Towards an *Architecture after Next* for ICT

Towards a New Cognitive Hourglass

Resolving the Diversity Dilemma in Cognitive Architecture via a Uniform Implementation Level Based on Graphical Models

Paul S. Rosenbloom
Department of Computer Science & Institute for Creative Technologies
University of Southern California
August 12, 2009

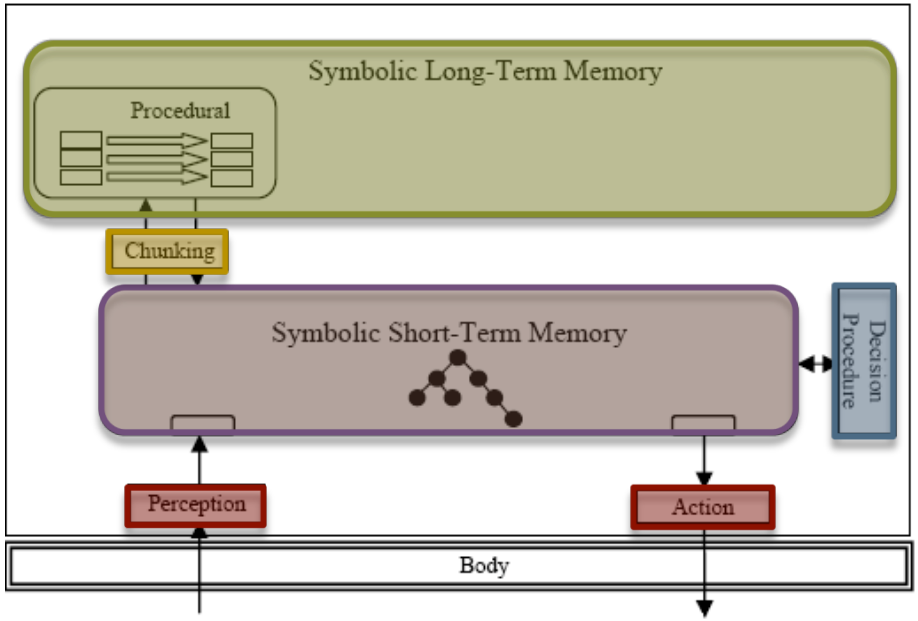
Planned Near-Term CAWG Topics

- Review vision, background and published work
  - 1-2 sessions, as needed to get through material at understandable pace
- A Bayesian framework for the elaboration phase
  - Mixing symbolic and probabilistic processing
  - Integrating procedural, semantic and episodic knowledge
- Status update on implementing Bayesian framework
- Function/memory/message representation
  - Integrating symbolic, integral, and continuous data
  - Yielding closed, efficient computation

Cognitive Architecture

- Hypothesis about fixed structure underlying cognition
  - Defines core memories, reasoning processes, learning mechanisms, external interfaces, etc.
- Yields intelligent behavior when combined with knowledge in memories
  - Including more advanced reasoning, learning, etc.
- May model human cognition, strive for human-like intelligence, or be purely artificial

Example: Soar (3-8)



Soar 3-8

- Symbolic working memory
- Long-term memory of rules
- Decide what to do based on preferences retrieved into working memory
- Reflect when can't decide
- Learn results of reflection
- Interact with world

Systems Levels in Cognition

- In the large, cognitive architecture is a theory about one or more *systems levels* in an intelligent being
  - Usually part of Cognitive Band
- At each level, a combination of structures and processes implements basic elements at the next higher level

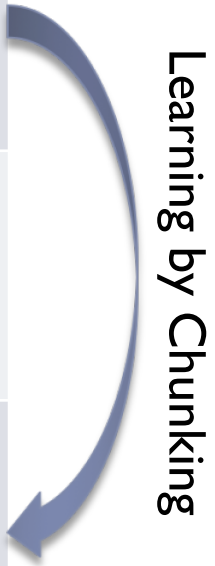
TIME SCALE OF HUMAN ACTION

Scale (sec)	Time Units	System	World (theory)
10^{7}	months		
10^{6}	weeks		SOCIAL BAND
10^{5}	days		
10^{4}	hours	Task	
10^{3}	10 min	Task	RATIONAL BAND
10^{2}	minutes	Task	
10^{1}	10 sec	Unit task	
10^{0}	1 sec	Operations	COGNITIVE BAND
10^{-1}	100 ms	Deliberate act	
10^{-2}	10 ms	Neural circuit	
10^{-3}	1 ms	Neuron	BIOLOGICAL BAND
10^{-4}	100 µs	Organelle	

(Newell, 1990)

Hierarchical View of Soar (3-8)

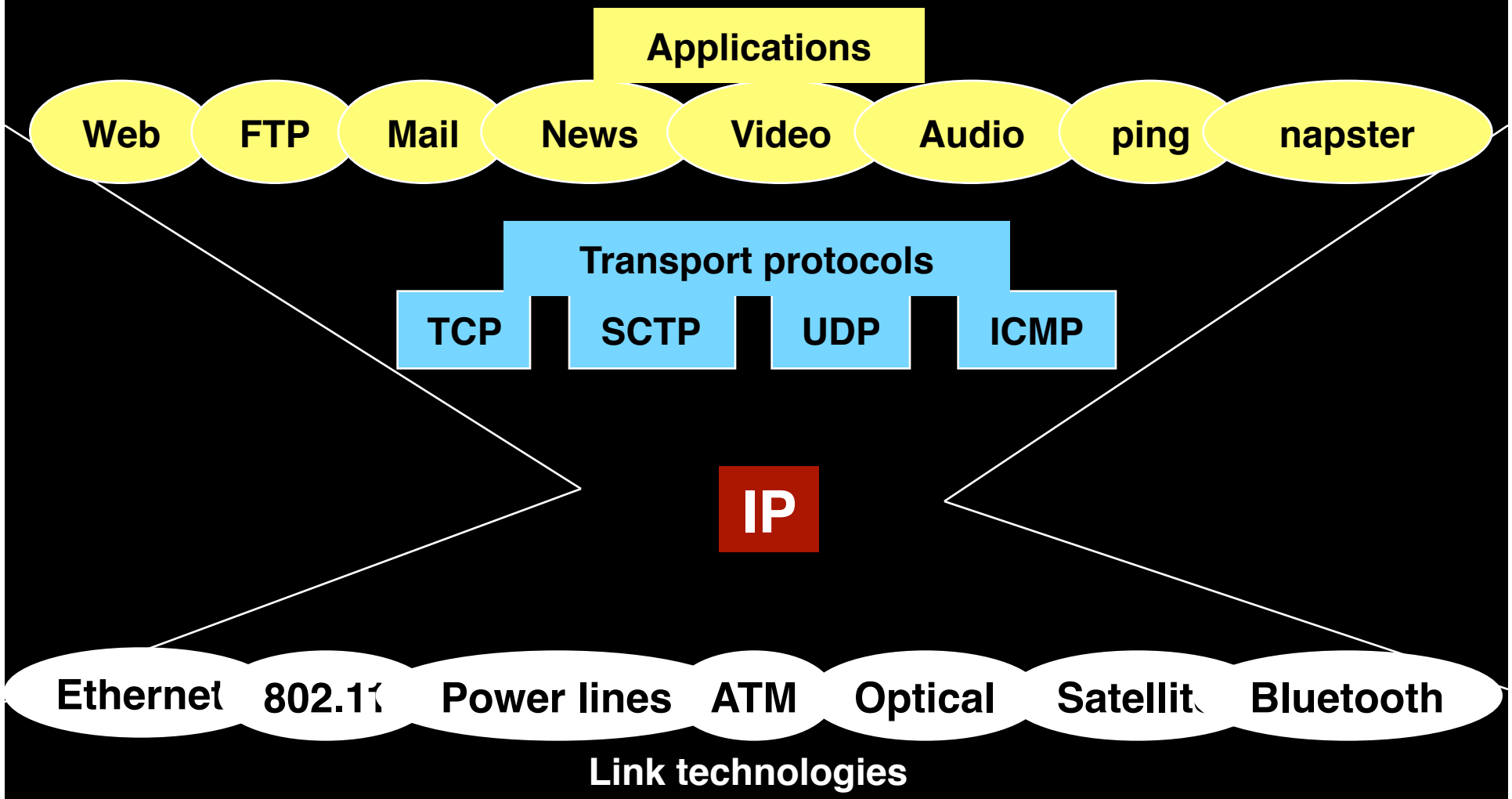
Scale	Functionality	Mechanism	Details
1 sec	Reflective	Problem Space Search	Impasse/Subgoal if can't Decide
100 ms	Deliberative	Decision Cycle	Preference-based Decisions upon Quiescence
10 ms	Reactive	Elaboration Cycle	Parallel Rule Match & Firing



Architecture/Level Girth

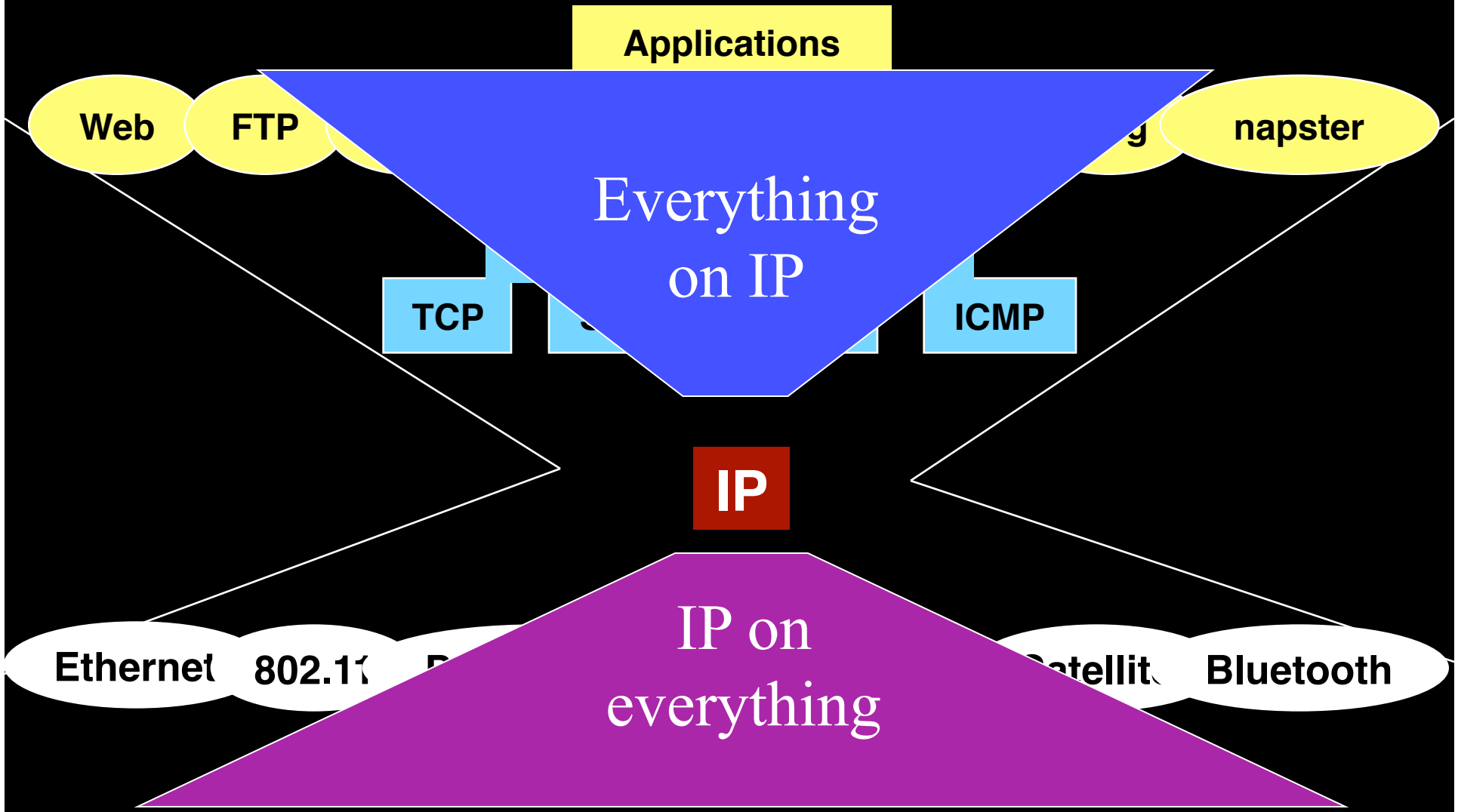
- An architecture/level's *girth* is its range of structures & processes
  - Uniformity versus diversity
- Uniformity: Minimal mechanisms combining in general ways
  - Appeals to simplicity and elegance
  - The "physicist's approach"
  - Challenge: Achieving full range of required functionality/coverage
- Diversity: Large variety of specialized mechanisms
  - Appeals to functionality and optimization
  - The "biologist's approach"
  - Challenge: Achieving integrability, extensibility and maintainability
- Across a hierarchy, level girth may stay comparable or vary
  - Physicists and biologists likely assume uniform
  - Network researchers assume hourglass

The Internet hourglass



From Hari Balakrishnan

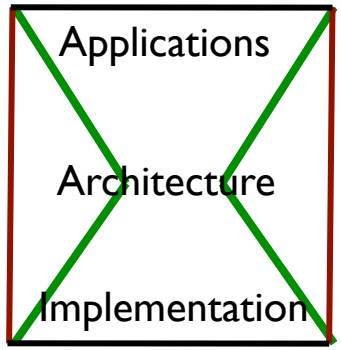
The Internet hourglass



From Hari Balakrishnan

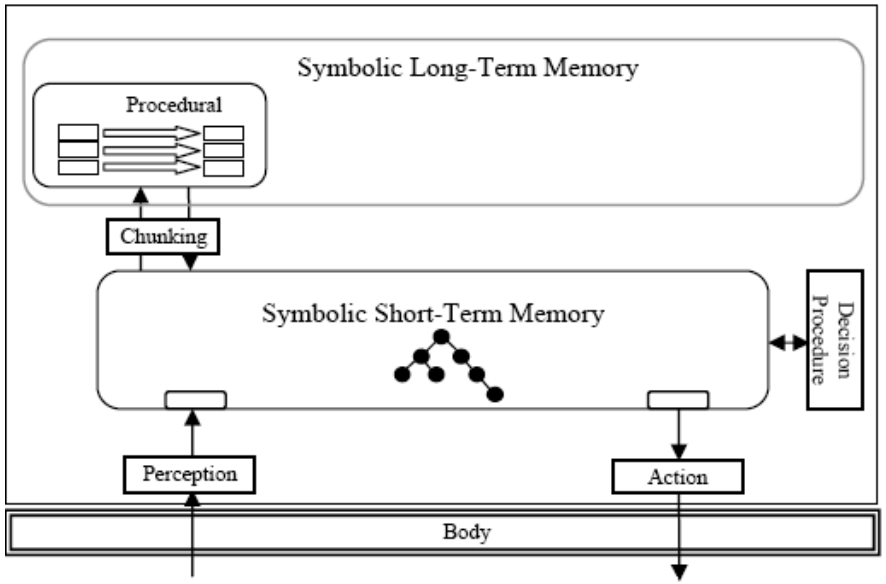
What About Cognition?

- Top (applications) is clearly diverse
  - Key part of what architectures try to explain
- Bottom is likely diverse as well
  - Physicalism: Grounded in diversity of biology
  - Strong AI: Also groundable in other technologies
- Is the waist uniform or diverse?
  - Hourglass or rectangle
  - Traditionally question about the architecture

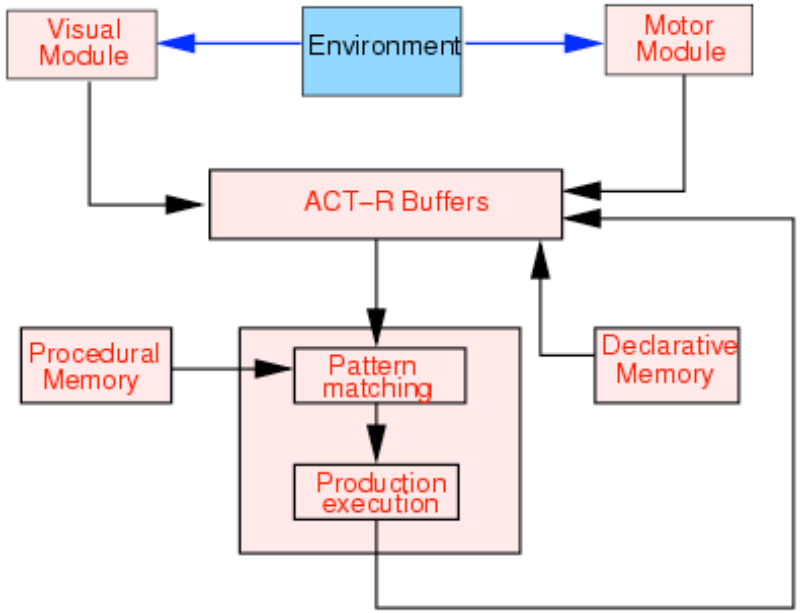


Examples

- Soar (3-8) is a traditional uniform architecture
- ACT-R is a traditional diverse architecture



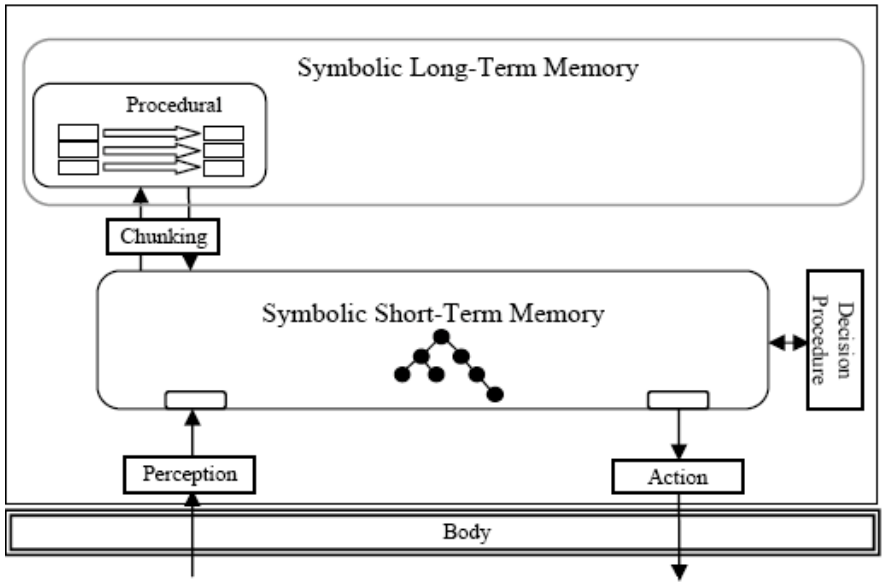
Soar 3-8



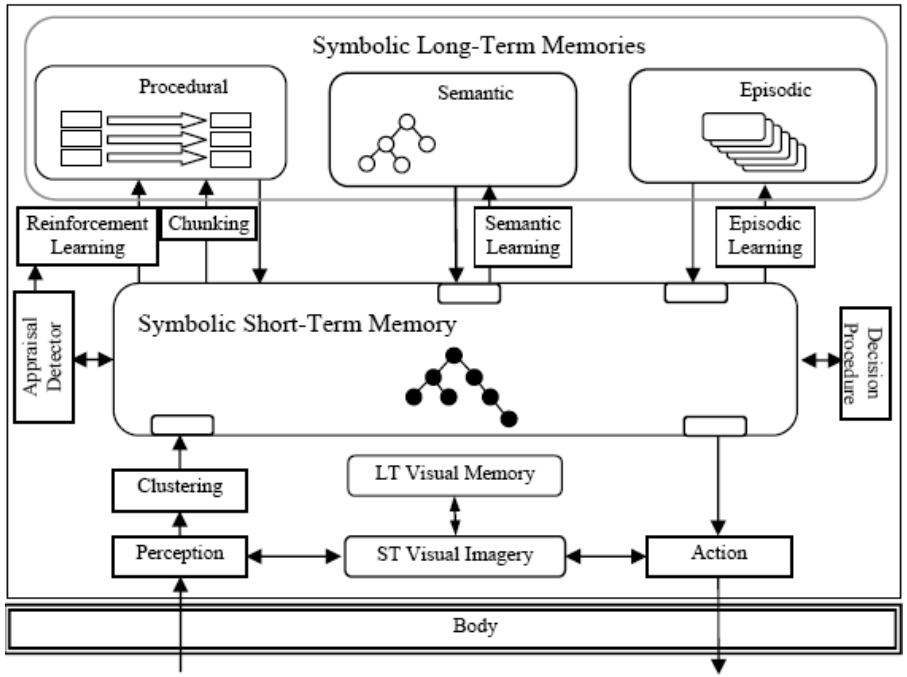
ACT-R

Examples

- Soar is a traditional uniform architecture
- ACT-R is a traditional diverse architecture
- Recently Soar 9 has become highly diverse



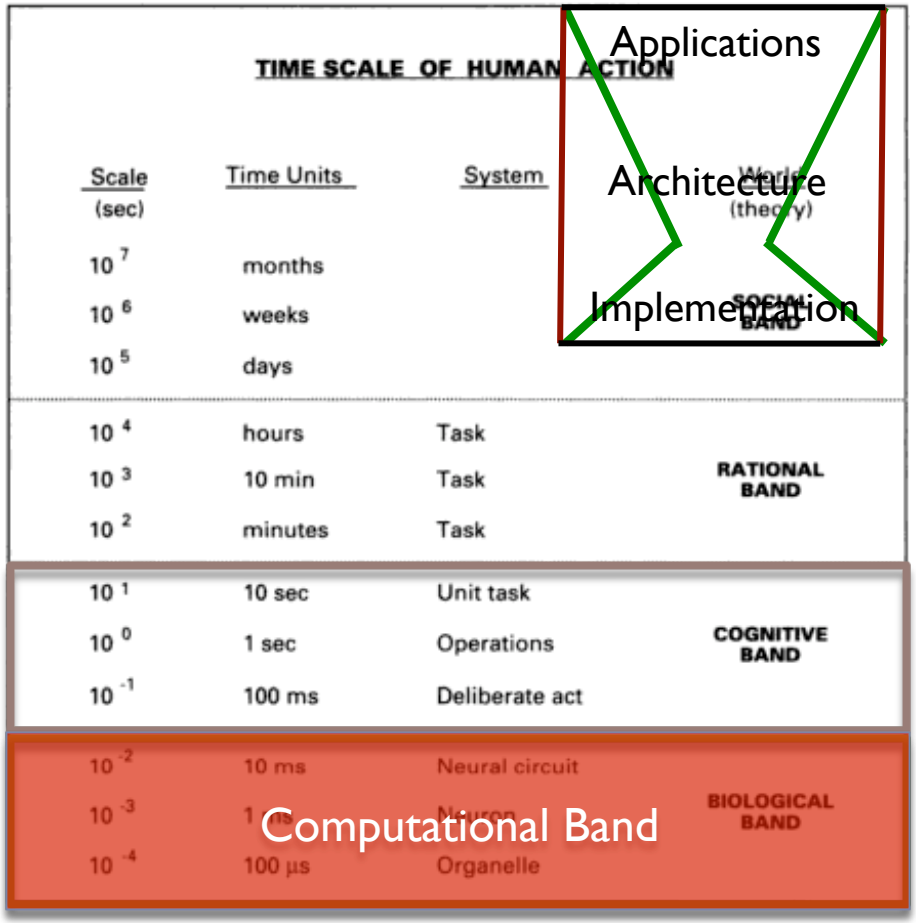
Soar 3-8



Soar 9

Towards Reconciling Uniformity and Diversity

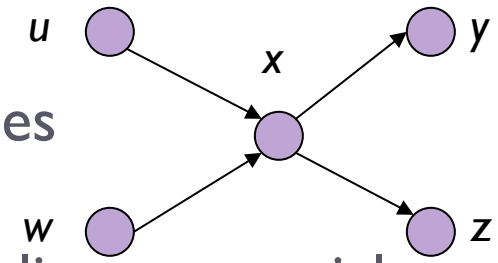
- Accept diversity at the architectural level
- Move search for uniformity down to *implementation level*
  - *Biological Band* in humans
    - Locus of neural modeling
  - *Computational Band* in AI
    - Normally just Lisp, C, Java, etc.
      - Impacts efficiency and robustness but usually not part of theory
- Base on *graphical models*
  - Uniform approach to symbol, probability & signal processing
  - Related to neural networks but broader

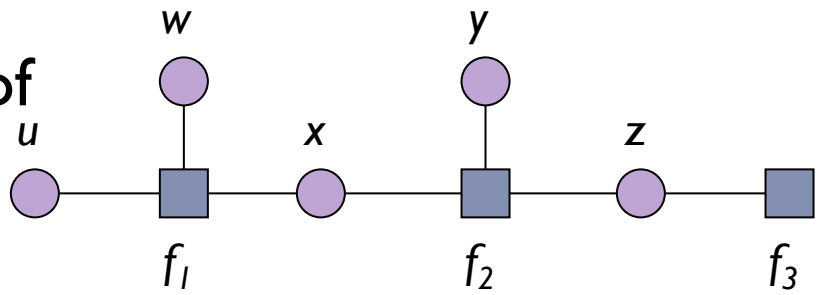


(Newell, 1990)

Graphical Models

- Efficient computation with multivariate functions: $f(u,w,x,y,z)$
  - By decomposition over partial independencies among arguments
  - For constraints, probabilities, speech, etc.
- Come in a variety of related flavors
  - **Bayesian networks**: Directed, variable nodes
    - E.g, $p(u,w,x,y,z) = p(u)p(w)p(x|u,w)p(y|x)p(z|x)$
  - **Markov networks**: Und., variable nodes & clique potentials
    - Basis for *Markov logic* and *Alchemy*
  - **Factor graphs**: Und., variable & factor nodes
    - E.g., $f(u,w,x,y,z) = f_1(u,w,x)f_2(x,y,z)f_3(z)$
- Compute marginals via variants of
  - Sum-product (message passing)
  - Monte Carlo (sampling)



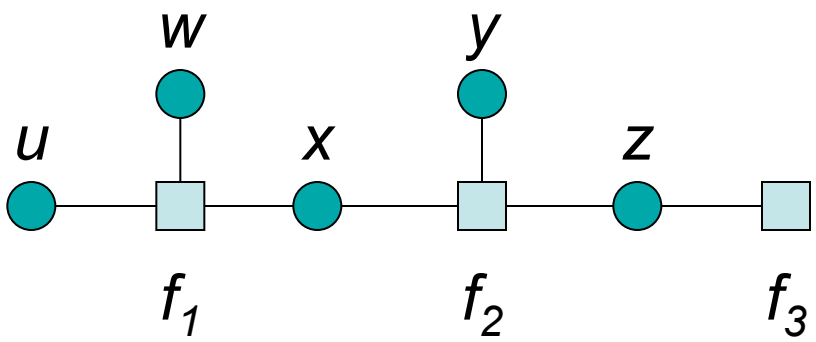


Potential for the Implementation Level

- State-of-the-art algorithms for *symbol*, *probability* and *signal* processing all derivable from the *sum-product algorithm*
  - Belief propagation in Bayesian networks
  - Forward-backward in hidden Markov models
  - Kalman filters, Viterbi algorithm, FFT, turbo decoding
  - Arc-consistency in constraint diagrams
- Potential to go beyond existing architectures to yield an effective and uniform basis for:
  - Fusing symbolic and probabilistic reasoning (mixed)
  - Unifying cognition with perception and motor control (hybrid)
  - Bridging from symbolic to neural processing
- Raises hope of a uniform implementation level that integrates broad functionality at the architecture level

Factor Graphs

- Decompose functions into product of nearly independent *factors* (or *local functions*)
  - E.g., $f(u,w,x,y,z) = f_1(u,w,x)f_2(x,y,z)f_3(z)$
- Draw as bipartite graph
  - Nodes for factors and variables
  - Links between factors and their variables



Bayesian Network (BN) Example

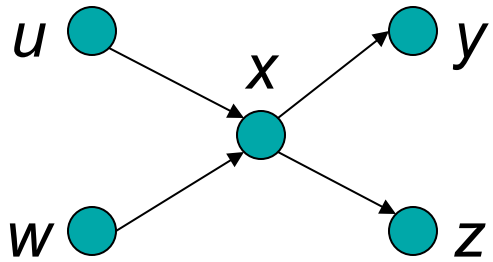
- Probabilistic reasoning involves computations of various quantities from *joint probability distributions* over random variables
  - E.g., compute the *marginal probability* $p(u)$ from the joint probability distribution $p(u,w,x,y,z)$ by summing out/over all of the other variables
    - $p(u) = \text{SUM}_{w,x,y,z}\, p(u,w,x,y,z)$
  - Key for tractability is to do so without having to explicitly examine every combination of values of all of the other variables

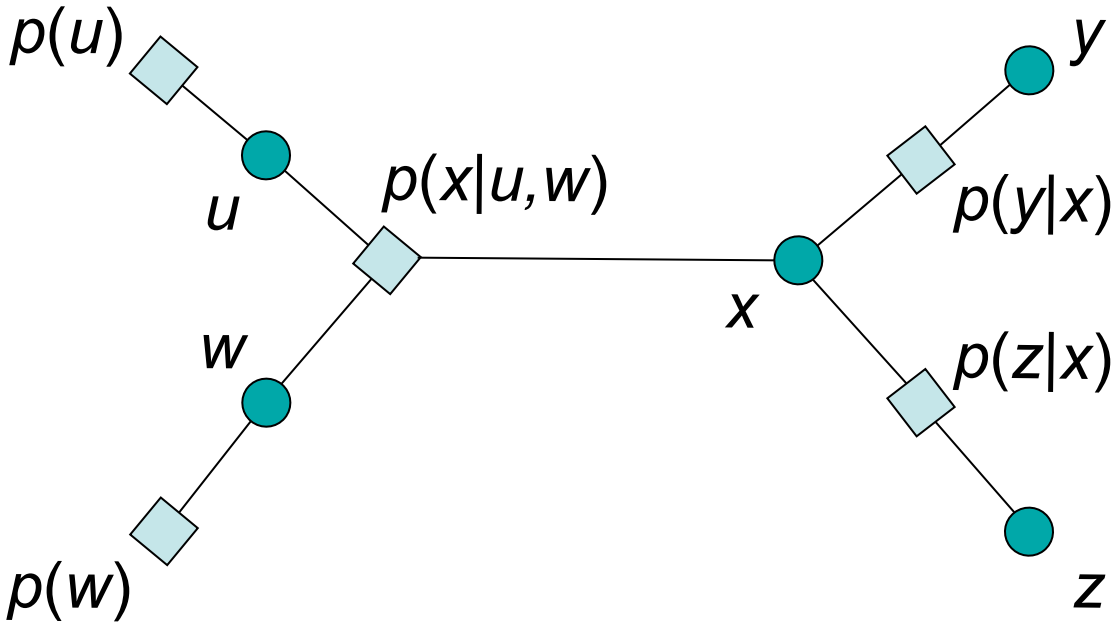
BN Example (cont.)

- A Bayesian network represents a joint probability distribution as the product of the conditional probabilities of each random variable
  - E.g., $p(u,w,x,y,z) = p(u)p(w)p(x|u,w)p(y|x)p(z|x)$
- Each conditional probability only involves a subset of the total set of variables (its *parents*)
  - Each variable is *conditionally independent* of all of the other variables, given its parents
    - I.e., once you know the values of the parent variables, the probability of a value of the variable can be determined independently of the values of all of the other variables
- Can radically reduce scope of combinatorics

Example BN and Factor Graph

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





Inference in Factor Graphs

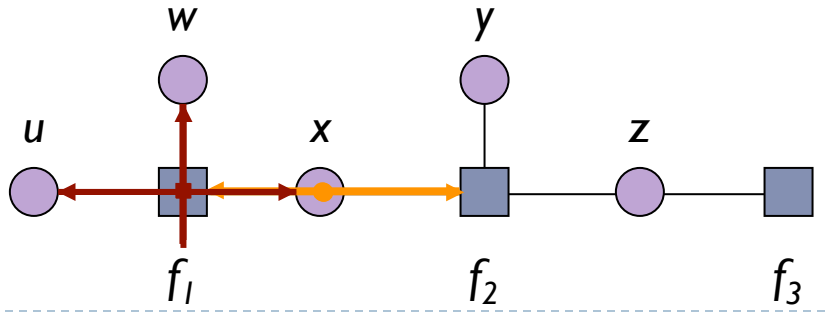
- Reasoning occurs via message passing/propagation
  - From variable nodes to factor nodes
  - From factor nodes to variable nodes
- Each message from one node to a second node conveys some kind of information about the binding of a variable based on the information the first node has received from all of its neighbors other than the second node
  - Warning: A variable must take on a specific value
    - Simple setting of variable values
  - Belief: Weights/probabilities/potentials on domain of variable
    - This is the standard used in sum-product algorithm and Bayes nets
  - Survey: Likelihood of warning being required on domain elements
    - More effective than belief propagation
  - *Typical algorithms use only one of these uniformly*

Factor Graph Sum-Product Algorithm

- Computes *marginals* (or MAP/MPE)

$f(x) = \Sigma_{u,w,y,z} f(u,w,x,y,z) = \Sigma_{u,w,y,z} f_1(u,w,x) f_2(x,y,z) f_3(z) = \Sigma_{u,w} f_1(u,w,x) \Sigma_z f_3(z) \Sigma_y f_2(x,y,z)$

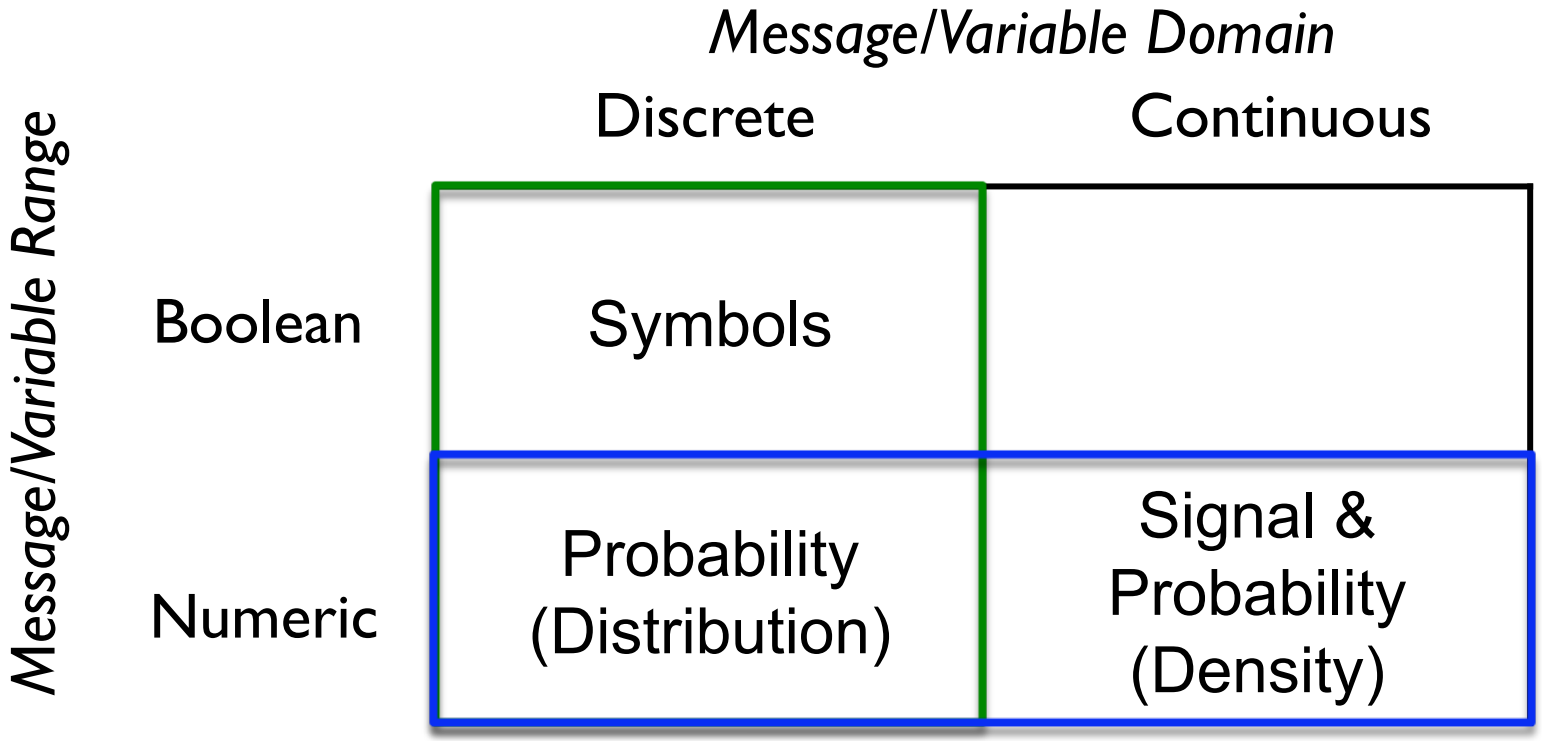
- Message passing about possible values of variables
  - Variable node combines messages from factors to which it is connected (except target factor)
    - Point-wise *product* of values for elements of variable's domain
  - Factor node combines information from all variables to which it is connected (except target variable) plus its own function
    - Does a point-wise *product* as well
    - Also marginalizes out variables not part of target variable node (*sum*)



Properties of Sum-Product

- Applicable to any pair of ops defining a *commutative semi-ring*
  - Both ops are associative, commutative and have identity elements
  - The distributive law is defined: a(b+c) = ab+ac
  - *E.g., +/*, max/*, OR/AND*
- Behaves differently as function of structure of graph
  - Reduces to evaluation of expression trees for *tree-structured graphs* where only care about root
  - Guaranteed to produce correct answer for *polytrees*
    - At most one undirected path between any two vertices
  - For graphs with loops, works well iteratively in many cases, but not guaranteed to produce correct answer
    - May need to add a termination criterion as well
- Tied to statistical mechanics, leading to extensions
  - Minimizes the Bethe free energy

Scope of Sum-Product Algorithm



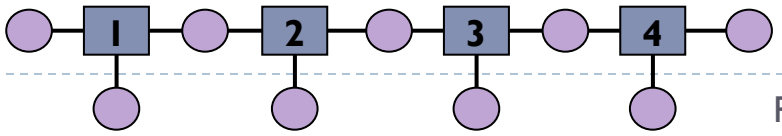
- *Mixed* models combine Boolean and numeric ranges
- *Hybrid* models combine discrete and continuous domains
- *Hybrid mixed* models combine all possibilities
- *Dynamic hybrid mixed* models add a temporal dimension

Research Strategy

- Goals
  - Evaluate extent to which graphical models can provide a uniform implementation layer for existing architectures
  - Develop novel, more functional architectures
    - Enhancing and/or hybridizing existing architectures
    - Starting from scratch leveraging strengths of graphical models
    - *Improving elegance, functionality, extensibility, integrability and maintainability*
- Initial approach
  - Reimplement and enhance the Soar architecture
    - One of the longest standing and most broadly applied architectures
    - Exists in both uniform (Soar ≤8) and diverse (Soar 9) forms
  - Start from the bottom up, implementing uniform version while looking for opportunities to more uniformly incorporate Soar 9's diversity plus critical capabilities beyond all versions of Soar

Published Progress to Date

- *Elaboration cycle* implementation via factor graphs
  - **Production match** and firing
- *Decision cycle* implementation via Alchemy (Markov logic)
  - **Elaboration phase** and decision procedure
- With both also went beyond existing capability
  - *Lower complexity bound* for production match
  - *Mixed* elaboration phase with simple *semantic memory* and *trellises*
- Still preliminary, partial implementations
  - Sufficient to demonstrate initial feasibility
  - Insufficient for full evaluation of uniformity and functionality



Simple Mapping of Production Match onto Factor Graphs

P1: Inherit Color

C1: (<v0> ^type <v1>)

C2: (<v1> ^color <v2>)

-->

A1: (<v0> ^color <v2>)

Model as a Boolean function:

$P_1(v_0,v_1,v_2) = C_1(v_0,v_1)C_2(v_1,v_2)A_1(v_0,v_2)$

WM is 3D Boolean array (*obj* x *att* x *val*)

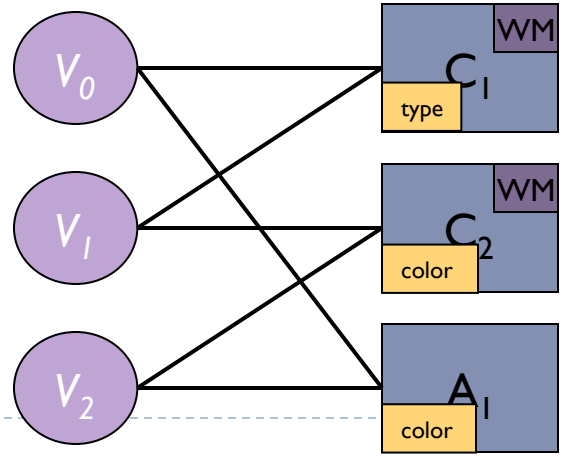
1 when triple in WM

0 otherwise

Messages are Boolean vectors

1 when variable value possible

0 when variable value ruled out



Constant tests hidden in factors

WM is embedded in factors

Confuses binding combinations

May not check if rule completely matches

Move Constant Tests into FG

P1: Inherit Color

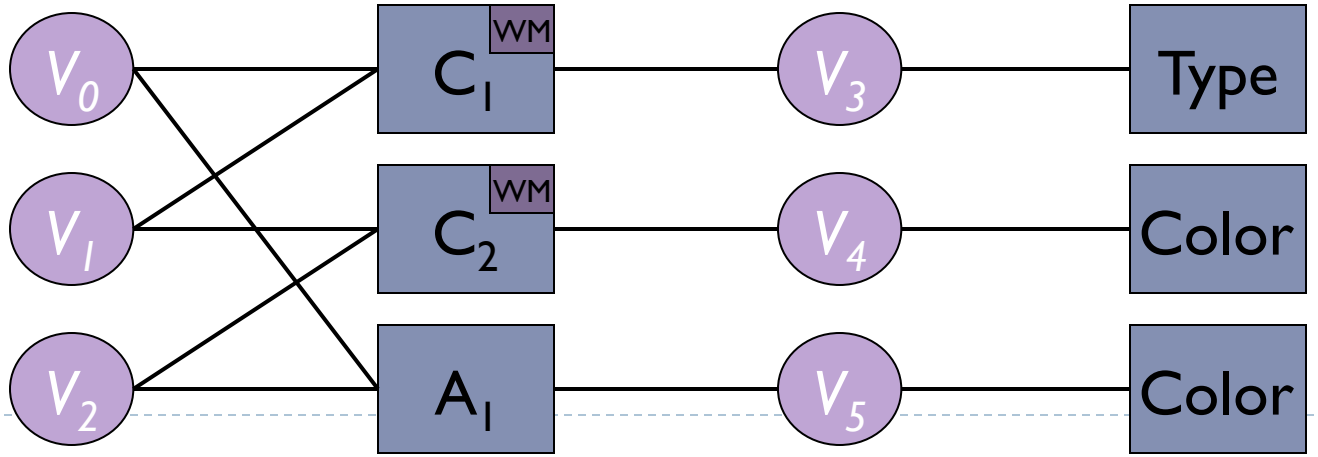
C1: (<v0> ^type <v1>)

C2: (<v1> ^color <v2>)

-->

A1: (<v0> ^color <v2>)

$$P_1(v_0,v_1,v_2,v_3,v_4,v_5) = C_1(v_0,v_1,v_3)C_2(v_1,v_2,v_4)A_1(v_0,v_2,v_5)$$
$$\text{Type}(v_3)\text{Color}(v_4)\text{Color}(v_5)$$



27

Bring WM into Factor Graph

P1: Inherit Color

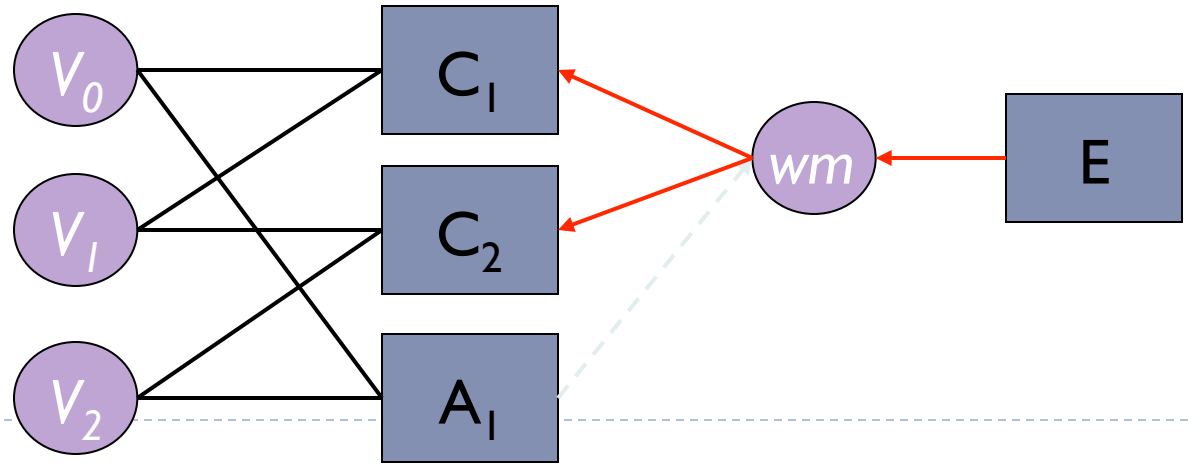
C1: (<v0> ^type <v1>)

C2: (<v1> ^color <v2>)

-->

A1: (<v0> ^color <v2>)

$$P_1(wm,v_0,v_1,v_2) = E(wm)C_1(v_0,v_1,wm)C_2(v_1,v_2,wm)A_1(v_0,v_2,wm)$$



28

Binding Confusion

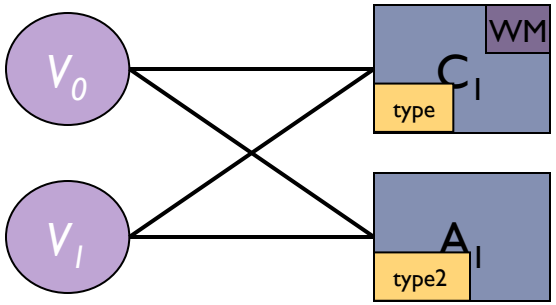
P2: Binding Confusion

C1: (<v0> ^type <v1>)

-->

A1: (<v0> ^type2 <v1>)

$P_2(v_0,v_1) = C_1(v_0,v_1)\, A_1(v_0,v_1)$



Called *instantiationless match* in earlier work

WM:

W1: (A ^type B)

W2: (C ^type D)

Match yields:

v_0 = {A, C}

v_1 = {B, D}

Action yields:

(A ^type2 B)

(A ^type2 D)

(C ^type2 B)

(C ^type2 D)

Binding Confusion Trace

A
B
C
D
type
type2

P2: Binding Confusion

C1: (<v0> ^type <v1>)

-->

A1: (<v0> ^type2 <v1>)

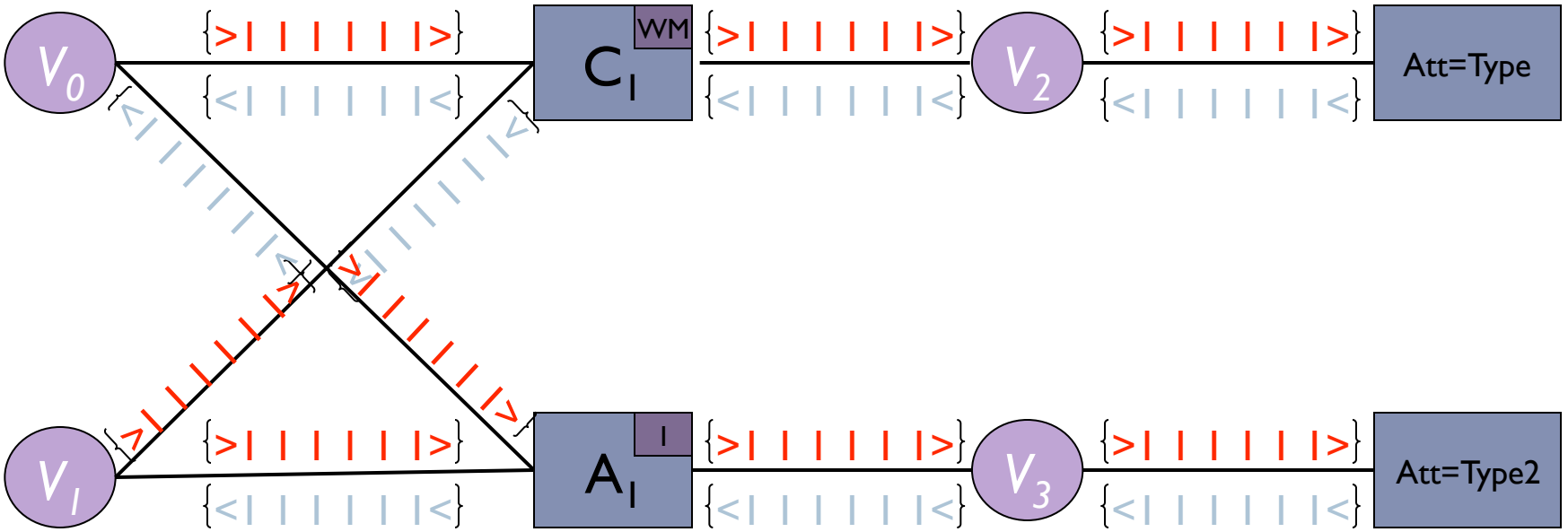
WM:

W1: (A ^type B)

W2: (C ^type D)

[Really 6^3 Boolean array]

Left-to-Right Message Right-to-Left Message



30

Binding Confusion Trace

A
B
C
D
type
type2

P2: Binding Confusion

C1: (<v0> ^type <v1>)

-->

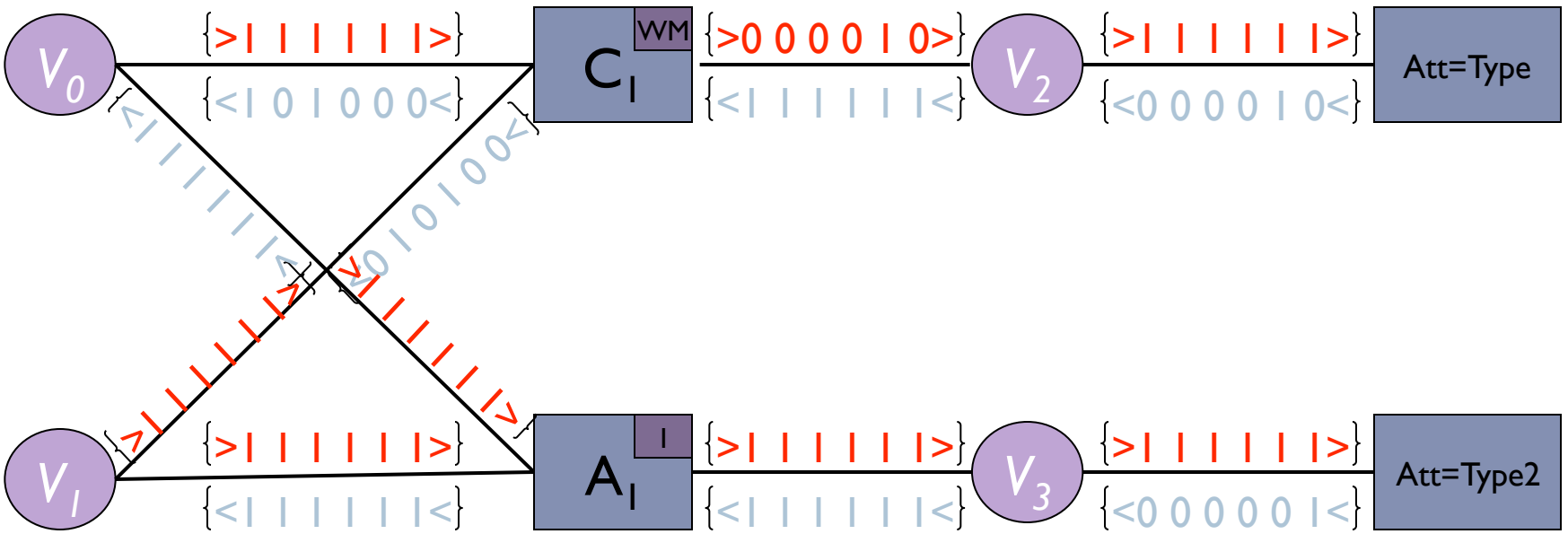
A1: (<v0> ^type2 <v1>)

WM:

W1: (A ^type B)

W2: (C ^type D)

[Really 6^3 Boolean array]



31

Binding Confusion Trace

P2: Binding Confusion

C1: (<v0> ^type <v1>)

-->

A1: (<v0> ^type2 <v1>)

WM:

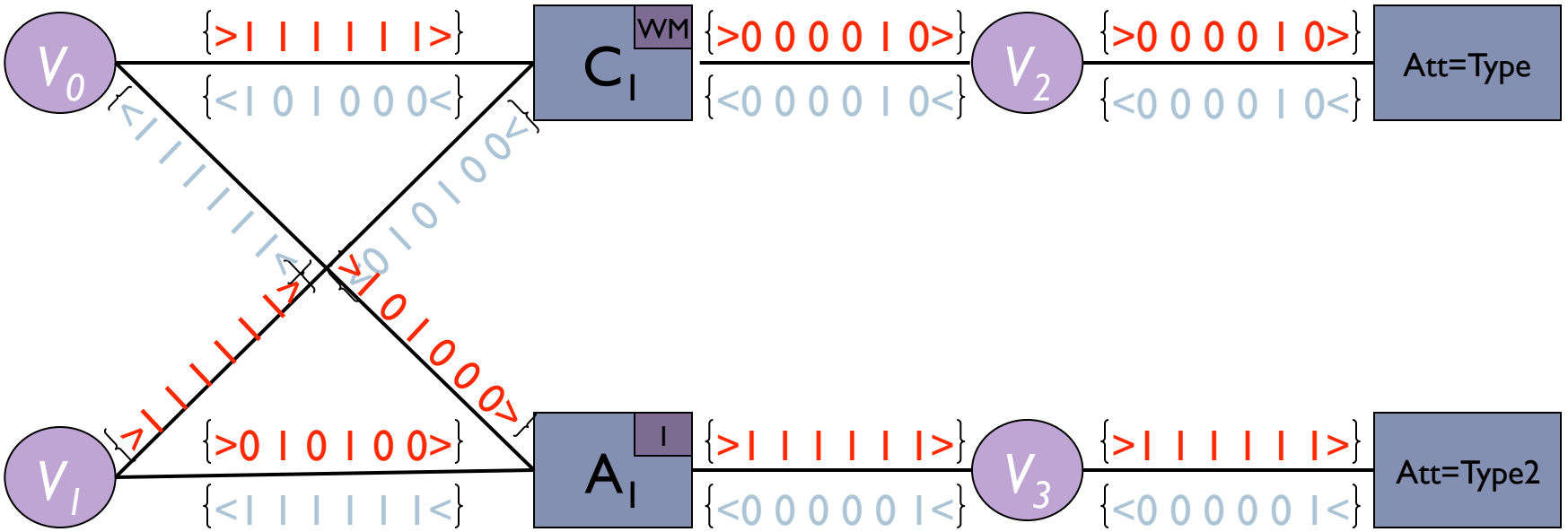
W1: (A ^type B)

W2: (C ^type D)

[Really 6^3 Boolean array]

A
B
C
D
type
type2

Left-to-Right Message Right-to-Left Message



32

Total of 22 Messages in Actual Implementation

Some Possible Solutions

- Alter processing to yield full $f(v_1, v_2, \ldots)$
  - Extract instantiations after generate independent binding sets
  - Alter factor graph to directly generate instantiations (à la Rete)
- Redefine what production match is to achieve
  - Define match to generate what factor graph yields : $f_1(v_1), f_2(v_2), \ldots$
    - Write rules so that this altered semantics is sufficient
  - *Variation*: Divide action *weight* among ambiguous wmes in WM
    - E.g., each new wme set to .25 rather than 1
    - Leverages potential mixed representation to handle ambiguity
  - *Variation*: Compute enough of full function for correct action bindings
    - Eliminate confusion by tracking variable combinations as needed
      - Related to *variable stretching* and *junction trees*
    - Eliminate redundant instantiations that have same action variable bindings and different condition variable bindings

Tracking Variable Combinations

P1: Inherit Color

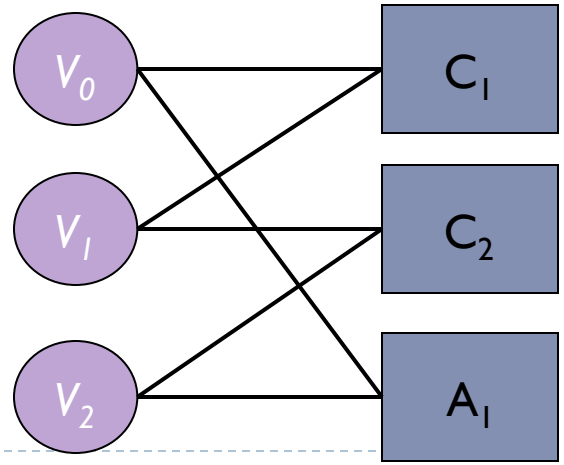
C1: (<v0> ^type <v1>)

C2: (<v1> ^color <v2>)

-->

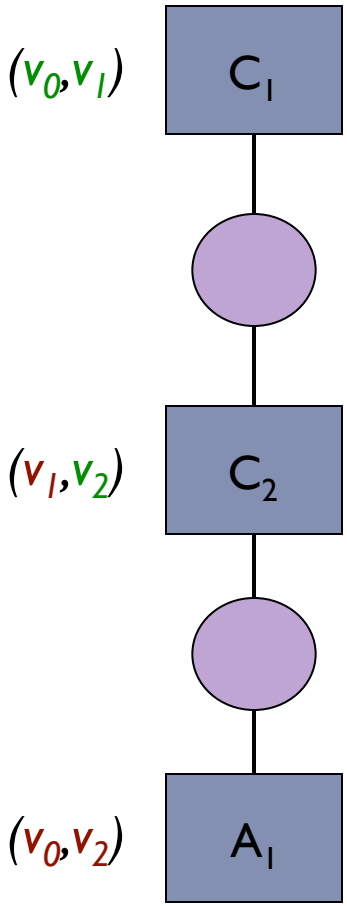
A1: (<v0> ^color <v2>)

$P_1(v_0,v_1,v_2) = C_1(v_0,v_1)C_2(v_1,v_2)A_1(v_0,v_2)$



Tracking Variable Combinations

$P_1(v_0,v_1,v_2) = C_1(v_0,v_1)C_2(v_1,v_2)A_1(v_0,v_2)$



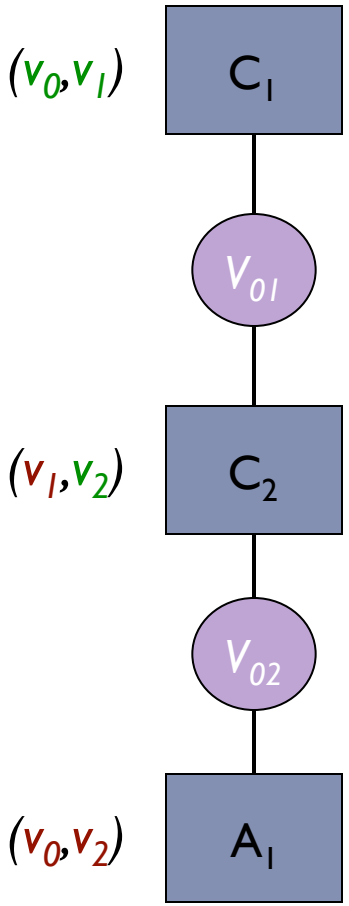
Order factor nodes

Add intervening variable nodes

Compute where each rule variable is first and last used

Tracking Variable Combinations

~~$P_1(v_0,v_1,v_2) = C_1(v_0,v_1)C_2(v_1,v_2)A_1(v_0,v_2)$~~ $P_1(v_0,v_1,v_2) = C_1(V_{01})C_2(V_{01},V_{02})A_1(V_{02})$



- Order factor nodes
- Add intervening variable nodes
- Compute where each rule variable is first and last used
- *Stretch* rule variables over all nodes between first and last usage
- Complexity bounded by max rule variables at a node (*treewidth*)
  - Versus number of conditions in Rete
- The approach works, but increases the combinatorics over the simple mapping

Ensuring Rule Matches

- The stretched factor graph generates appropriate bindings for action variables, but doesn't guarantee all of the conditions actually match working memory
  - E.g., a condition that is all constants may not match, yet all of the variables in the actions will be instantiated from other conditions that do successfully match
  - Messages communicate information about variable values rather than acting as triggers, as instantiations do in Rete
- Add a new production variable in every condition factor to compute product of all condition matches
  - Not actually needed in Soar because of guaranteed linkage among variables, but may be needed in general

37

Matching Rule Example

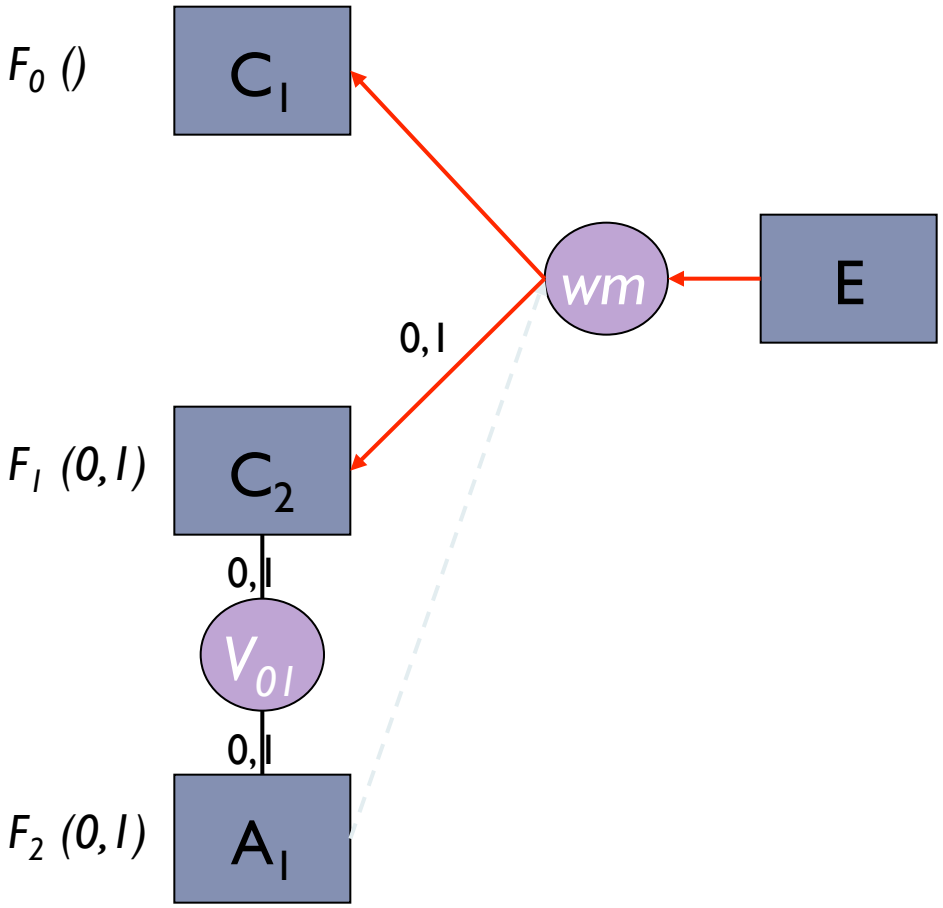
P1: Inherit Color

C1: (X ^color Y)

C2: (<v0> ^type <v1>)

-->

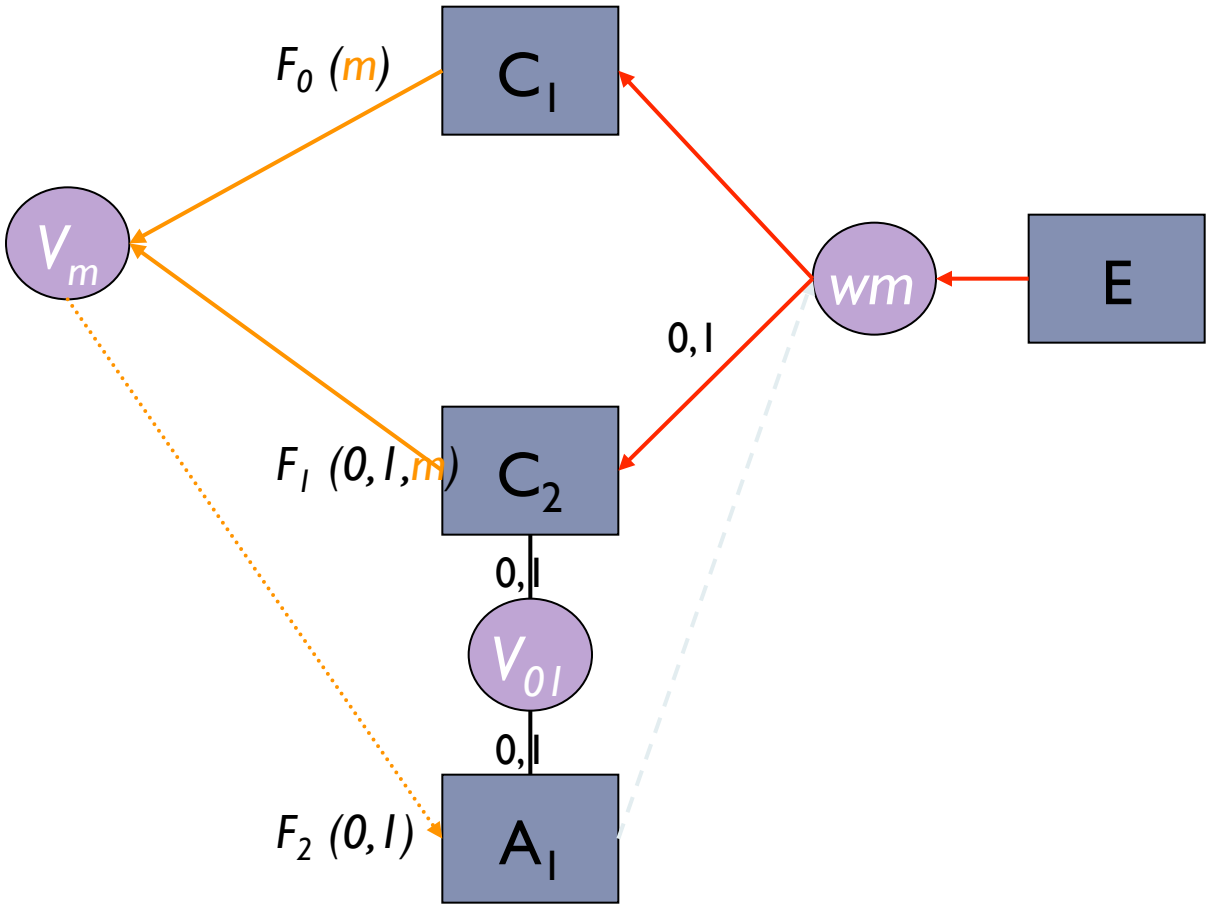
A1: (<v0> ^color <v1>)



$P_1(wm,v_0,v_1) = E(wm)C_1(wm)C_2(v_0,v_1,wm)A_1(v_0,v_1,wm)$

38

Matching Rule Example



$P_1(wm,v_0,v_1,v_m) = E(wm)C_1(v_m,wm)C_2(v_0,v_1,v_m,wm)A_1(v_0,v_1,v_m,wm)$

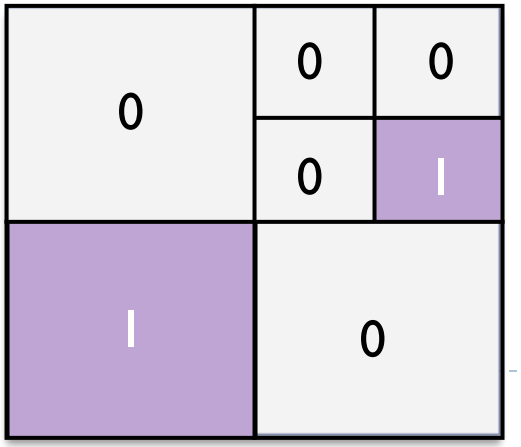
39

Message (& WM) Size Problem

- Solutions to binding confusion and rule matching increase number of rule variables processed at variable nodes
  - Yields exponential growth in message size and processing cost
- WM may itself be quite large: *max-symbols*^{3}
- Need to take advantage of the tendency towards uniform values in order to reduce size and processing cost
  - Both WM and messages are nearly all 0 or 1
- *Solution*: A hierarchical approach to nested region specification

Hierarchical Memories and Messages

- N dimensional generalization of quad/octrees (*exptrees*)
  - If entire space has one value, just assign it to the region
  - Otherwise partition space into 2^N regions at next level, and recur
- Views WM (and messages) as piece-wise constant functions
  - Could also conceivably be extended to piecewise linear functions
  - Natural compact representation for probabilities, signals, images, etc.
- Could alternatively do more adaptive partitioning that clusters/ reorders points into regions based on patterns in the data



Example Match Times

P1: Inherit Color

C1: (<v0> ^type <v1>)

C2: (<v1> ^color <v2>)

-->

A1: (<v0> ^color <v2>)

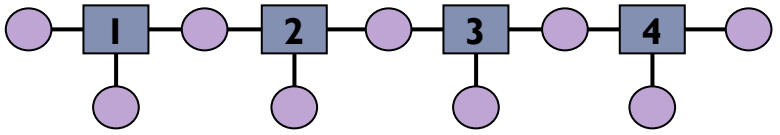
With solutions to all four problems, rule graph comprises 8 factor nodes and 8 variable nodes.

WM is 16^3 in size, with 4 wmes

	Product before Sum	**Redistribute P over S**	
Arrays	*Exceeded heap space*	1.7 sec.	~7
Hierarchies	132 sec.	.25 sec.	
	~500		

Implementing Soar's Elaboration Phase via Alchemy (Markov logic)

- Markov logic = First order logic + Markov networks
  - Node for each ground predicate
  - Weight for each ground clause (clique potentials)
    - Along with links among all nodes in ground clause
- Goals for implementation
  - Explore a *mixed* elaboration phase (rules & probabilities)
  - Enable bidirectional message flow across rules
    - Normal elaboration cycle only propagates information forward
    - But need bidirectional settling across elaboration cycles and *trellises*
  - Explore semantic (fact) memory and trellises



Encoding

- Convert productions into logical implications
  - Define types for objects and values of triples
    - colors={Red, Blue, Green} and objects = {A, B, C, D, E, F}
  - Define predicates for attributes
    - Color(objects, colors) and Type(objects, objects)
  - Specify implications/clauses for rules
    - (Type(v0, v1) ^ Color(v1, v2)) => Color(v0, v2).
  - Add weights to clauses as appropriate

P1: Inherit Color

C1: (<v0> ^type <v1>)

C2: (<v1> ^color <v2>)

-->

A1: (<v0> ^color <v2>)

- Initialize database file with WM
  - Color(C, Red), Color(D, Blue), Type(A, C), Type(B, D)
- Semantic memory: weighted ground predicates: 10 Color(F, Green)
- Trellis: define via a pair of implications (*accept* & *reject* prefs.)
  - Size(step, size) => Size(step+1, size*2).
  - (Size(step, size1) ^ size1!=size2) => !Size(step, size2).

Results

- Mapping basically works (modulo trellis strangeness)
  - Mixed representation with simple semantic memory and trellises
- Match occurs via graph compilation not message propagation
  - As Alchemy compiles first-order clauses to ground network
    - All symbolic reasoning in compilation and probabilistic in propagation?
  - Falls short of uniform processing in the graph itself
- Use of exptrees is analogous to Alchemy's laziness and lifting
  - Deal with both default values and elements that can be processed in the same way
  - Exptrees may be cruder, but extension to piecewise linear functions is intriguing for dealing with continuous quantities

Speculations from Mapping to System Levels

- If Alchemy maps onto Soar's elaboration phase – and ultimately its full decision cycle – then:
  - Soar's decision cycle needs to be expanded to three phases
    1. Compile/match to generate a ground/instantiated network
    2. Perform probabilistic inference in the ground network
    3. Decide
  - Perhaps systems like Alchemy should not seek global minima
    - Elaboration Cycle (10 ms): Local propagation of information
    - Decision Cycle (100 ms): Global propagation but local minima
    - Problem Space Search (≥ 1 sec): Global minima (via sequence of locals)
  - Global processing should not happen within the elaboration cycle
    - Creating unique identifiers in rules, non-monotonic processing, etc.
    - Raises questions about extent to which WM should be global as well

Conclusion

- Despite increasing diversity within cognitive architectures, there is still hope for uniformity in the cognitive hierarchy
  - The narrow waist in the cognitive hourglass
- Uniformity at the implementation level should facilitate:
  - Exploring, understanding and improving existing architectures
  - Developing novel architectures with increased elegance and functionality
- Factor graphs (and graphical models in general) promise to yield a wide diversity of capabilities in a uniform manner
  - Mixed, hybrid and neural processing
  - Procedural and declarative (semantic/episodic) memories
- Initial progress made, but much more still to be done
  - Complete reimplementation and extension of Soar
  - Reexamine other cognitive architectures and hybrids among them
  - Experiment with radically new architectures enabled by graphical models