selected)` to print operator that is selected

```
SELECTED
      WM-STATE x WM-OPERATOR:
                     [0:100>
  [LEFT]              0
 [RIGHT]              1
  [NONE]              0
```

- Prints the *working memory function* for `selected`
  - The `state` is the level of reflection at which this operator is selected
  - Since no level was specified in conditional, it is selected at all levels

2. OPERATOR SELECTION

random-walk-2 (Operator Selection)

```
(defun random-walk-2()
  (init '(left right none))
  (operator-selection 'boltzmann)
  (setq post-d '((pwmb 'selected)))

  (conditional 'acceptable
               :actions '((selected (operator *))))

  (d 1)
)
```

- Default selection is *best.*
- *`Boltzmann`* (or *`prob-match`*) selection allows a true random walk
- `*` denotes the entire domain of the variable.

3. INTERNAL ACTION EXECUTION (+TYPES & PREDICATES)

Types

- Types specify the domain of variables, including: their scope, whether they are numeric or symbolic, and whether they are discrete or continuous.
- Types may be *symbolic* or *numeric* (*discrete* or *continuous*)
  - `(new-type 'id :constants '(i1 i2 i3))`
  - `(new-type 'type :constants '(walker table dog human))` Symbolic
  - `(new-type 'color :constants '(silver brown white))`
  - `(new-type 'i04 :numeric t :discrete t :min 0 :max 5)` Discrete Numeric
    - Discrete [0, 5) => 0, 1, 2, 3, 4
  - `(new-type 'weight :numeric t :min 0 :max 500)` Continuous Numeric
    - Continuous [0, 500) => [0, 500-ε]

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 %)))`
- 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

random-walk-3 (Internal World)

- Mental simulation of the walk

```
(defun random-walk-3()
 …
  (new-type '1D-grid :numeric t :discrete t :min 1 :max 8)
  (predicate 'location :world 'closed :arguments '((x 1D-grid !)))

  …
  (conditional 'move-left
               :conditions '(
                             (selected (operator left))
                             (location (x (value))))
               :actions '((location (x (value -1)))))
  …
  (evidence '((location (x 4))))

  (d 5)
)
```

4. TRIALS

random-walk-4 (Trials)

- Run experiments

```
(defun random-walk-4()
 …
 (setq pre-t '((evidence '((location (x 4)) ))))
 …
 (conditional 'halt-at-location-1
              :conditions '(
                            (location (x 1))
                            )
              :actions '(
                         (halt)
                         )
              )
  …
 (trials 1)
)
```

5. EXTERNAL ACTION EXECUTION (+ PERCEPTION & ACTION)

random-walk-5 (External World)

- Define a world external to the model and make the Sigma model to interact with this world through perceptions and actions.
- perceive-location and execute-action are two functions defined as the interaction interface
- Perception predicates induce a segment of the *perceptual buffer*
  - Input is latched in perceptual buffer until changed
  - `(perceive '(0.8 (location (x 4))))`
- The location of the agent in this external world is captured by the Lisp variable 1d-grid-location and actions are executed by changing the value of this variable.

random-walk-5 (External World)

```
(defun random-walk-5()
 …
 (predicate 'location :perception t :arguments '((x 1D-grid %)))
 …
 (setq perceive-list
       `((perceive-location ,perception-prob ,perception-mass))
       )

 (setq action-list `((execute-operator ,action-prob)))
 …
)
```

6. VALUE SELECTION

Decisions

- Choice of *best* alternative at the cognitive level is computed as a side effect of MAX summarization over arriving messages
  - 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

random-walk-6 (Value Selection)

- The location-selected predicate is defined as closed-world with ! (select best) as the unique symbol

```
(defun random-walk-6()
…
  (predicate 'location              :perception t
                                    :arguments '((x 1D-grid %)))

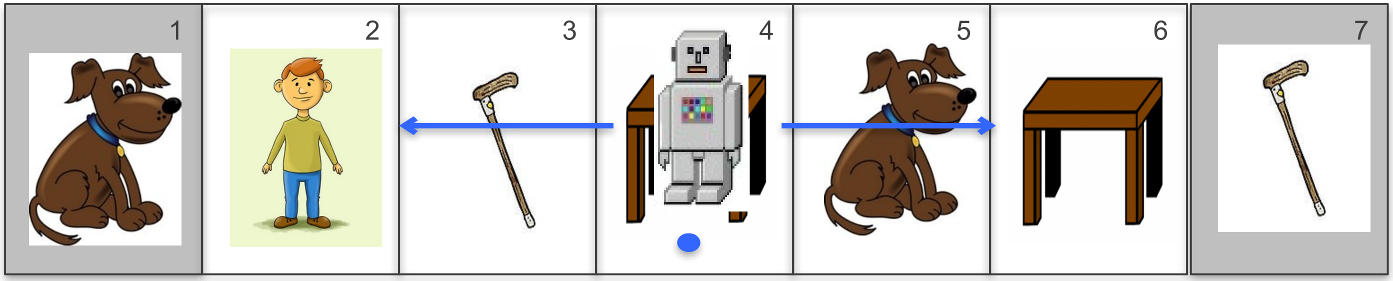
  (predicate 'location-selected            :world 'closed
                                           :arguments '((x 1D-grid !)))
…
  (conditional 'select-location

               :conditions '((location (x (location))))
               :actions '((location-selected (x (location)))))
…
)
```

7. EXTERNAL OBJECTS

random-walk-7 (External Objects)

- Assume there are objects at each grid location



random-walk-7 (External Objects)

```
(defun random-walk-7()
 …
 (new-type 'obj-type :constants '(walker table dog human))
 (predicate 'object :perception t :arguments '((object obj-type %)))
 (predicate 'object-perceived :world 'closed
                             :arguments ’(  (location 1D-grid)
                                            (object obj-type !)))
 …
 (conditional 'perceived-objects
              :conditions '(
                            (object (object (obj)))
                            (location (x (loc)))
                            )
              :actions '((object-perceived  (object (obj))
                                            (location (loc))))
              )
…
)
```

BREAK

SHORT SUMMARY (+FUNCTIONS)

Types

- Types specify the domain of variables, including: their scope, whether they are numeric or symbolic, and whether they are discrete or continuous.
- Types may be *symbolic* or *numeric* (*discrete* or *continuous*)
  - `(new-type 'id :constants '(i1 i2 i3))`
  - `(new-type 'type :constants '(walker table dog human))` Symbolic
  - `(new-type 'color :constants '(silver brown white))`
  - `(new-type 'i04 :numeric t :discrete t :min 0 :max 5)` Discrete Numeric
    - Discrete [0, 5) => 0, 1, 2, 3, 4
  - `(new-type 'weight :numeric t :min 0 :max 500)` Continuous Numeric
    - Continuous [0, 500) => [0, 500-ε]

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 %)))`
- 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*, one or more *patterns* and possibly a *function*
- Patterns may be *conditions* and *actions*, as in a rule

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

- Or even just actions that are always to be applicable

```
(conditional 'acceptable
   :actions '((selected (operator left))
              (selected (operator right))
              (selected (operator none))))
```

Always make all three RW operators available for selection

- Patterns may also be *condacts*
  - Support bidirectional reasoning, as needed with probabilities
- Patterns may include *constants* and *variables*

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)

8. LEARNING (OF MAPS)

Learning

- Learning occurs in Sigma via a process of gradient descent over functions defined in predicates or conditionals

- For instance, learning a map of objects in the single dimensional grid would require defining a function representative of the concept being learned

random-walk-8 (Map Learning)

```
(defun random-walk-8()
 …
  (learn '(:gd))
 …
  (predicate 'map      :arguments '(  (location 1D-grid)
                                      (object obj-type %))
                       :function 1)
 …
  (conditional 'perceived-objects
                       :conditions '(
                              (object (object (obj)))
                              (location (x (loc)))
                             )
                       :condacts '(
                              (map (object (obj)) (location (loc)))
                           )
  )
 …
)
```

9. SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)

random-walk-8 (Map Learning)

```
(defun random-walk-8()
 …
(learn '(:gd))
 …
(predicate 'map          :arguments '(  (location 1D-grid)
                                        (object obj-type %))
                         :function 1)
 …
(conditional 'perceived-objects
                :conditions '(
                               (object (object (obj)))
                               (location (x (loc)))
                               )
                :condacts '(
                             (map (object (obj)) (location (loc)))
                             )
                )
 …
)
```

random-walk-8 (Map Learning)

CORRECT LOCATION : 4
PERCEIVED LOCATION : 5

LOCATION POSTERIOR
(0.19999999: WM-X(1D-GRID)[4]) (0.6: WM-X(1D-GRID)[5]) (0.19999999: WM-X(1D-GRID)[6])

OBJECT PERCEIVED
(1: WM-OBJECT(OBJ-TYPE)[TABLE])

WM for LOCATION-SELECTED
WM-X:

[1]	[2]	[3]	[4]	**[5]**	[6]	[7]
0	0	0	0	**1**	0	0

MAP:
WM-LOCATION x WM-OBJECT:

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
[WALKER]	0.05167313	0.34530607	0.7832272	0.11975537	0.050506998	0.11340284	0.79965735
[TABLE]	1.55521E-4	1.55521E-4	0.21646221	0.87993443	0.2564895	0.8417874	0.20002768
[DOG]	0.40140295	0.19412153	1.552944E-4	1.5503877E-4	0.69284845	0.04465457	1.5746542E-4
[HUMAN]	0.5467684	0.46041688	1.552944E-4	1.5503877E-4	1.5503877E-4	1.5503877E-4	1.5746542E-4

random-walk-9 (SLAM)

```
(defun random-walk-9()
…
 (conditional 'perceived-objects
              :conditions '(
                             (object (object (obj)))

                             )
              :condacts '(
                           (location (x (loc)))
                           (map (object (obj)) (location (loc)))
                           )
              )
 …
)
```

random-walk-9 (SLAM)

CORRECT LOCATION : 5
PERCEIVED LOCATION : 4

LOCATION POSTERIOR
(1.0623143E-4: WM-X(1D-GRID)[3]) (0.2195618: WM-X(1D-GRID)[4]) (0.78033197: WM-X(1D-GRID)[5])

OBJECT PERCEIVED
(1: WM-OBJECT(OBJ-TYPE)[DOG])

WM for LOCATION-SELECTED
WM-X:

[1]	[2]	[3]	[4]	**[5]**	[6]	[7]
0	0	0	0	**1**	0	0

MAP:
WM-LOCATION x WM-OBJECT:

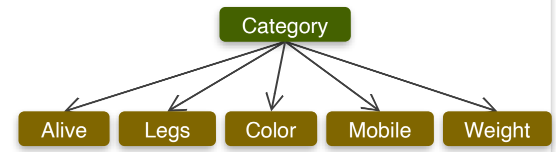
	[1]	[2]	[3]	[4]	[5]	[6]	[7]
[WALKER]	1.00908E-4	0.1106981	0.8140824	0.02834832	0.06153891	0.3477502	0.9556432
[TABLE]	1.00908E-4	0.039249763	0.114756346	0.72501874	0.13430768	0.6173503	0.044159383
[DOG]	0.9138201	0.34931234	0.071062826	0.2465345	0.804055	0.034801	9.87167E-5
[HUMAN]	0.085978076	0.5007398	9.852217E-5	9.852217E-5	9.852217E-5	9.8619E-5	9.8716686E-5

Relevant Publications

- Chen et al. (2011). Fusing symbolic and decision-theoretic problem solving + perception in a graphical cognitive architecture. *Proceedings of the Second International Conference on Biologically Inspired Cognitive Architectures*.
- Rosenbloom, P. S., Demski, A., Han, T. & Ustun, V. (2013). Learning via gradient descent in Sigma. *Proceedings of the 12th International Conference on Cognitive Modeling*.
- Ustun, V. & Rosenbloom, P. S. (2015). Towards adaptive, interactive virtual humans in Sigma. *Proceedings of the 15th International Conference on Intelligent Virtual Agents*.

10. SEMANTIC MEMORY (& LEARNING)

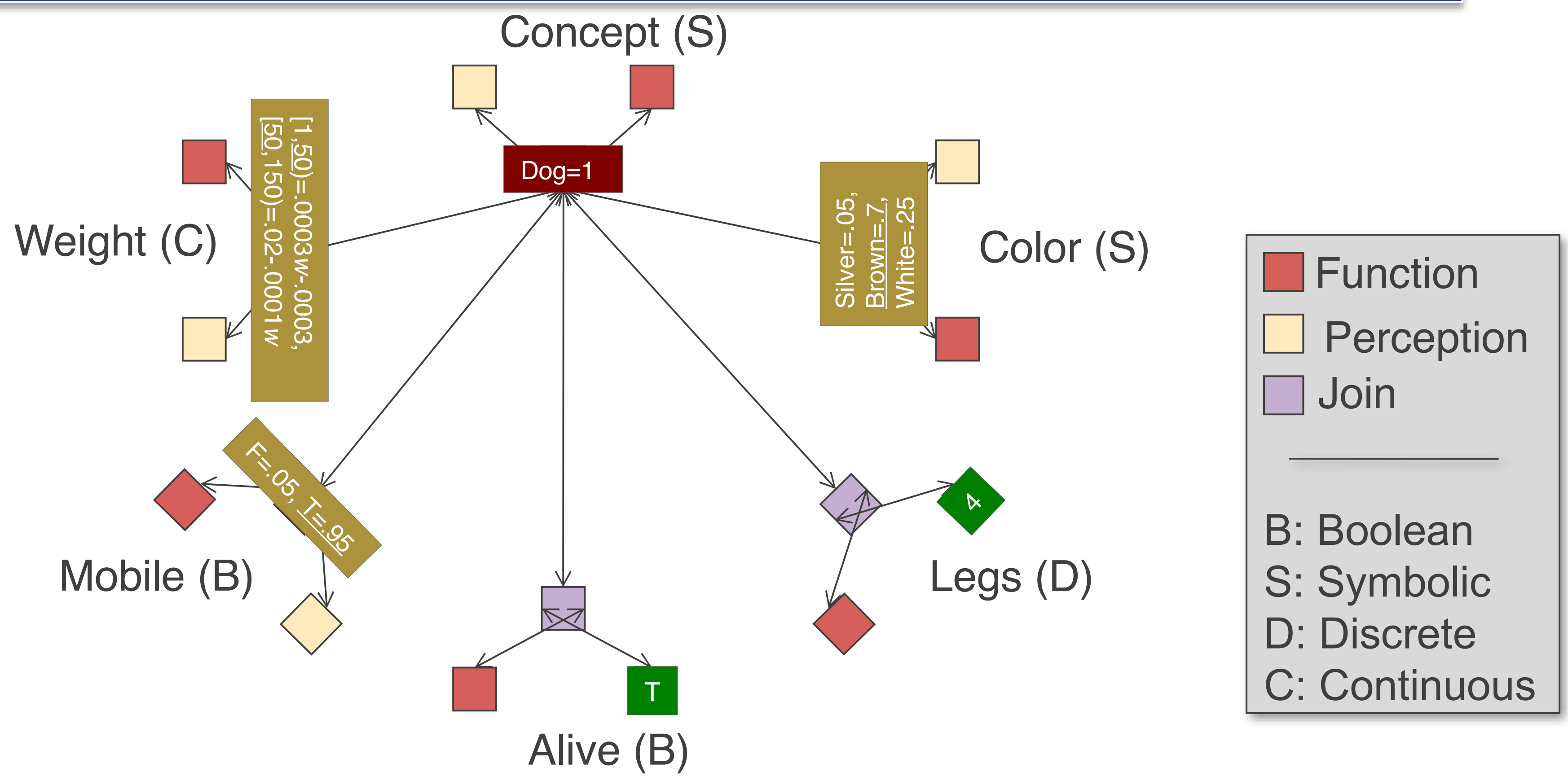
Semantic Memory (Classifier)



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



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

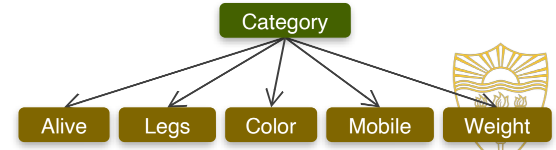
```
(conditional 'concept-color
             :conditions '((object (state (state)) (id (id))))
             :condacts '((concept (id (id)) (value (concept)))
                         (color (id (id)) (value (color))))
             :function-variable-names '(concept color)
             :function '((.95 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)))

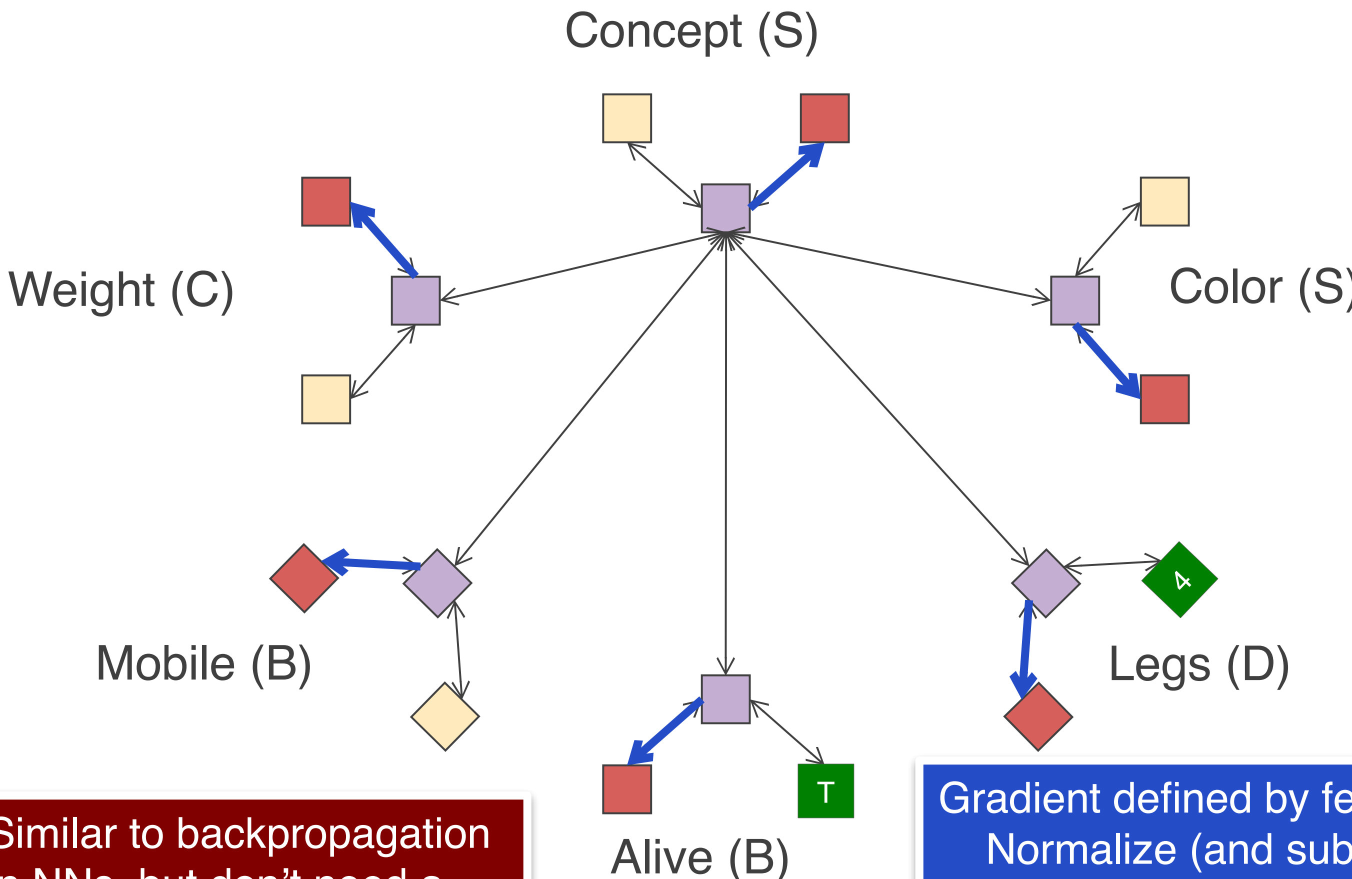
(conditional 'concept-color*join
    :conditions '((object (state (state)) (id (id))))
    :condacts '((concept (id (id)) (value (concept)))
                (color (id (id)) (value (color)))
                (concept-color (concept (concept)) (color (color))))
```

Pattern types and functions can be mixed arbitrarily in conditionals

Learning at Function Nodes

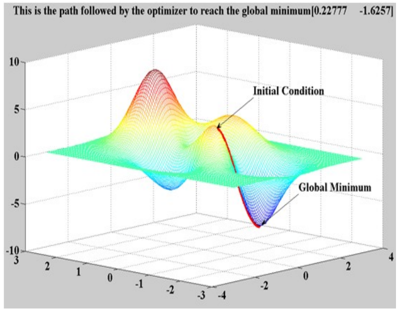


Naïve Bayes classifier



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 and cap

Add to function, smooth and normalize

Only function/parameter learning, not structure learning

random-walk-10 (Semantic Memory)

- Features used are `color`, `legs`, and `alive`

```
(defun random-walk-10()
 …
(new-type 'obj-type :constants '(walker table dog human))
(new-type 'color :constants '(silver brown white))
 …
(predicate 'object:perception t :arguments '((object obj-type %)))
(predicate 'color :perception t :arguments '((value color %)))
 …
(predicate 'object-prior
                      :arguments'(   (object obj-type %))
                      :function 1)
(predicate 'object-color
                      :arguments’(   (object obj-type)
                                     (color color %))
                      :function 1)
 …
)
```

random-walk-10 (Semantic Memory)

```
(defun random-walk-10()
 …
  (conditional 'perceived-objects
               :condacts '(
                    (object (object (obj)))
                    (object-prior (object (obj)))
                   ))

  (conditional 'object-color*join
               :condacts '(
                    (object (object (obj)))
                    (color (value (color)))
                    (object-color (object (obj)) (color (color)))
                    ))
…
)
```

11. SLAM + SEMANTIC MEMORY

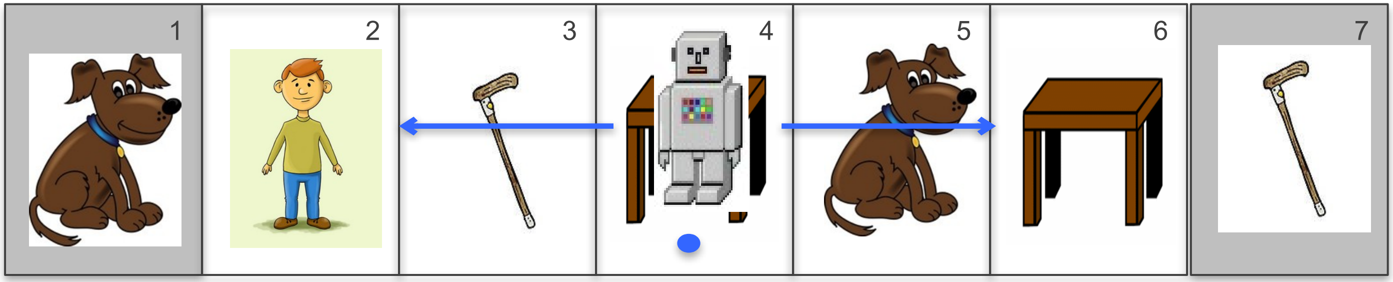
USC Institute for Creative Technologies

ARL

University of Southern California

random-walk-11 (SLAM + Semantic Memory)

- Observe features of the object, not the objects themselves



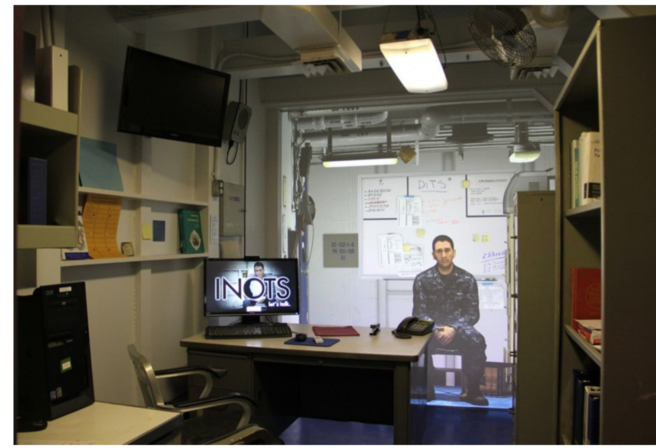
random-walk-11 (SLAM + Semantic Memory)

```
(defun random-walk-10()
 …
  (conditional 'perceived-objects
               :condacts '(
                     (object (object (obj)))
                     (object-prior (object (obj)))
                    ))
…)

(defun random-walk-11()
 …
   (conditional 'perceived-objects
               :condacts '(
                          (object (object (obj)))
                          (location (x (loc)))
                          (map (object (obj)) (location (loc)))
                          )
               )
…)
```

Integration: Replicating a Virtual Human "Mind"

- Immersive Naval Officer Training System (INOTS)
  - Targets leadership and basic counseling for junior Navy leaders
  - Trained over 5000 sailors since 2012



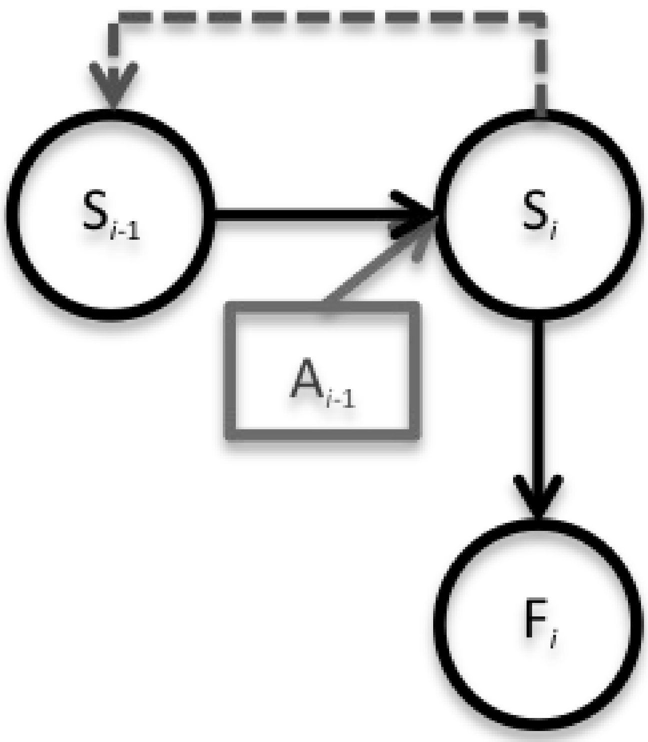
- INOTS "mind" based on two tools
  - Statistical query-answering tool (NPCEditor)
  - Transition diagram for dialogue management
- Both aspects reimplemented and integrated together in Sigma
  - Query answering via semantic memory (*reactive*)
  - Dialogue management by sequences of operators (*deliberative*)

12. ACTION MODELING (& TEMPLATES)

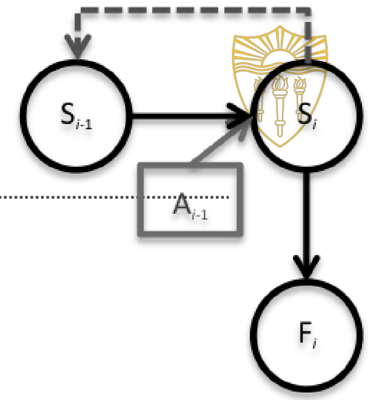
Template-Based Structure Creation

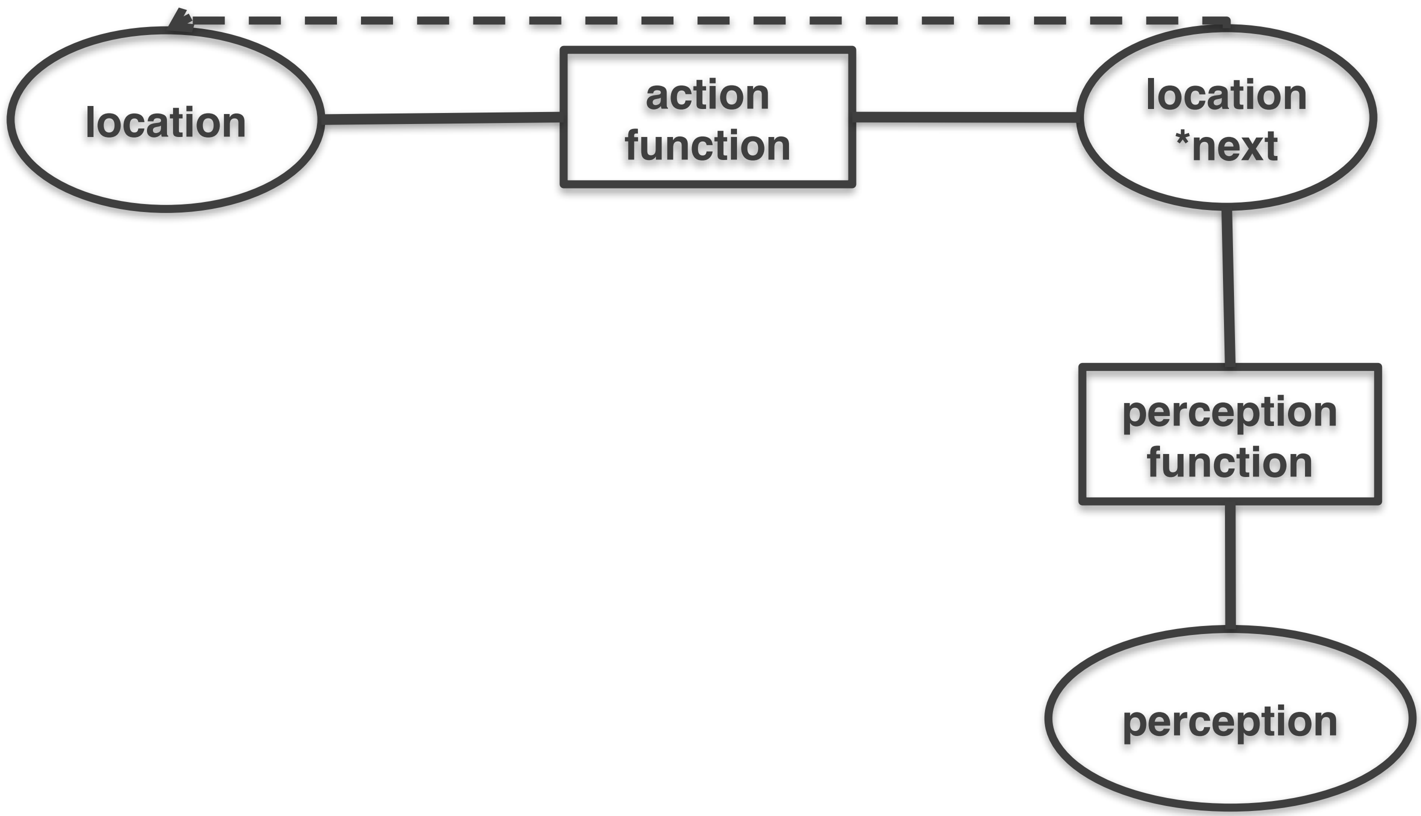
- From specifications of core state predicates automatically generate additional types, predicates and conditionals as needed for various forms of learning
- *Synchronic prediction*
  - Map learning in SLAM
  - Acoustic function learning in speech HMM
- *Diachronic prediction*
  - Learning action models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

Generic single slice trellis with optional action



Agent model


S_{i-1}
S_i
A_{i-1}
F_i


location
action function
location *next
perception function
perception

random-walk-12 (Action Modeling)

```
(defun random-walk-12()
 …
  (learn '(:am))
 …
  (predicate 'location          :world 'closed
                                :perception t
                                :arguments ’(  (state state)
                                               (x 1D-grid !)))
…
)
```

Automatically Generated Predicates and Conditionals

```
(PREDICATE 'LOCATION*NEXT :WORLD 'OPEN :UNIQUE '(X) :PERCEPTION T
                          :ARGUMENTS '((STATE STATE) (X 1D-GRID %)))

(PREDICATE 'ACTION-2043   :WORLD 'OPEN :UNIQUE '(X-2)
                          :ARGUMENTS '(     (X-0 1D-GRID) (OPERATOR-1 OPERATOR)
                                            (X-2 1D-GRID))
                          :FUNCTION 1)

(CONDITIONAL 'LOCATION-PREDICTION
         :CONDITIONS '(   (STATE (STATE (S)))
                          (LOCATION (STATE (S)) (X (X-0)))
                          (SELECTED (STATE (S)) (OPERATOR (OPERATOR-1))))

         :CONDACTS '(     (LOCATION*NEXT    (STATE (S)) (X (X-2)))
                          (ACTION-2101      (X-0 (X-0)) (OPERATOR-1 (OPERATOR-1))
                                            (X-2 (X-2))))
    )
```

13. PERCEPTION MODELING

random-walk-13 (Action & Perception Modeling)

```
(defun random-walk-13()
 …
  (learn '(:pm :am))
 …
  (predicate 'location          :world 'closed :perception t
                                :arguments '(  (state state)
                                               (x 1D-grid !)))

  (predicate 'object            :perception t
                                :arguments '(  (state state)
                                (object obj-type %)))
…
)
```

Automatically Generated Predicates

```
(PREDICATE 'LOCATION*NEXT            :WORLD 'OPEN :UNIQUE '(X)
                                     :PERCEPTION T
                                     :ARGUMENTS '((STATE STATE) (X 1D-GRID %)))

(PREDICATE 'ACTION-2103              :WORLD 'OPEN :UNIQUE '(X-2)
                                     :ARGUMENTS '(    (X-0 1D-GRID)
                                                      (OPERATOR-1 OPERATOR)
                                                      (X-2 1D-GRID))
                                     :FUNCTION 1)

(PREDICATE 'PERCEPTION-2104          :WORLD 'OPEN :UNIQUE '(OBJECT-1)
                                     :ARGUMENTS '((OBJECT-1 OBJ-TYPE) (X-0 1D-
GRID))
                                     :FUNCTION 1)
```

Automatically Generated Conditionals

```
(CONDITIONAL 'LOCATION-PREDICTION
        :CONDITIONS '(   (STATE (STATE (S)))
                         (LOCATION (STATE (S)) (X (X-0)))
                         (SELECTED (STATE (S)) (OPERATOR (OPERATOR-1))))

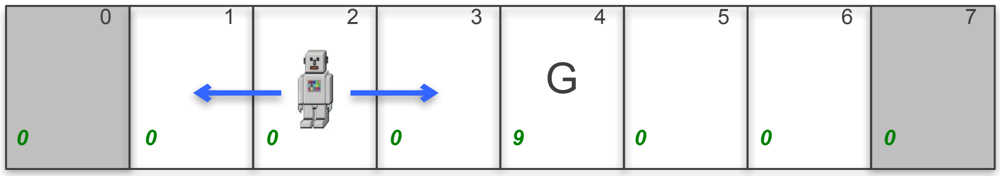
        :CONDACTS '(     (LOCATION*NEXT (STATE (S)) (X (X-2)))
                         (ACTION-2103 (X-0 (X-0))
                         (OPERATOR-1 (OPERATOR-1)) (X-2 (X-2))))
    )


(CONDITIONAL 'OBJECT-PERCEPTION-PREDICTION
        :CONDITIONS '(   (STATE (STATE (S))))

        :CONDACTS '(     (OBJECT (STATE (S)) (OBJECT (OBJECT-1)))
                         (LOCATION*NEXT (STATE (S)) (X (X-0)))
                         (PERCEPTION-2104 (OBJECT-1 (OBJECT-1)) (X-0 (X-0))))
    )
```

14. REINFORCEMENT LEARNING

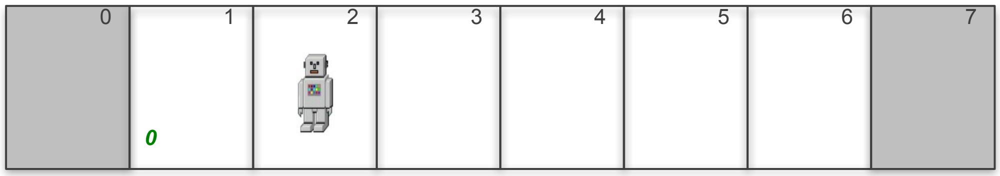
Example: Reinforcement Learning



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

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

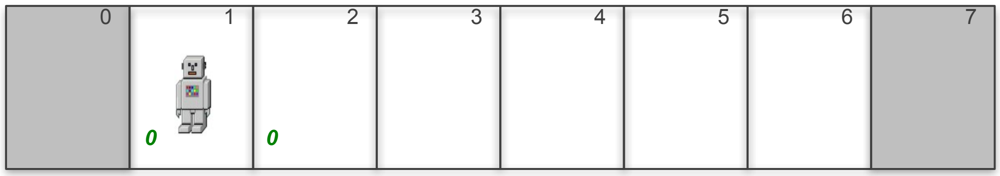
Example: Reinforcement Learning



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

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

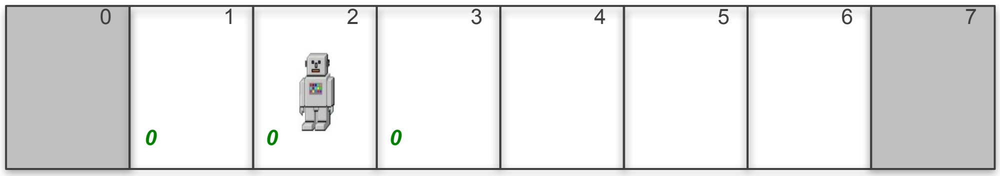
Example: Reinforcement Learning



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

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

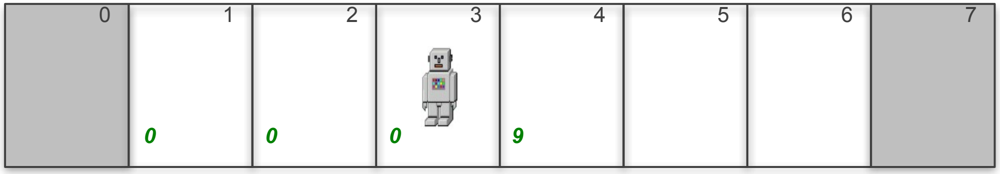
Example: Reinforcement Learning



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

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

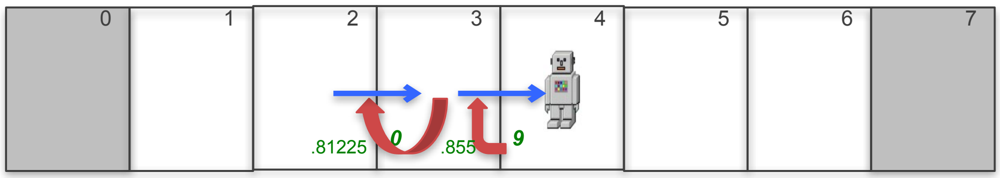
Example: Reinforcement Learning



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

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

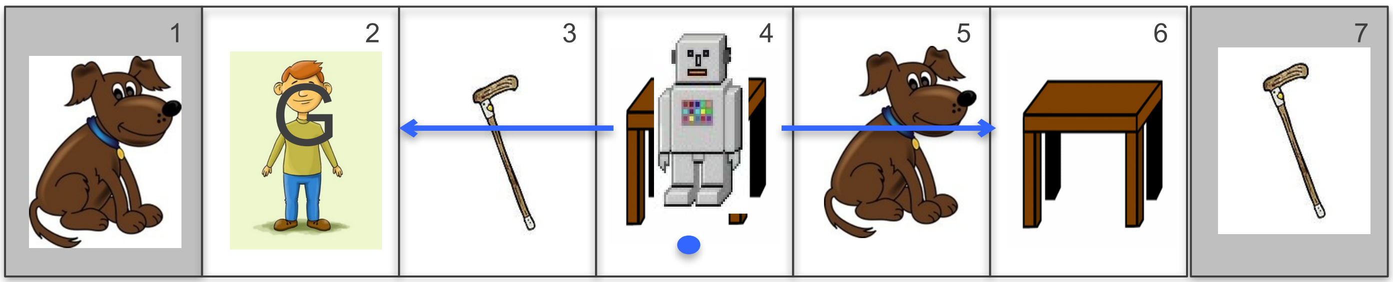
Example: Reinforcement Learning



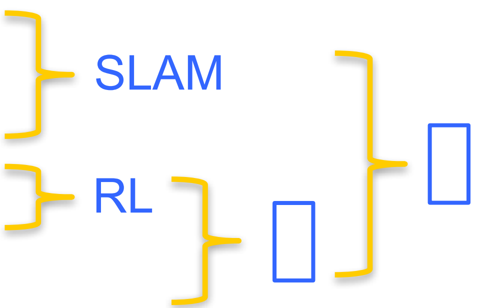
Learn values of actions for states by backwards propagation of rewards received during exploration:

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

Integration: Simulated Robot in 1D Corridor



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



random-walk-14 (Reinforcement Learning)

```
(defun random-walk-14()
…
  (learn '(:pm :am :rl))
…
  (predicate 'location :world 'closed :perception t
                       :arguments '((state state) (x 1D-grid !)))

  (predicate 'object   :perception t
                       :arguments '((state state) (object obj-type %)))
…
)
```

Automatically Generated Predicates (Action & Perception)

```
(PREDICATE 'LOCATION*NEXT           :WORLD 'OPEN :UNIQUE '(X)
                                    :PERCEPTION T
                                    :ARGUMENTS '((STATE STATE) (X 1D-GRID %)))


(PREDICATE 'ACTION-2103             :WORLD 'OPEN :UNIQUE '(X-2)
                                    :ARGUMENTS '(      (X-0 1D-GRID)
                                                       (OPERATOR-1 OPERATOR)
                                                       (X-2 1D-GRID))
                                    :FUNCTION 1)

(PREDICATE 'PERCEPTION-2104         :WORLD 'OPEN :UNIQUE '(OBJECT-1)
                                    :ARGUMENTS '((OBJECT-1 OBJ-TYPE) (X-0 1D-
GRID))
                                    :FUNCTION 1)
```

Automatically Generated Predicates (RL)

```
(PREDICATE 'PROJECTED      :WORLD 'OPEN :UNIQUE '(VALUE)
                           :ARGUMENTS '((LOCATION-X 1D-GRID) (VALUE UTILITY %))
                           :FUNCTION 1)

(PREDICATE'PROJECTED*NEXT  :WORLD 'OPEN :UNIQUE '(VALUE)
                           :ARGUMENTS '((LOCATION-X  1D-GRID) (VALUE UTILITY %))
                           :FUNCTION 'PROJECTED)

(PREDICATE 'REWARD         :WORLD 'OPEN :UNIQUE '(VALUE) :PERCEPTION T
                           :ARGUMENTS '((LOCATION-X 1D-GRID) (VALUE UTILITY %))
                           :FUNCTION '((0 * (0 20)) (0.1 * (0 10))))

(PREDICATE 'Q              :WORLD 'OPEN :UNIQUE '(VALUE)
                           :ARGUMENTS '((LOCATION-X 1D-GRID) (OPERATOR OPERATOR)
                                         (VALUE UTILITY %))
                           :FUNCTION 1)
```

Relevant Publications

- Rosenbloom, P. S. (2012). Deconstructing reinforcement learning in Sigma. *Proceedings of the 5th Conference on Artificial General Intelligence*.
- Pynadath, D. V., Rosenbloom, P. S. & Marsella, S. C. (2014). Reinforcement learning for adaptive Theory of Mind in the Sigma cognitive architecture. *Proceedings of the 7th Annual Conference on Artificial General Intelligence*.
- Ustun, V. & Rosenbloom, P. S. (2015). Towards adaptive, interactive virtual humans in Sigma. *Proceedings of the 15th International Conference on Intelligent Virtual Agents*.

ADDITIONAL TOPICS

Additional Topics

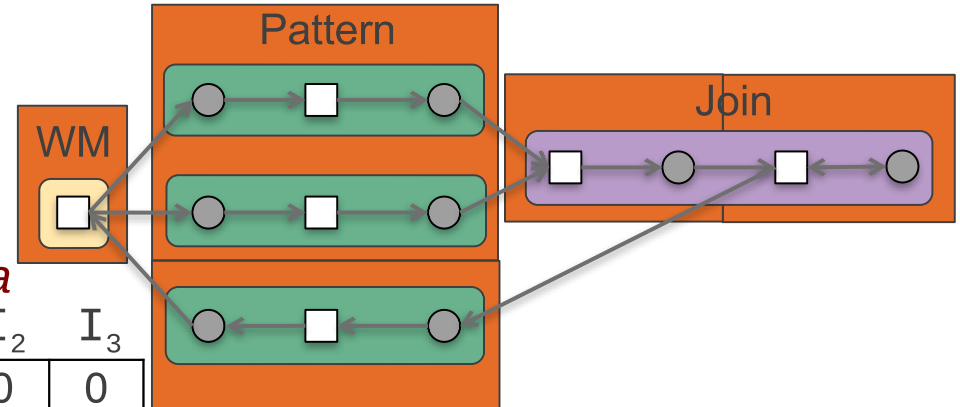
- Rule memory (& mapping to graphical models)
- Mental imagery
- Distributed vectors (word embeddings)
- Episodic memory
- Appraisal & attention
- Theory of Mind (& multiagent systems)
- Interactive adaptive virtual humans

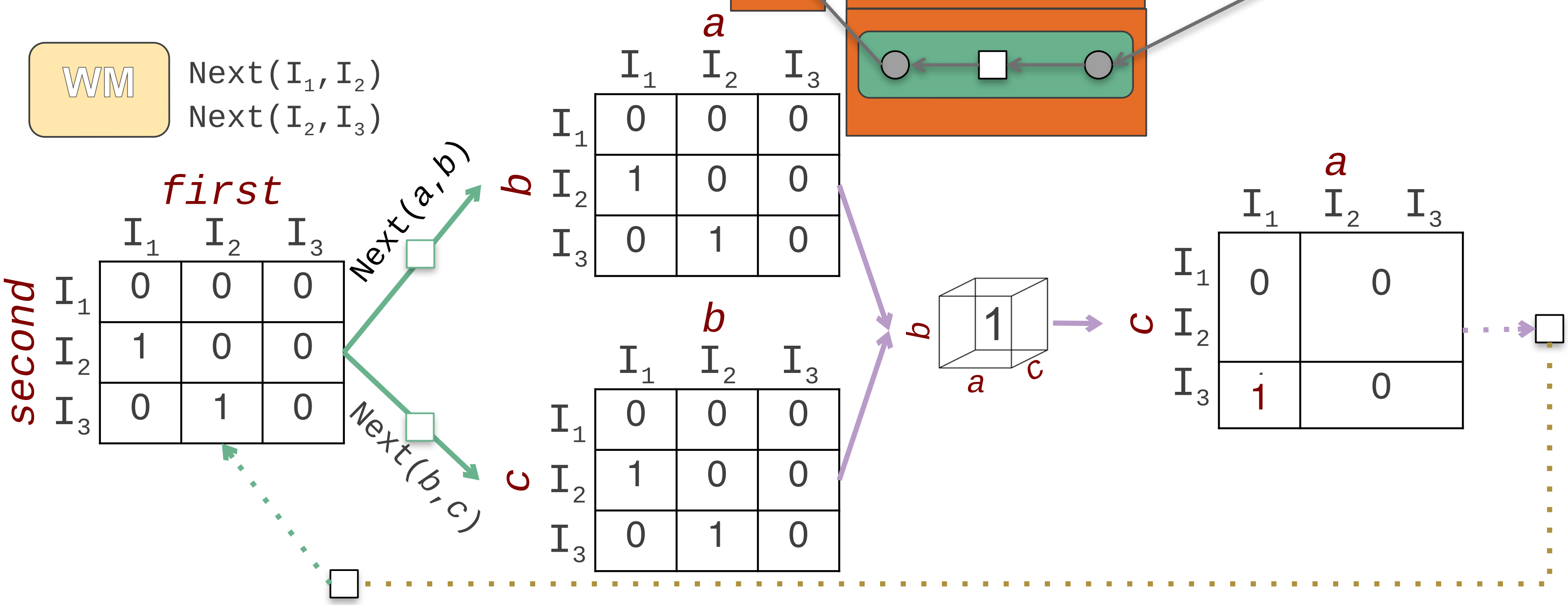
RULE MEMORY (& MAPPING TO GRAPHICAL MODELS)

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

Procedural Memory (Rules)

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



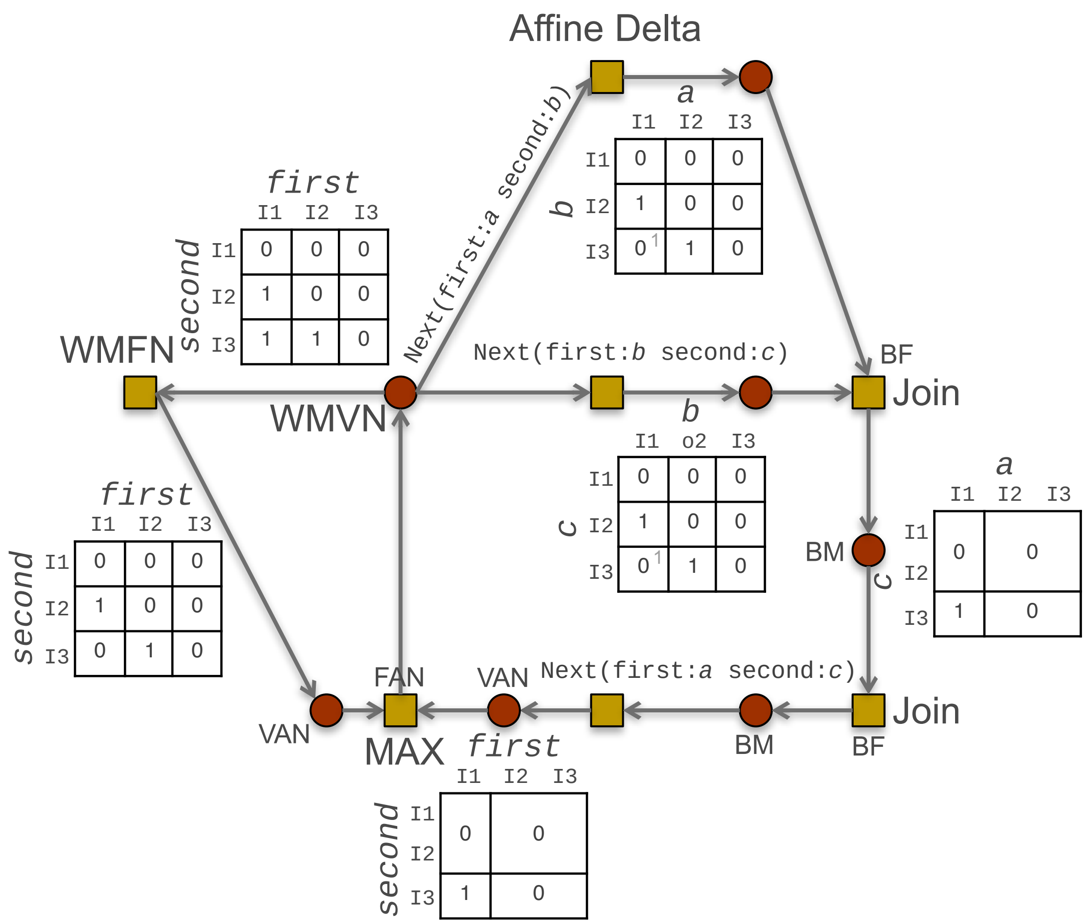



```
(type 'ID :constants '(I1 I2 I3))
(predicate 'Next '((first ID) (second ID)) :world 'closed)
```

Procedural Memory (Rules)

In More Detail

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

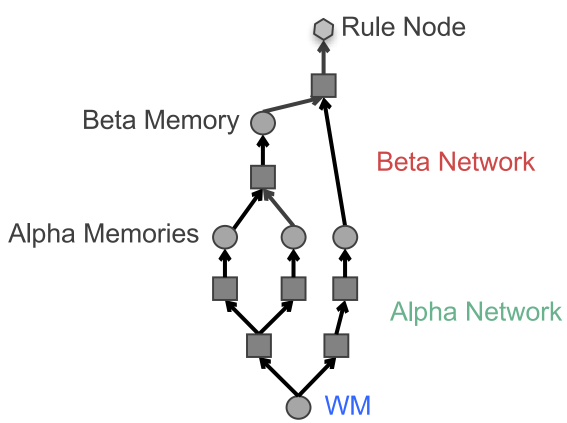


Examining Graphs via (g)

- (test-rule-one)

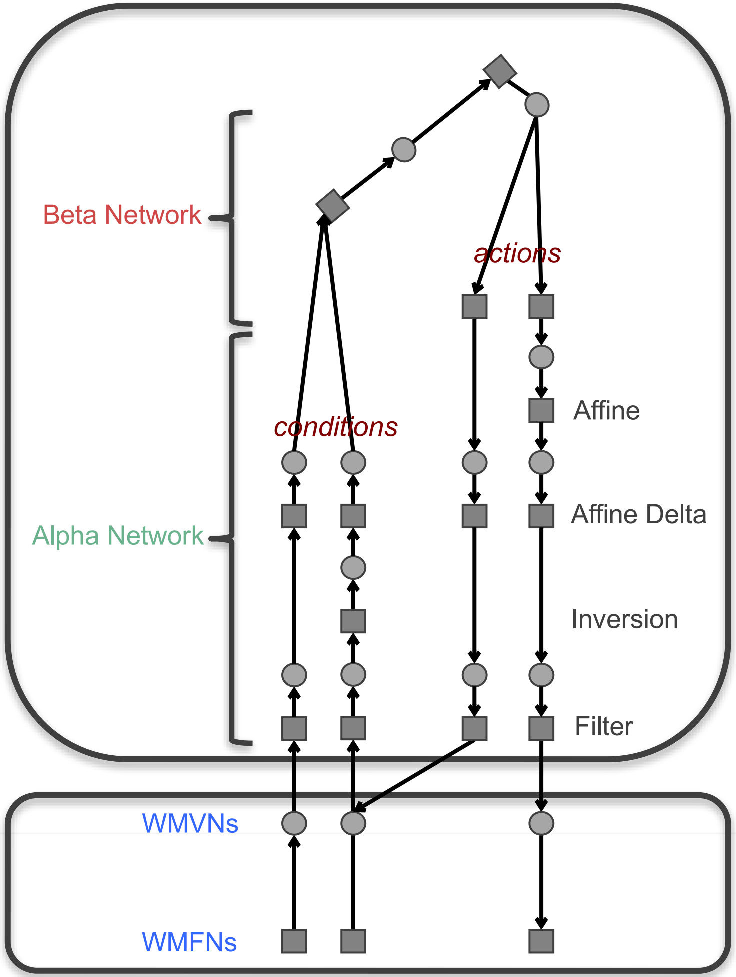
Compiler (Rules)

- Predicates and conditionals compile into portions of factor graph



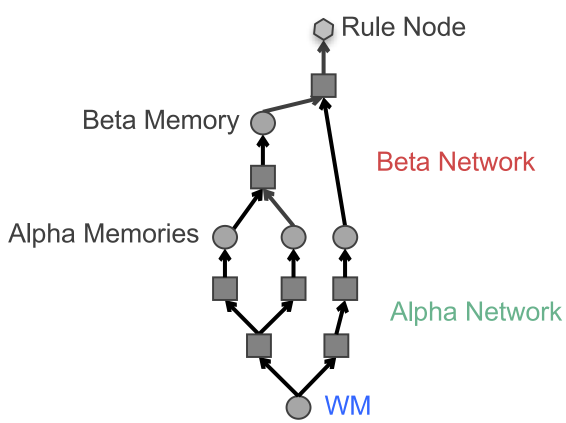
Rete for rule match

C1 & C2 & C3 ➔ A1 & A2



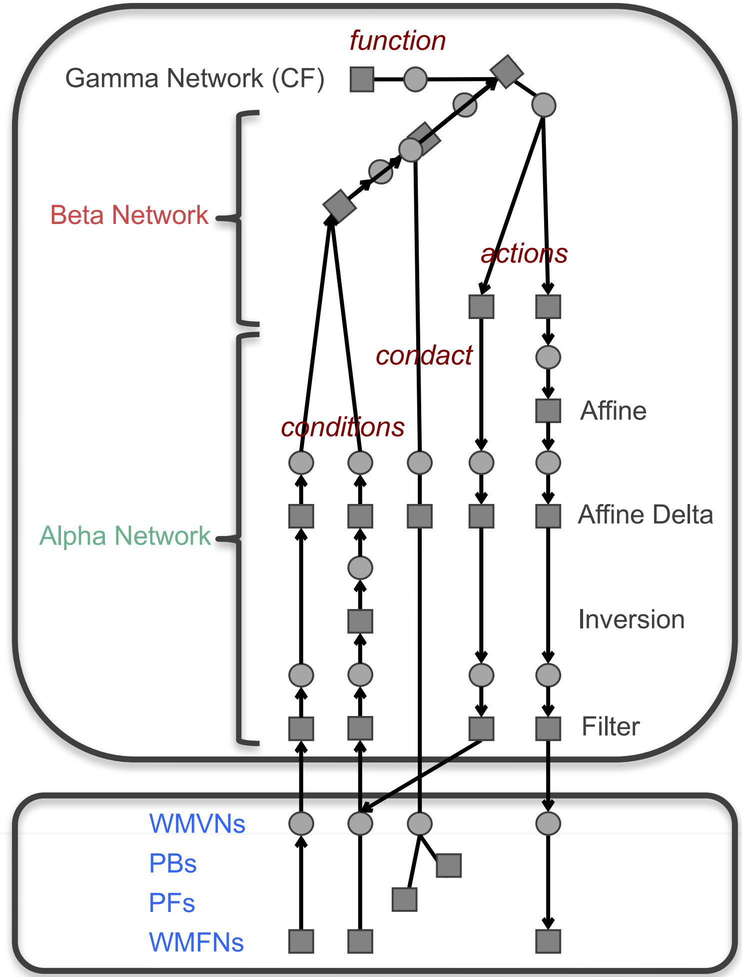
Compiler (Condacts and Functions)

- Predicates and conditionals compile into portions of factor graph



Rete for rule match

C1 & C2 & C3 ➔ A1 & A2

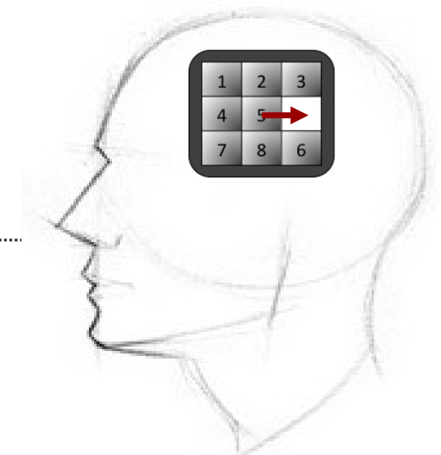


Relevant Publications

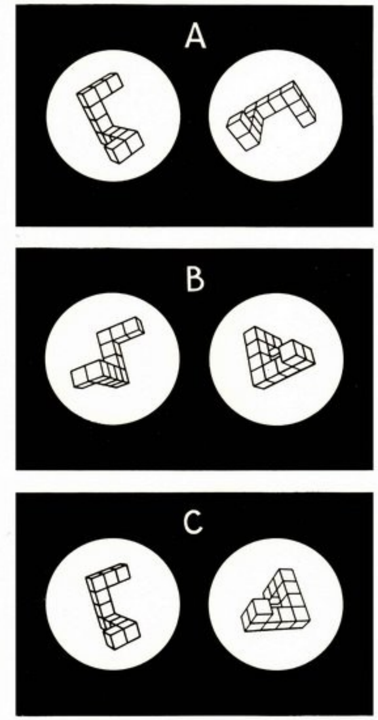
- Rosenbloom, P. S. (2010). Combining procedural and declarative knowledge in a graphical architecture. *Proceedings of the 10th International Conference on Cognitive Modeling*.

MENTAL IMAGERY

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



Affine Transforms

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

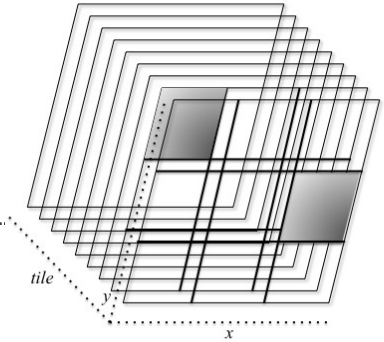
- Translation: Addition (offset)
  - Negative (e.g., y + -3.1 or y – 3.1): Shift to the left
  - Positive (e.g., y + 1.5): Shift to the right

- 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*

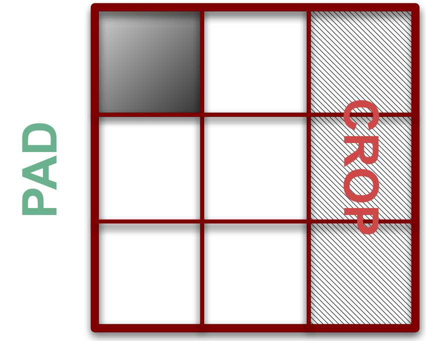
Yields a form of primitive mental arithmetic

How to Slide a Tile

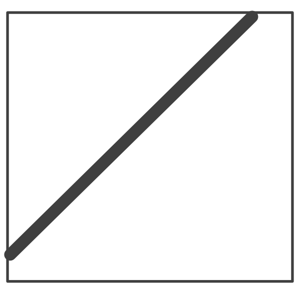


- Offset boundaries of regions along a dimension

```
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)
```

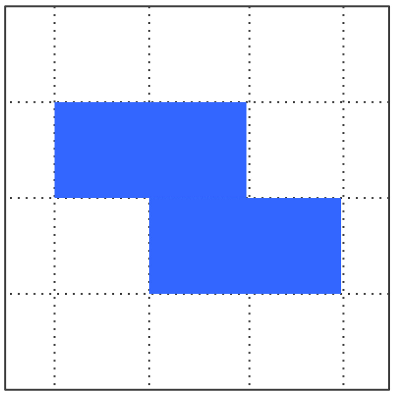


- Special purpose optimization of a *delta function*

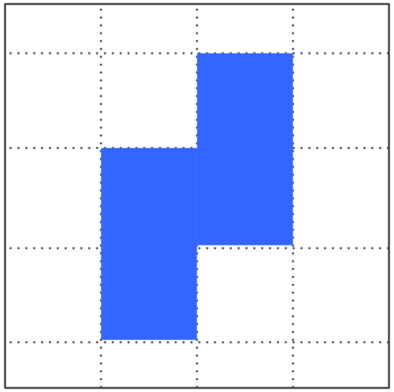


2	0	1	0
1	1	0	0
0	0	0	0
y/x	*0*	*1*	*2*

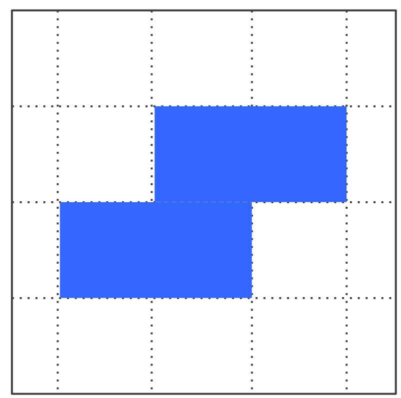
Transform a *Z Tetromino* (via Affine Nodes)



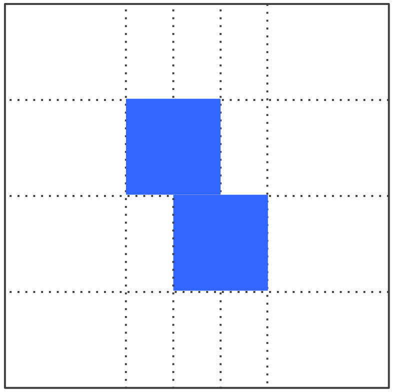
CONDITIONAL *Rotate-90-Right*
  Conditions: (tetromino x:*x* y:*y*)
  Actions: (tetromino x:*4-y* y:*x*)



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



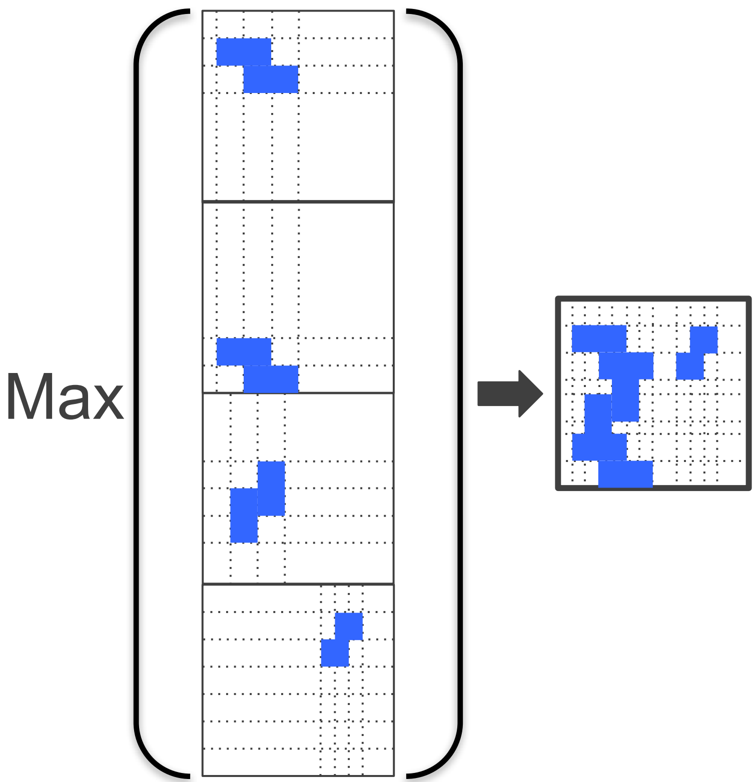
CONDITIONAL *Scale-Half-Horizontal*
  Conditions: (tetromino x:*x* y:*y*)
  Actions: (tetromino x:*x*/2+1 y:*y*)



Composition and Extraction

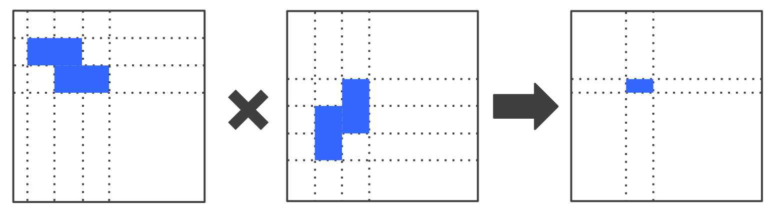
Object Composition

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



Overlap Detection

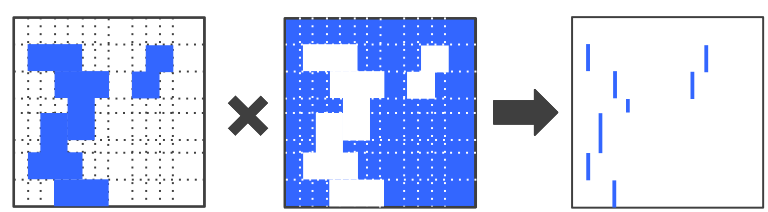
```
CONDITIONAL Ovelap-0-1
   Conditions: (Image object:0 x:x y:y)
               (image object:1 x:x y:y)
   Actions: (Overlap overlap:0 x:x y:y)
```



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)
```



Relevant Publications

- Rosenbloom, P. S. (2011). Mental imagery in a graphical cognitive architecture. *Proceedings of the Second International Conference on Biologically Inspired Cognitive Architectures*.
- Rosenbloom, P. S. (2012). Extending mental imagery in Sigma. *Proceedings of the 5th Conference on Artificial General Intelligence*.

DISTRIBUTED VECTORS (WORD EMBEDDINGS)

Distributed Vector Representation or Word Embedding

- Simple yet general approach to integrating large amounts of diverse knowledge while yielding natural measures of similarity
- Assign long (e.g., 1000) random vectors to words & concepts

0.60665036	- 0.5666231	0.41830373	- 0.5400135	0.61649907	0.02903163	0.16481042	. . .

- Evolve "better" vectors from experience with usage
  - Co-occurring words, n-grams, phonetic structure, visual features, …
- Degree of similarity is a function of distance in vector space
  - For richer language models, simple forms of analogy, …

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

- Long history in cognitive science (particularly neural networks)
  - More recently an important thread in machine learning
  - Started to appear in a few cognitive architectures

Our Hypothesis

Sigma can efficiently and effectively support a *distributed vector representation* that enables implicit learning of the meanings of words and concepts from large but shallow information resources

Distributed Vector Representations in Sigma (DVRS)

Ordering

Context

The AGI conferences encourage interdisciplinary research based on different understandings of intelligence, and exploring different approaches.

Ordering Vector

Context Vector

Lexical Vector

$$o(k) = \sum^{4} s(i) * e(k+i)$$

$$l(k) = l(k) + \widehat{c(k)} + \widehat{o(k)}$$

where $j \neq 0$ and $0 < (k+j) \leq n$

DVR in Sigma

- Vectors are discrete piecewise-constant functions

0.60665036	-0.5666231	-0.4183037	0.54001356	-0.6164990	0.02903163	0.16481042

- Sum-product algorithm manipulates (× & +) vectors
- Gradient-descent evolves lexical representations

Context Vector

w
1
0
1

✖

w \ d				
0.66	0.14	0.92	0.17	0.14
0.43	0.1	0.17	0.53	0.53
0.01	0.71	0.77	0.08	0.53
0.51	0.54	0.70	0.81	0.94

=

w \ d				
0.66	0.14	0.92	0.17	0.14
		0		
0.51	0.54	0.70	0.81	0.94

↓ **Summarization**

d				
1.17	0.68	1.62	0.98	1.08

↓ **L2 Normalization**

d				
0.46	0.27	0.63	0.38	0.42

CONDITIONAL *Co-occurence*

Conditions: Co-occuring-Words(word:*w*)

Actions: Context-Vector(distributed:*d*)

Function(*w*,*d*): *environmental-vectors*

$$c(k) = \sum_{i=1}^{n} e(i), where\ i \neq k$$

Conditionals for Ordering Information

```
CONDITIONAL Skip-gram
   Conditions: Skip-Gram-Words(word:w position:p)
               Environmental-Vectors(word:w distributed:d)
   Actions: Skip-Gram-Matrix(distributed:d position:p)
```

```
CONDITIONAL Ordering-vector
   Conditions: Skip-Gram-Matrix(distributed:d position:p)
   Actions: Ordering-Vector(distributed:d)
   Function(p,d): *sequence-vectors*
```

Ordering Vector

$$o(k) = \sum_{j=-4}^{4} s(j) \mathbin{.*} e(k+j)$$

$$where\ j \neq 0\ and\ 0 < (k+j) \leq n$$

Conditionals for Meaning/Lexical Vector

```
CONDITIONAL Context
   Conditions: Context-Vector(distributed:d)
               Current(word:w)
   Actions: Meaning-Vector(word:w distributed:d)

CONDITIONAL Ordering
   Conditions: Ordering-Vector(distributed:d)
               Current(word:w)
   Actions: Meaning-Vector(word:w distributed:d)
```

Lexical Vector

$$\underbrace{l(k)_t = l(k)_{t-1}}_{\text{Gradient Descent}} + \underbrace{\widehat{c(k)} + \widehat{o(k)}}_{\text{Action Combination}}$$

Sample Results

Context	Ordering	Composite
spoken	cycle	languages
languages	society	vocabulary
speakers	islands	dialect
linguistic	industry	dialects
speak	era	syntax

language

film

Context	Ordering	Composite
director	movie	movie
directed	german	documentary
starring	standard	studio
films	game	films
movie	french	movies

External
Simulator

Assessment of DVRS

- Word2Vec's Semantic-Syntactic Word Relationship Test Set*
  - "What is the word that is similar to *small* in the same sense as *biggest* is similar to *big*?"
    - $V = (I_{biggest} - I_{big}) + I_{small}$
  - or "Which word is the most similar to *Paris* in the way *Germany* is similar to *Berlin*?"
    - $V = (I_{germany} - I_{berlin}) + I_{paris}$

* https://code.google.com/p/word2vec/

Accuracy on Semantic-Syntactic Word Relationship Test Set

	Vector Size	Semantic	Syntactic	Overall
Co-occurrence only	1024	33.7 (31.1)	18.8 (18.6)	25.3 (24.3)
3-Skip-Bigram only	1024	2.7 (2.5)	5.0 (4.9)	4.0 (3.8)
3-Skip-bigram composite	512	29.8 (27.5)	18.5 (18.3)	23.4 (22.4)
3-Skip-bigram composite	1024	32.7 (30.2)	19.2 (18.9)	25.1 (24.0)
3-Skip-bigram composite	1536	34.6 (31.9)	20.1 (19.9)	26.4 (25.3)
3-Skip-bigram composite	2048	34.3 (31.7)	20.1 (19.9)	26.3 (25.2)

Word2Vec
19.3%

Training data is enwik8 -> First 10^8 bytes of the English Wikipedia dump from 2006.
~12.6M words

Relevant Publications

- Ustun, V., Rosenbloom, P. S., Sagae, K. & Demski, A. (2014). Distributed vector representations of words in the Sigma cognitive architecture. *Proceedings of the 7th Annual Conference on Artificial General Intelligence*.
- Kommers, C., Ustun, V., Demski, A. & Rosenbloom, P. (2015). Hierarchical reasoning with distributed vector representations. *Proceedings of the the 37th Annual Conference of the Cognitive Science Society*.

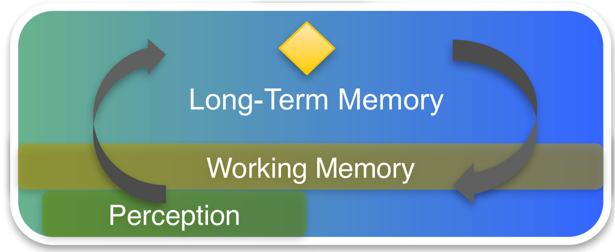
EPISODIC MEMORY

Episodic Memory

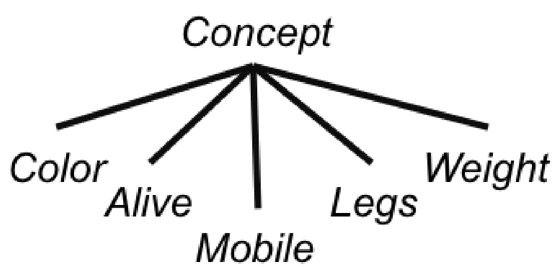
- A core competency in cognition
  - Back at least to Tulving (1983) in psychology
  - Back at least to Vere & Bickmore (1990) in AI
- Spans ability to
  - Store history of what has been experienced
    - Autobiographical and temporal
  - Selectively retrieve and reuse information from past episodes
  - 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

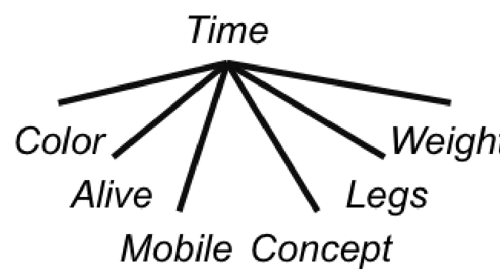
- Episode: Distributions over state predicates at decision time
- Three key processes
  - Learning a new episode
  - Selecting best previous time/episode
  - Retrieving features from selected time



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



Semantic

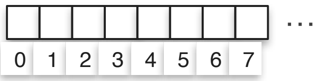


Episodic

```
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`

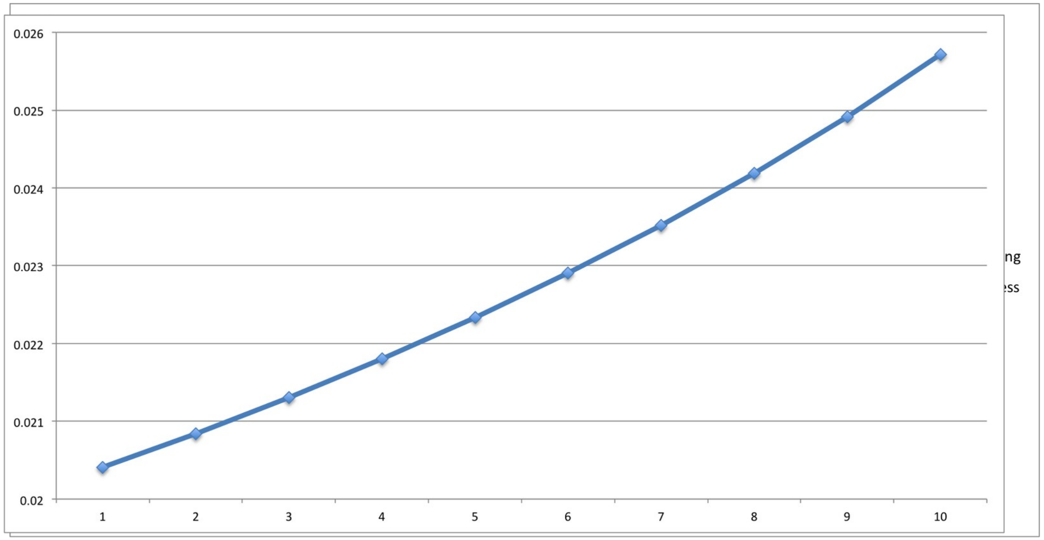
Time as a Function

- 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
```



USC Institute for Creative Technologies

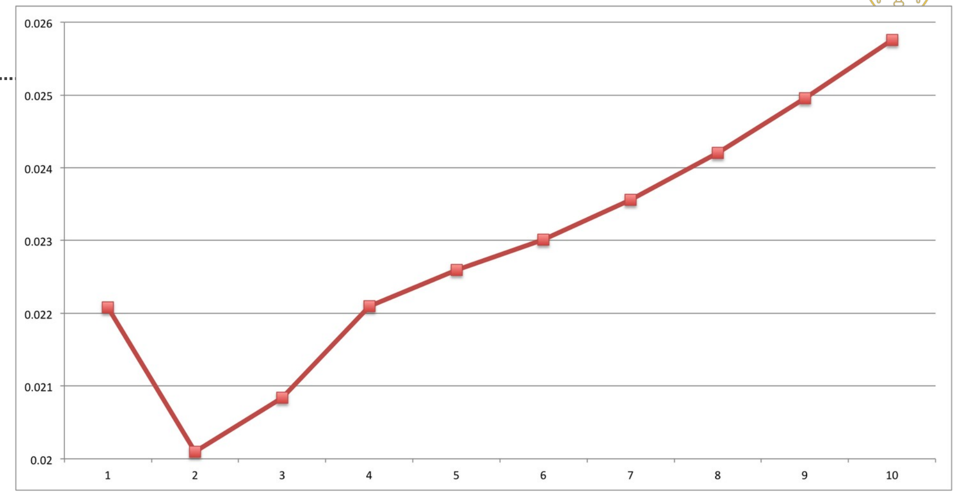
Mimics *base-level activation*

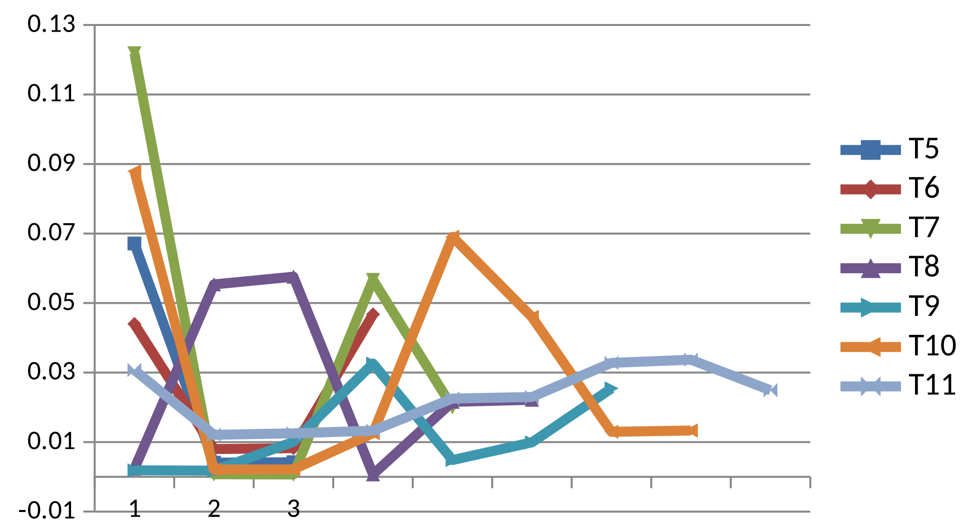
Results

	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





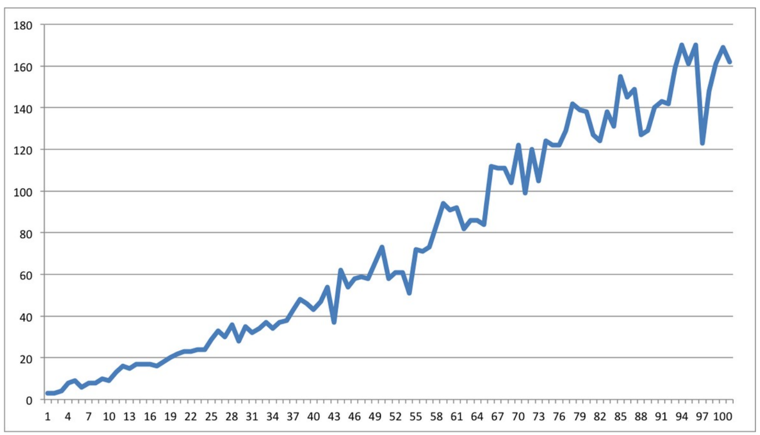
Efficiency

+: 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*

USC Institute for Creative Technologies

ARL

University of Southern California

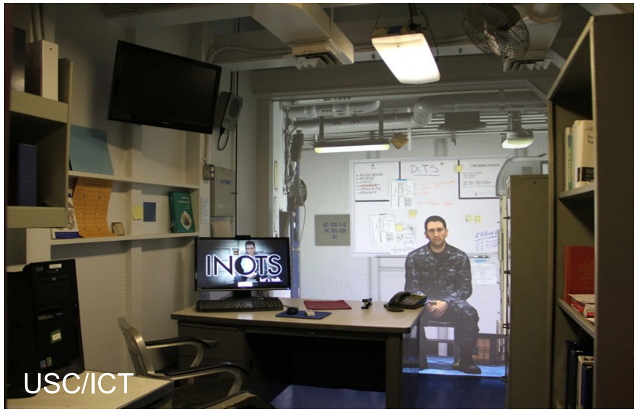
Relevant Publications

- Rosenbloom, P. S. (2010). Combining procedural and declarative knowledge in a graphical architecture. *Proceedings of the 10th International Conference on Cognitive Modeling*.
- Rosenblooom, P. S. (2014). Deconstructing episodic memory and learning in Sigma. *Proceedings of the 36th Annual Conference of the Cognitive Science Society*.

APPRAISAL & ATTENTION

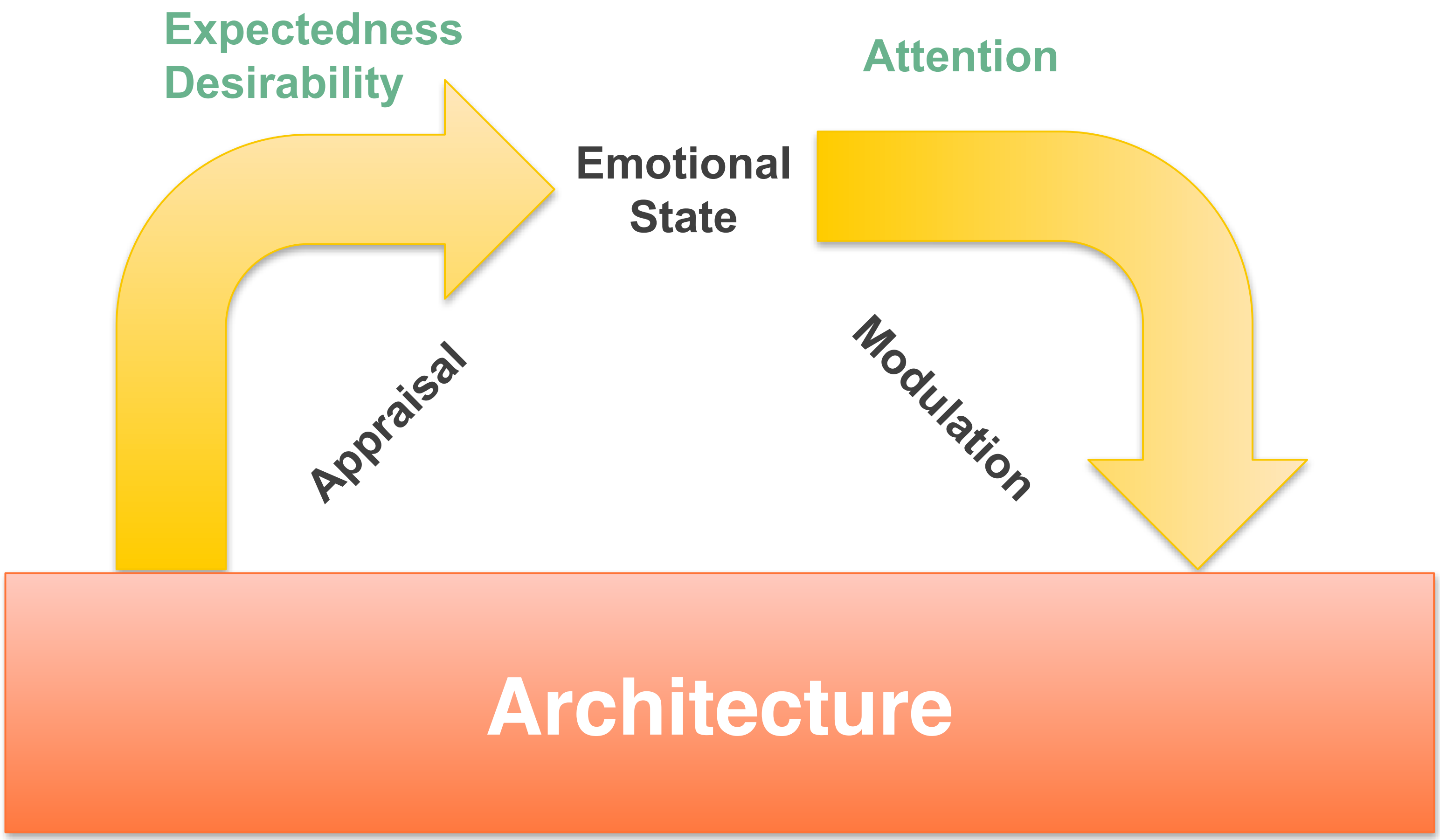
Emotions in Sigma

- Initial exploration motivated by combination of:
  - The theoretical desiderata of *grand unification* and *generic cognition*
  - The practical goal of building useful *virtual humans*
  - The hypothesis that emotion is *critical for surviving and thriving in complex physical and social environments*
    - Part of the *wisdom of evolution*



- Largely non-voluntary and immutable
  - Likely a significant architectural component
  - But also affected by knowledge and skills
- Focusing initially on architectural grounding

Architecturally Grounded Emotional Processing



Appraisal

- Typically considered first phase of emotion processing
  - Sense emotionally relevant state of system
  - Architectural proprioception at the lowest level
- Many different theories with different sets of appraisals
  - E.g., EMA includes *relevance*, *desirability*, *likelihood*, *expectedness*, *causal attribution*, *controllability*, and *changeability* (Gratch & Marsella)
- Initial work in Sigma focuses on
  - *Expectedness:* Extent an event is predicted by past knowledge
  - *Desirability:* Extent an event facilitates what is wanted

(Un)Expectedness

- Bayesian Theory of Surprise (Itti)
  - Surprise is difference between prior and posterior distributions
    - Adaptation of distributions is by *Bayesian belief updating*
    - Comparison of distributions is by *KL divergence*

$$S(D,M) = KL(P(M \mid D), P(M)) = \int_M P(M \mid D) \log \frac{P(M \mid D)}{P(M)} dM.$$

$D = \text{Data}$
$M = \text{Model}$

- Surprise (i.e., unexpectedness) in Sigma
  - Adaptation of distributions is by *gradient-descent learning*
  - Comparison of distributions is by *Hellinger distance*

$$S'(D,M) = HD(P(M \mid D), P(M)) = \sqrt{1 - \int \sqrt{P(M \mid D)(x) P(M)(x)}\, dx}$$

    - Can cope with 0s in $P(M)$ and is symmetric, so provides a metric

Visual Image:

G	Y	B	R
Y	G	B	R
R	R	R	R
G	B	B	B

.0283	.0283	.0287	.0283
.0283	.0283	.0283	.0283
.5739	.0287	.0287	.0287

Surprise map

Desirability

- Relationship of current state to goal
  - *Difference:* Hellinger difference between the two distributions

$$Difference(S,G) = HD(S,G) = \sqrt{1 - \int \sqrt{s(x)g(x)}\,dx}.$$

  - *Progress/similarity:* Bhattacharya coefficient between the distributions

$$Similarity(S,G) = BC(S,G) = \int \sqrt{s(x)g(x)}\,dx.$$

    - Inner portion of computation of Hellinger distance

Visual Search:

G	Y	B	R
Y	G	B	R
R	R	R	R
G	B	B	B

Visual field

Y

Goal

0	.5	0
.5	0	0
0	0	0

Progress Map

.07	0	.07
0	.07	.07
.07	.07	.07

Difference Map

Eight Puzzle:

1	2	3
8	4	5
7		6

State

1	2	3
8	X	4
7	6	5

Goal

.125	.125	.125
.125	0	0
.125	0	0

Progress Map

0	0	0
0	0	.125
0	.125	.125

Difference Map

Attention

- Effective allocation of limited resources
  - At all (three) layers of control
    - *Reactive:* Perceptual and low-level cognitive attention
    - *Deliberative:* Control of operator/action selection
    - *Reflective:* Focus of metacognition
- Bottom up: Data driven
- Top down: Goal driven

G	Y	B	R
Y	G	B	R
R	R	R	R
B	B	B	B

Initial field

G	Y	B	R
Y	G	B	R
R	R	R	R
G	B	B	B

Changed field

Y

Goal

.0283	.0283	.0287	.0283
.0283	.0283	.0283	.0283
.5739	.0287	.0287	.0287

Surprise map

0	.5	0
.5	0	0
0	0	0

Progress Map

G	Y	B,R
Y	G	B,R
R	R	R
G	B	B

Abstracted Message from Memory

Reduced one message from 160K regions down to 12

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B) \approx P(A) + P(B) - P(A)P(B)$$

.014	.261	.015	.014
.261	.014	.015	.014
.014	.014	.014	.014
.291	.015	.015	.015

Attention map

Relevant Publications

- Rosenbloom, P. S., Gratch, J. & Ustun, V. (2015). Towards emotion in Sigma: From appraisal to attention. *Proceedings of the 8th Conference on Artificial General Intelligence*.

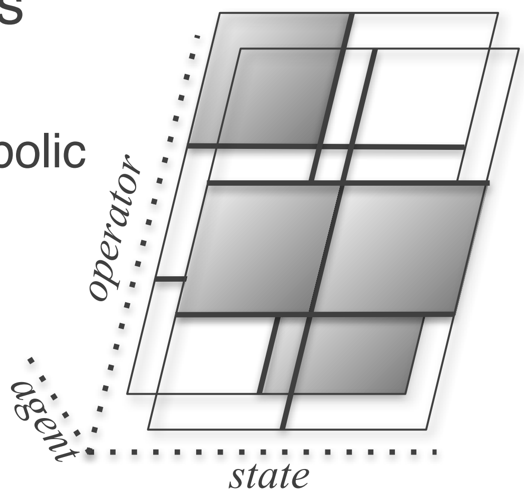
THEORY OF MIND
(& MULTIAGENT SYSTEMS)

Theory of Mind (ToM) in Sigma

- ToM models the minds of others, to enable for example:
  - Understanding multiagent situations
  - Participating in social interactions
- ToM approach based on *PsychSim* (Marsella & Pynadath)
  - Decision theoretic problem solving based on POMDPs
  - Recursive agent modeling
- Questions to be answered
  - Can Sigma elegantly extend to comparable ToM?
  - What are the benefits for ToM?
  - What new phenomena emerge from this combination?

Multiagent Sigma

- Core idea: Add agent argument to predicates
  - E.g., Selected(<u>*agent*</u>, *operator*, *state*)
  - A discrete dimension, but may be numeric or symbolic



- Details
  - *Agent* argument added to architectural predicates: *Selected*, *Impasse*
    - Directly yielded an agent-specific decision procedure, but needed to further modify impasse detection and removal to be agent-specific
  - When graph is defined, specify # of agents or list of agent names
- Could instead have defined a new graph for each agent, but this approach can enable sharing across agents

One-Shot, Two-Person Games

B

Prisoner's Dilemma	Cooperate	Defect
Cooperate	.3	.1(,.4)
Defect	.4(,.1)	.2

A

- Two players
- Played only once (not repeated)
  - So do not need to look beyond current decision
- *Symmetric*: Players have same payoff matrix
- *Asymmetric*: Players have distinct payoff matrices
- *Socially preferred outcome*: optimum in some sense
- *Na*... playe... payoff by ch... f oth...
  - ...est N...

B

A Rewards	Cooperate	Defect
Cooperate	.1	.2
Defect	.3	.1

A

B

B Rewards	Cooperate	Defect
Cooperate	.1	.1
Defect	.4	.4

A

Symmetric, One-Shot, Two-Person Games

Agent A

```
CONDITIONAL Payoff-A-A
    Conditions: Choice(A,B,op-b)
    Actions:    Choice(A,A,op-a)
```
[B's model of B]

```
    Function:   payoff(op-a,op-b)

CONDITIONAL Payoff-A-B
    Conditions: Choice(A,A,op-a)
    Actions:    Choice(A,B,op-b)
    Function:   payoff(op-b,op-a)
```

Agent B

```
CONDITIONAL Payoff-B-B
    Conditions: Choice(B,A,op-a)
    Actions:    Choice(B,B,op-b)
    Function:   payoff(op-b,op-a)
```
[B's model of A]

```
CONDITIONAL Payoff-B-A
    Conditions: Choice(B,B,op-b)
    Actions:    Choice(B,A,op-a)
    Function:   payoff(op-a,op-b)
```

```
CONDITIONAL Select-Own-Op
    Conditions: Choice(ag,ag,op)
    Actions:    Selected(ag,op)
```

Other / Choosing

Prisoner's Dilemma	Cooperate	Defect	A Result	B Result
Cooperate	.3	.1	.43	.43
Defect	.4	.2	.57	.57

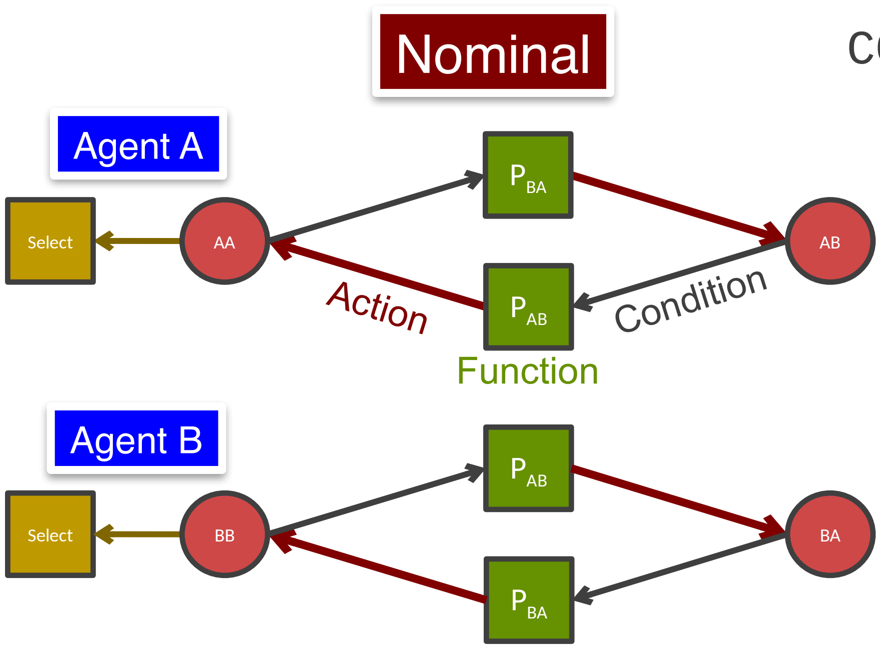
602 Messages

Other / Choosing

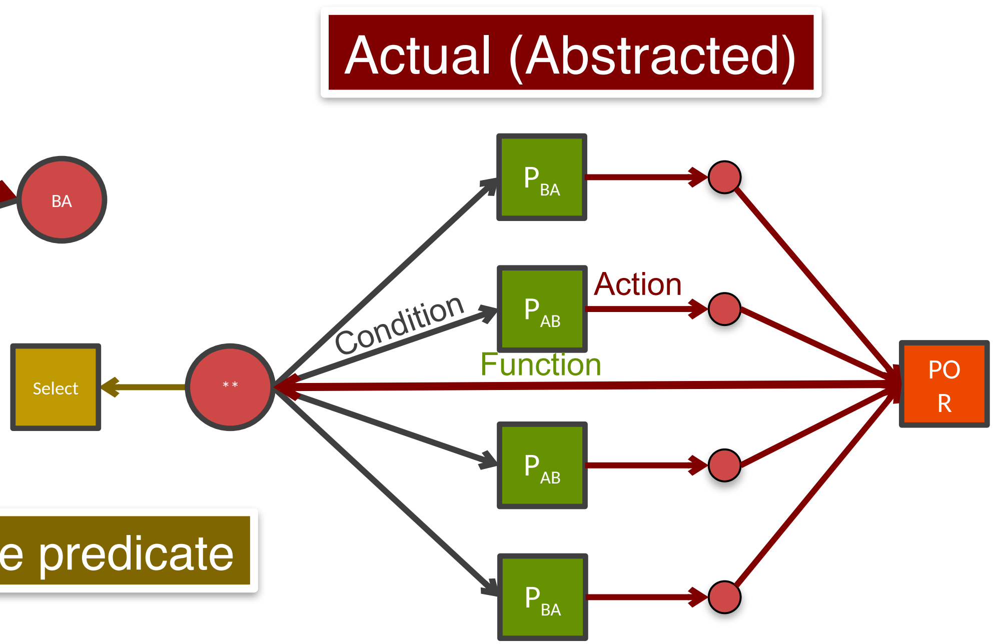
Stag Hunt	Cooperate	Defect	A Result	B Result
Cooperate	.25	0	.54	.54
Defect	.1	.1	.46	.46

962 Messages

Graph Structure




```
CONDITIONAL Payoff-A-A
    Conditions: Choice(A,B,op-b)
    Actions:    Choice(A,A,op-a)
    Function:   payoff(op-a,op-b)
```



All one predicate

Asymmetric, One-Shot, Two-Person Games

```
CONDITIONAL Payoff-A-A
    Conditions: Choice(A,B,op-b)
    Actions:    Choice(A,A,op-a)
    Function:   payoff(A,op-a,op-b)

CONDITIONAL Payoff-B-B
    Conditions: Choice(B,A,op-a)
    Actions:    Choice(B,B,op-b)
    Function:   payoff(B,op-b,op-a)

CONDITIONAL Payoff-A-B
    Conditions: Choice(A,A,op-a)
                Model(m)
    Actions:    Choice(A,B,op-b)
    Function:   payoff(m,op-b,op-a)

CONDITIONAL Payoff-B-A
    Conditions: Choice(B,B,op-b)
                Model(m)
    Actions:    Choice(B,A,op-a)
    Function:   payoff(m,op-a,op-b)

CONDITIONAL Select-Own-Op
    Conditions: Choice(ag,ag,op)
    Actions:    Selected(ag,op)
```

B

A Rewards	Cooperate	Defect
A: Cooperate	.1	.2
A: Defect	.3	.1

B

B Rewards	Cooperate	Defect
A: Cooperate	.1	.1
A: Defect	.4	.4

Choosing

Correct Other	A Result	B Result
Cooperate	.51	.29
Defect	.49	.71

374 Messages

Choosing

Other as Self	A Result	B Result
Cooperate	.47	.29
Defect	.53	.71

636 Messages

Sequential Games

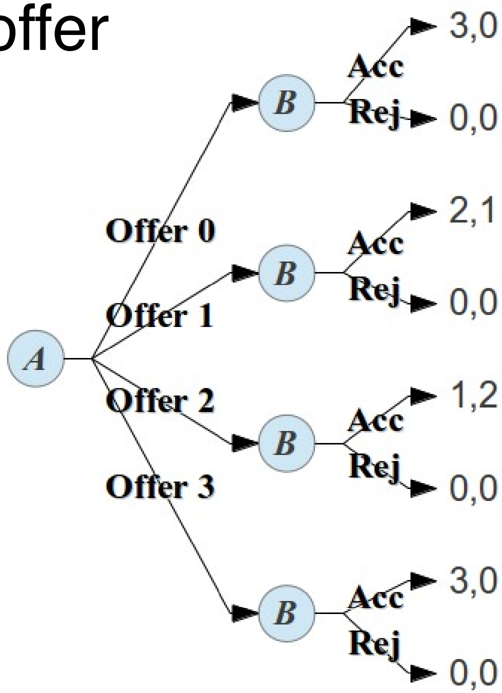
- Players (A, B) alternate moves
  - E.g., *Ultimatum*, *centipede* and *negotiation*
- Decision-theoretic approach with *softmax combination*
  - Use expected value at each level of search
  - Action **P**s assumed exponential in their utilities (à la Boltzmann)
- There may be many *Nash equilibria*
- Instead seek stricter concept of *subgame perfection*
  - Overall strategy is a Nash equilibrium over any subgame
- **Key result:** Games solvable in two modes:
  - Automatic/reactive/system-1
  - Controlled/deliberate/system-2

  Both modes well documented in humans for general processing

  Combination not found previously in ToM models

The Ultimatum Game

- *A* starts with a fixed amount of money (3)
- *A* decides how much (in 0-3) to offer *B*
- *B* decides whether or not to accept the offer
  - If *B* accepts, each gets the resulting amount
  - If *B* rejects, both get 0
- Each has a utility function over money
  - E.g., <.1, .4, .7, 1>



Automatic/Reactive Approach

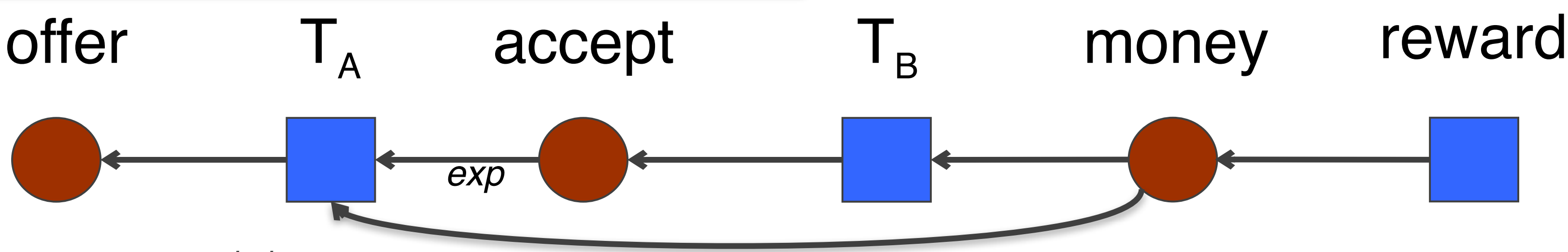
- A trellis (factor) graph in LTM with one stage per move
- Focus on backwards messages from reward(s)

```
CONDITIONAL Transition-B
  Conditions: Money(agent:B quantity:moneyb)
  Condacts: Accept(offer:offer acceptance:choice)
  Function(choice,offer,moneyb): 1<T,0,0>, 1<T,1,1>, 1<T,2,2>,
                                 1<T,3,3>, 1<F,*,0>
```

$$V_B(offer, choice) = \text{Reward}_B(\text{Money}_B(offer, choice))$$

$$V_A(offer) = \sum_{choice} \text{Reward}_A(\text{Money}_A(offer, choice)) * e^{K * V_B(offer, choice)}$$

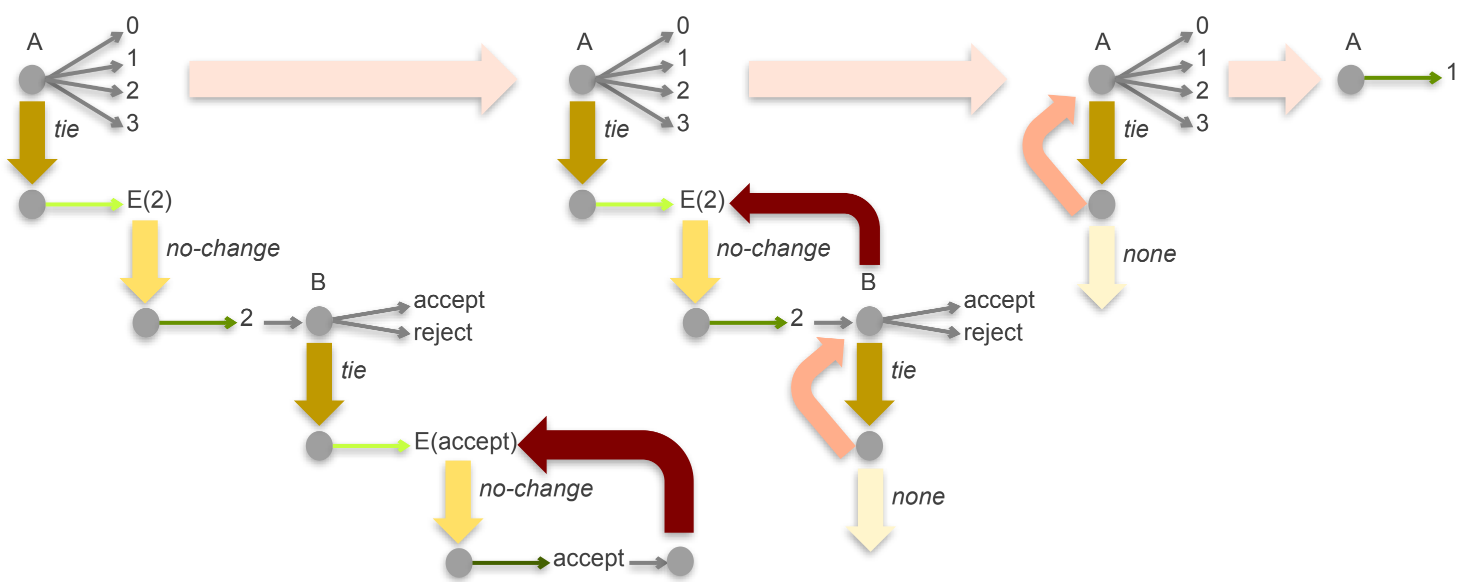
```
CONDITIONAL Reward
  Condacts: Money(agent:agent quantity:money)
  Function(agent,money): .1<*,0>, .4<*,1>,
                         .7<*,2>, 1<*,3>
```




```
CONDITIONAL Transition-A
  Conditions: Money(agent:A quantity:moneya)
              Accept-E(offer:offer acceptance:choice)
  Condacts: Offer(agent:A quantity:offer)
  Function(choice,offer,moneya): 1<T,0,3>, 1<T,1,2>, 1<T,2,1>,
                                 1<T,3,0>, 1<F,*,0>
```

Controlled/Deliberate(Reflective) Approach

- Decision-theoretic problem-space search across metalevels
  - Very Soar-like, but with softmax combination
    - Depends on *summary product* and Sigma's *mixed* aspect
  - Corresponds to PsychSim's online reasoning



Comments on the Ultimatum Game

- Automatic version (5 conditionals)
  - A's normalized distribution over offers: <.315, .399, .229, .057>
  - 1 decision (94 messages) and **.02 s** (on a MacBook Air)
- Controlled version (19 conditionals)
  - A's normalized distribution over offers: <.314, .400, .229, .057>
  - 72 decisions (868 messages/decision) and **126.69 s**

Distributions Comparable

Speed Ratio >6000

- Same result, with distinct computational properties
  - Automatic is fast and occurs in parallel with other memory processing, but is not (easily) penetrable by new bits of other knowledge
  - Controlled is slow, sequential, but can (easily) integrate new knowledge
  - Distinction also maps onto expert versus novice behavior in general

Raises possibility of a generalization of Soar's chunking mechanism

- Compile/learn automatic trellises from controlled problem solving
- Finer grained, mixed(/hybrid) learning mechanism

A Negotiation Domain

- Two agents, *A* and *B*
  - *A* learns
  - *B* does not
- Negotiating over an allocation of fruit: apples, oranges
  - Alternate offers to modify allocation on table
  - Each can accept current allocation, ending negotiation
  - Each has own reward function that depends on final allocation

Multiagent Learning in Negotiation Domain

- Four distinct multiagent reinforcement learning models
  - Without explicitly modeling other agent
    - Other agent effectively treated as part of environment
  - With a stationary policy model of other agent
    - Learned from experience with other agent's actions
  - With a set of possible reward functions for other agent
    - Learns to determine which is more likely
  - By *inverse reinforcement learning* (IRL) of other agent's reward
    - Learned from experience with other agent's actions
    - But inverts processing to learn other agent's reward function rather than directly using it to learn other agent's policy

Results

- Ran all four versions of A against two versions of B
  - Cooperative vs. Competitive
  - Switched B's policy after 1000 decision cycles
- All four multiagent RL methods converge to (roughly) optimal
  - All four Q functions are capable of representing the optimal policy
  - It thus follows a stationary policy, with some noise

Model of *B*	None	Stationary Policy	Reward Subset	IRL
R-A (coop. *B*)	7.11	7.13	7.12	**7.17**
R-A (-> comp.)	5.82	5.80	**5.85**	5.82
R-A (comp. *B*)	5.88	**5.88**	5.83	5.85
R-A (-> coop.)	7.00	6.96	**7.08**	6.99

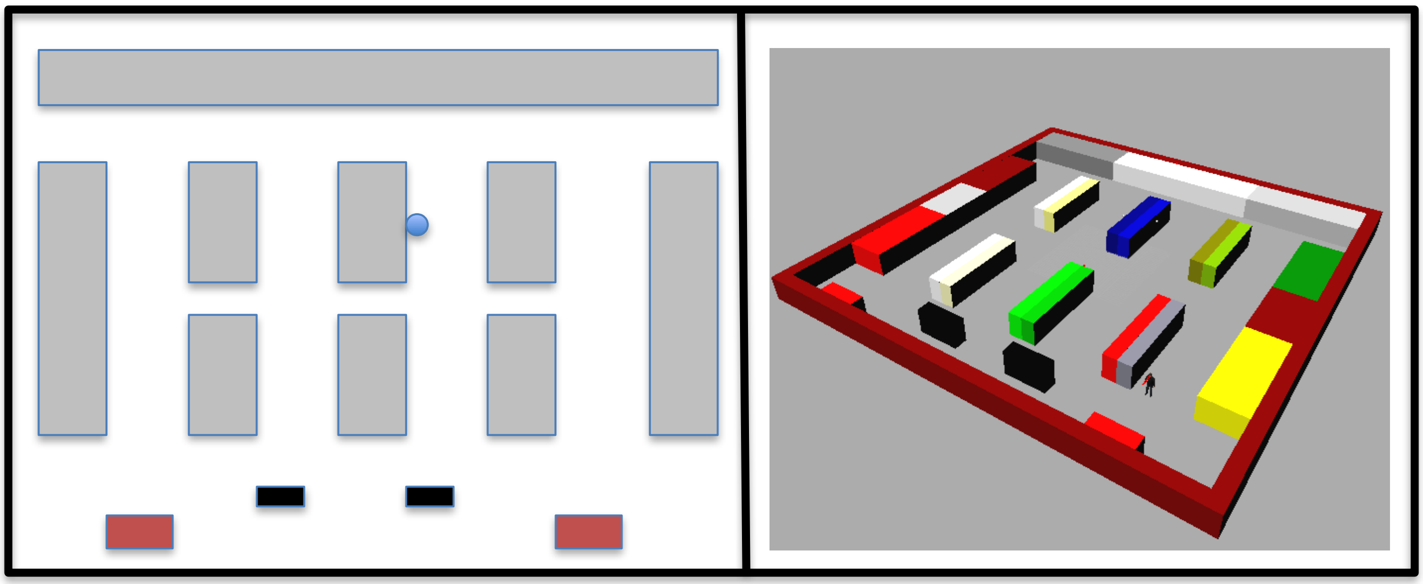
Relevant Publications

- Pynadath, D. V., Rosenbloom, P. S., Marsella, S. C. & Li, L. (2013). Modeling two-player games in the Sigma graphical cognitive architecture. *Proceedings of the 6th Conference on Artificial General Intelligence*.
- Pynadath, D. V., Rosenbloom, P. S. & Marsella, S. C. (2014). Reinforcement learning for adaptive Theory of Mind in the Sigma cognitive architecture. *Proceedings of the 7th Annual Conference on Artificial General Intelligence*.

INTERACTIVE, ADAPTIVE VIRTUAL HUMANS

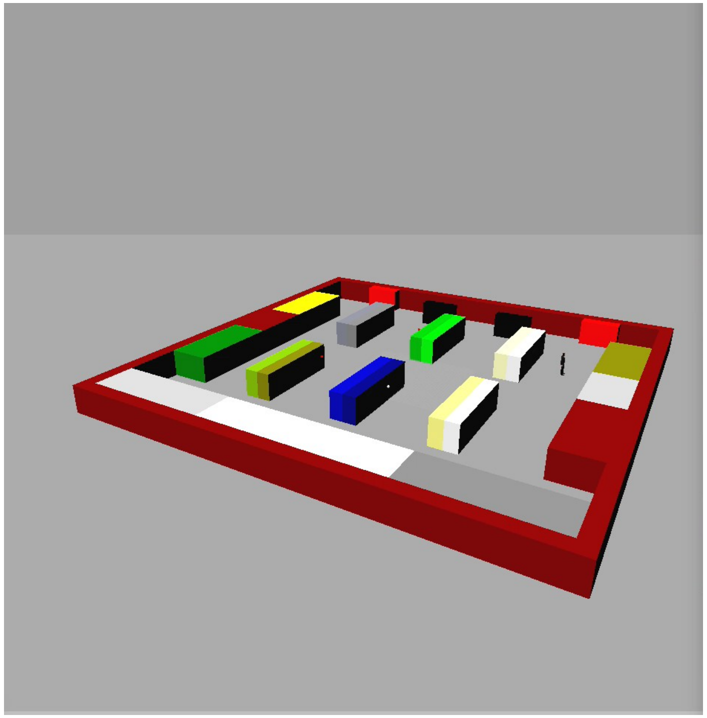
Interactive, Adaptive Virtual Humans

- Control behavior of SmartBody VH(s) in a retail store scenario
  - A civilian instance of a *physical security system*

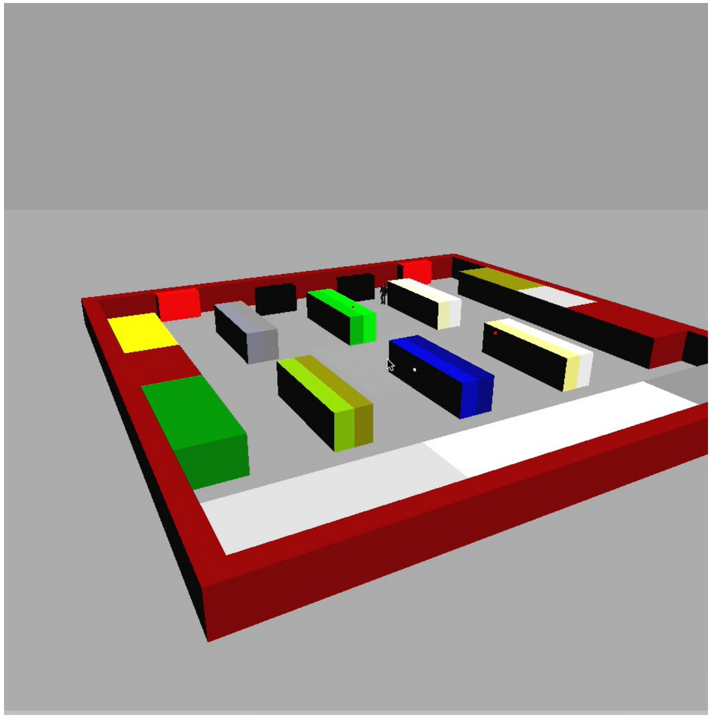


- Rule-based, probabilistic and social reasoning (ToM)
- Simultaneous localization and mapping (SLAM)
- Multiagent reinforcement learning (RL)
- [Appraisal+attention-based control]

Simultaneous Localization and Mapping (SLAM)



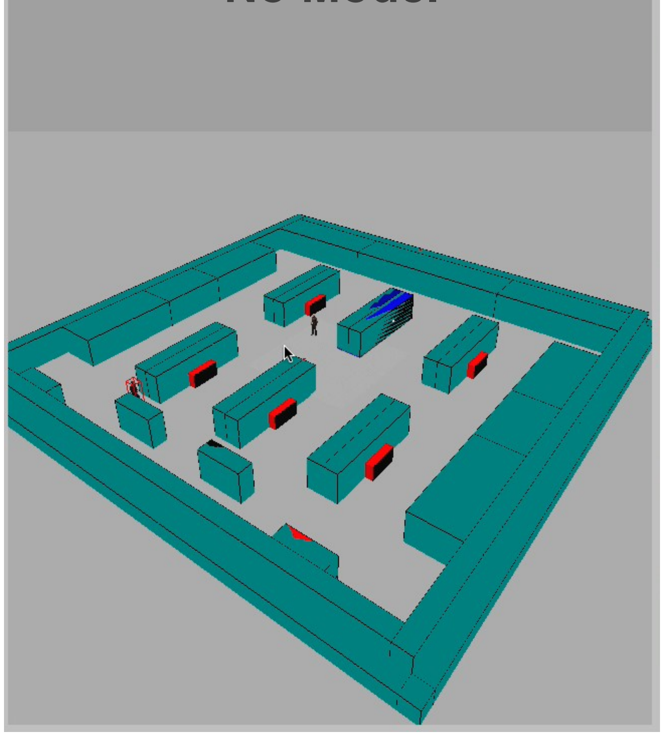
No Map



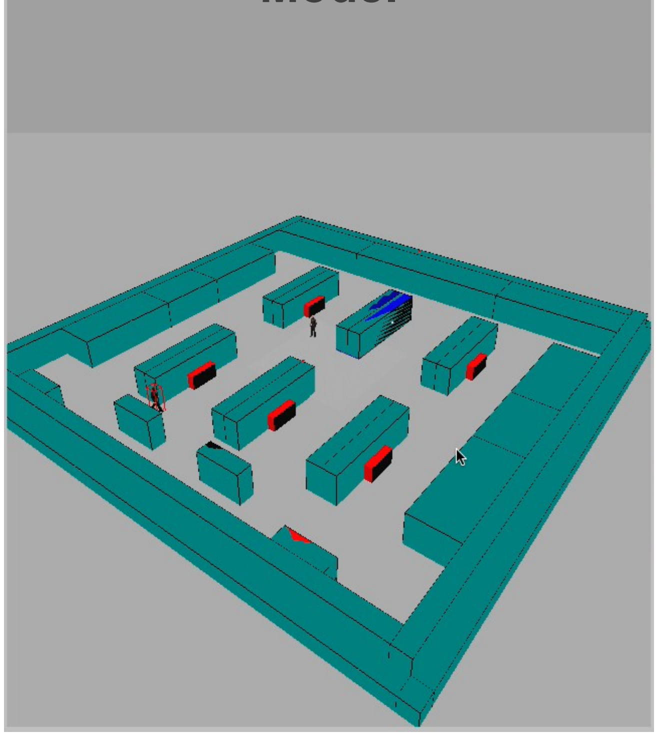
Map

Multiagent Reinforcement Learning (RL)

No Model



Model



Relevant Publications

- Ustun, V. & Rosenbloom, P. S. (2015). Towards adaptive, interactive virtual humans in Sigma. *Proceedings of the 15th International Conference on Intelligent Virtual Agents*.
- Ustun, V., Rosenbloom, P. S., Kim, J. & Li, L. (2015). Building high fidelity human behavior models in the Sigma cognitive architecture. *Proceedings of the 2015 Winter Simulation Conference*.

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`, `ps`
  - Working memory: `pwm` , `ppwm`, `pwmb`
- Graph: `g`
- Debugging
  - Recompute message: `debug-message`
  - Print pattern matches: `ppm`
- Learning: `learn`

Current and Near Future Topics

- Scaling up memory, reasoning and learning
- Continuous speech understanding, and its integration with language and cognition
- Theory of Mind
- Emotion/affect and its relationship to the architecture
- Distributed vectors/semantics (i.e., word embeddings)
- (Deep) neural networks
- A generalized skill acquisition mechanism (chunking)
- A new level below the graphical architectural
- Exploiting parallelism and GPUs for efficiency
- Interactive, adaptive, intelligent, emotional virtual humans

Broad Set of Capabilities from Space of Variations

Highlighting *Functional Elegance* and *Grand Unification*

- Rule memory
- Episodic memory
- Semantic memory
- Mental imagery
- Edge detectors

- Preference-based decisions
- POMDP-based decisions
- Localization

...

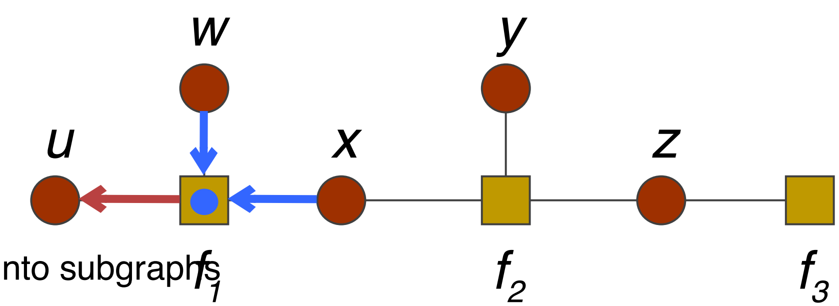
Closed vs. open world functions
Universal vs. unique variables
Discrete vs. continuous variables
Boolean vs. numeric function values

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

.5y	0
x+.3y	1
x-y	1
0	6x

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

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



Piecewise Continuous Functions

Factor graphs w/ Summary Product