Factor Graphs

Take Three…

Paul S. Rosenbloom

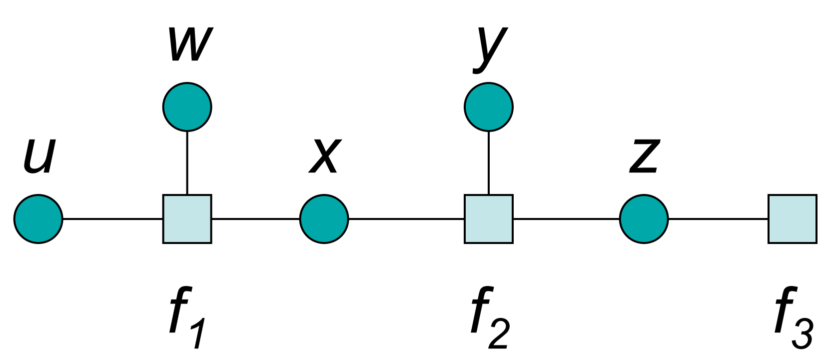
11/13/2008

Factor Graph Background

- Tame combinatorics in many calculations
  - Decoding codes (origin of factor graphs)
  - Bayesian networks and Markov random fields
  - HMMs (and signal processing more generally, including vision)
  - Constraint propagation
  - *Production match?*
- Form of divide-and-conquer with *nearly decomposable* components
- Many standard state-of-the-art algorithms can be derived from factor graph *sum-product algorithm*
  - Belief propagation in Bayesian networks
  - Forward-backward algorithm in HMMs
  - Kalman filter, Viterbi algorithm, FFT, turbo decoding
  - Equivalent to distributed arc-consistency in constraint diagrams
  - Overlaps with dynamic programming?
  - *Rete algorithm (or equivalent)?*

Structure of 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



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*

Sum(mary)-Product Algorithm

- A variable node combines the messages from all of the factors to which it is connected (except for one to which message is to be sent)
  - Typically by point-wise multiplication of probabilities/potentials for each element of variable's domain (*product*)

- A factor node combines information from all of the variable nodes to which it is connected (except for one to which message is to be sent) plus its own function
  - Also does a point-wise product, but in addition must marginalize out all of the variables not corresponding to the variable node to which the message is to be sent (*sum(mary)*)
  - *Although I'm still frankly a bit puzzled about what point-wise product really means when the messages into a factor node each represent different variables*
    - *I currently handle this by combining message with factor function first*
    - *Seems to do the right thing, but not clear what is really intended*

Properties of Algorithm

- Applicable to any pair of operations defining a *commutative semi-ring*
  - Both operations are associative and commutative
  - Both operations 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 *trees* in which only care about value of root variable
  - 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, which leads to extensions
  - Minimizes the Bethe free energy

Scope of FG and BP

Type of Variable Domains (columns) / *Form of Messages* (rows)

Form of Messages	Discrete	Continuous
Boolean	Symbols	
Numeric	Probability (Distribution)	Signal & Probability (Density)

- *Mixed* models combine Boolean and numeric
  - For example, constraints and Bayesian networks
- *Hybrid* models combine discrete and continuous
- *Hybrid mixed* models combine all possibilities
- *Dynamic hybrid mixed* models add in a further temporal dimension

Factor Graphs for Cognitive Architecture

- There is work on hybrid mixed models – although still very little and quite preliminary – but no one so far has looked at implications for cognitive architecture
- Idea/Hope/Fantasy: Factor graphs will provide a uniform layer for implementing and exploring cognitive architectures, while also pointing to novel architectures that are more uniform, integrated and functional
  - Yielding a better understanding of architectures and their modules/processes
  - Yielding hybrid mixed models that provide uniform integration
  - Potentially combining sequential and static reasoning
    - Dynamic hybrid mixed model
  - Generalizing STORM module interface approach
    - Rather than just setting interface variables (*warnings*), can send more subtle messages about their values (*beliefs*, *surveys*)

A Research Strategy

- Reimplement existing architectures via factor graphs
  – To help understand factor graphs and existing architectures
- Look to go beyond existing architectures by hybridization and simplification both within and across existing architectures
- Integrate in new capabilities that don't fit well into existing architectures
  – E.g., vision and speech
- I have started with Soar, and its production system
  – Something I understand very well, which helps in learning the intricacies of factor graphs
  – Need to extend to full elaboration phase, decision procedure, learning, (declarative memory, episodic memory,) etc.
  – Need to bring in probabilities and signals

Factor Graphs for Production Systems

- Provides a space of alternative match algorithms
  - Vary in power and complexity
- Points in directions of (symbolic) extensions beyond simply forward chaining of rules
  - Backward chaining
  - Abduction
  - Constraint satisfaction
  - Analogy(?)
  - Combinations of approaches
- Provides insight in thinking about mixed models

Production System
(with some added syntactic sugar)

P1: Inherit Color

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

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

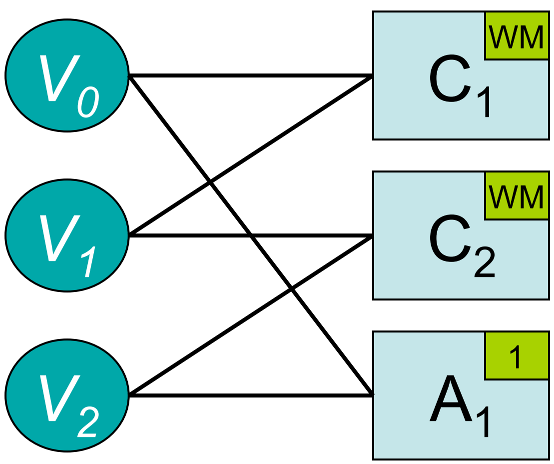
-->

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

WM is a 3D Boolean array
- 1 when triple in WM
- 0 otherwise

Messages are Boolean vectors
- 1 when value possible
- 0 when value ruled out

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



Constant tests are hidden

WM (and goal constraints) are embedded in factors

Only computes marginals of variables independently

Move Constant Tests into FG

(Implemented Production System)

P1: Inherit Color

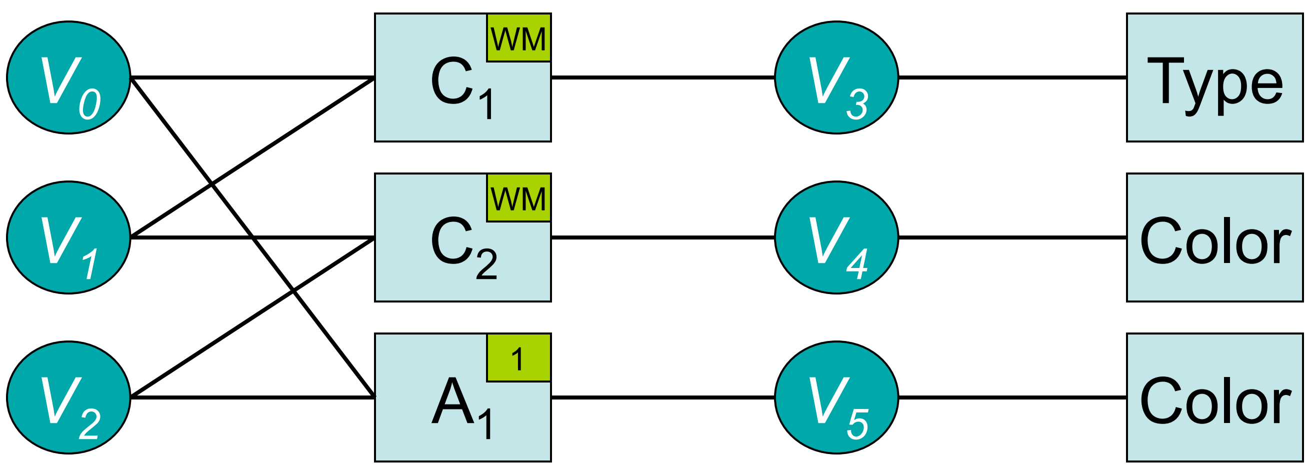
C1: (<v0> ^type <v1>)

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

-->

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

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



Bring WM(/Goals) into FG

P1: Inherit Color

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

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

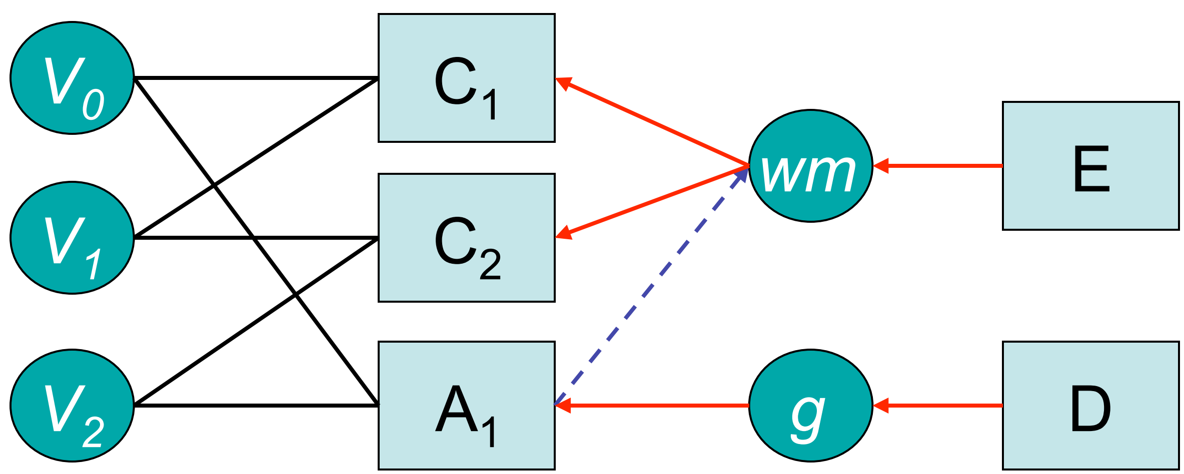
-->

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

Solid arrows indicate no messages in other direction. Just computing deterministic functions of fixed evidence.

Dashed arrow says to change WM on next cycle*

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



Problem: Independent Variable Marginals

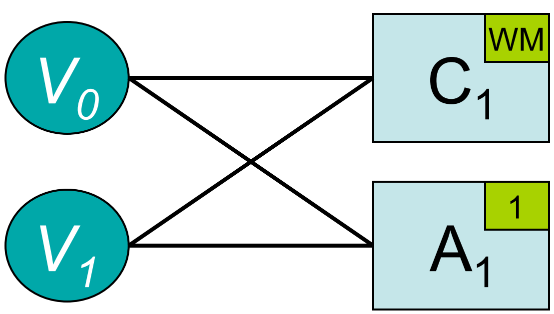
P2: Binding Confusion

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

-->

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

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



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)

Called *instantiationless match* in earlier work

Binding Confusion Trace

{ A B C D type type2 }

P2: Binding Confusion

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

-->

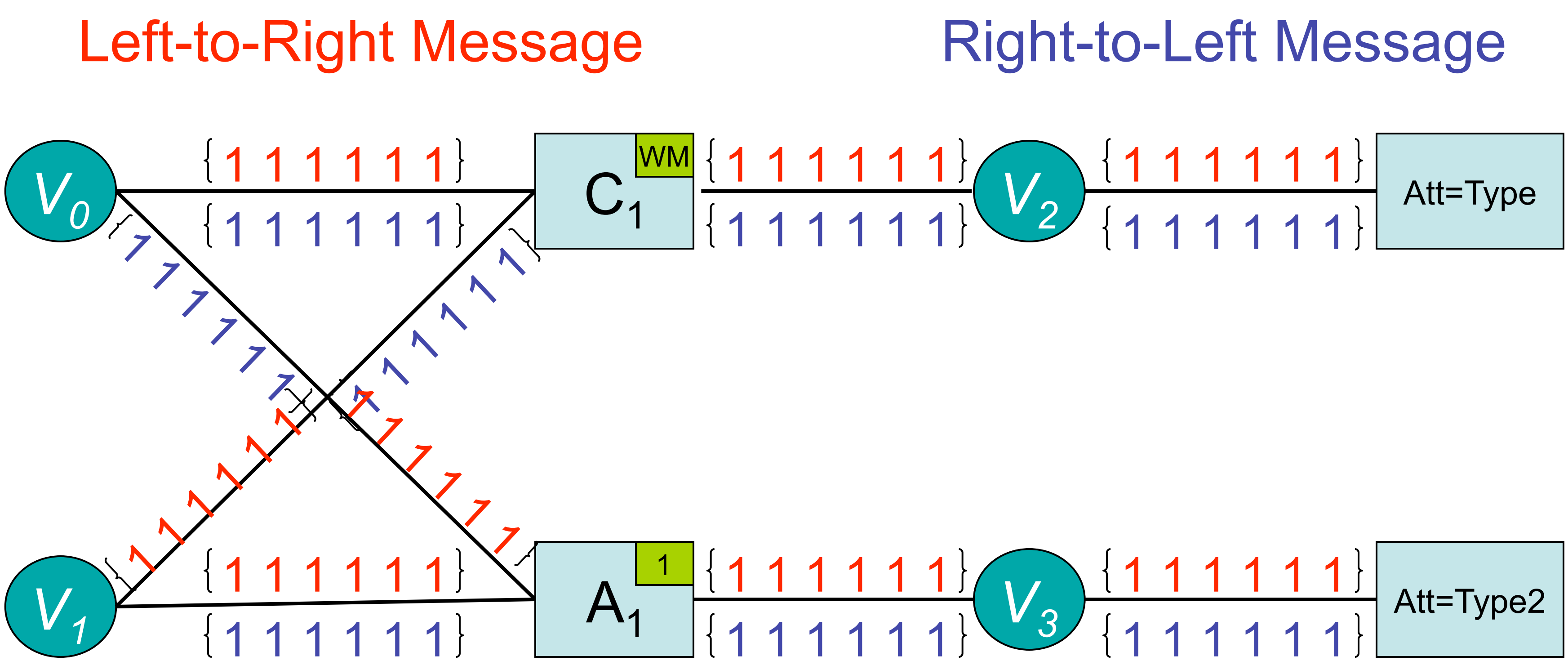
A1: (<v0> ^type2 <v1>)

WM:

W1: (A ^type B)

W2: (C ^type D)

[Really 6^3 Boolean array]



Binding Confusion Trace

P2: Binding Confusion

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

-->

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

WM:

W1: (A ^type B)

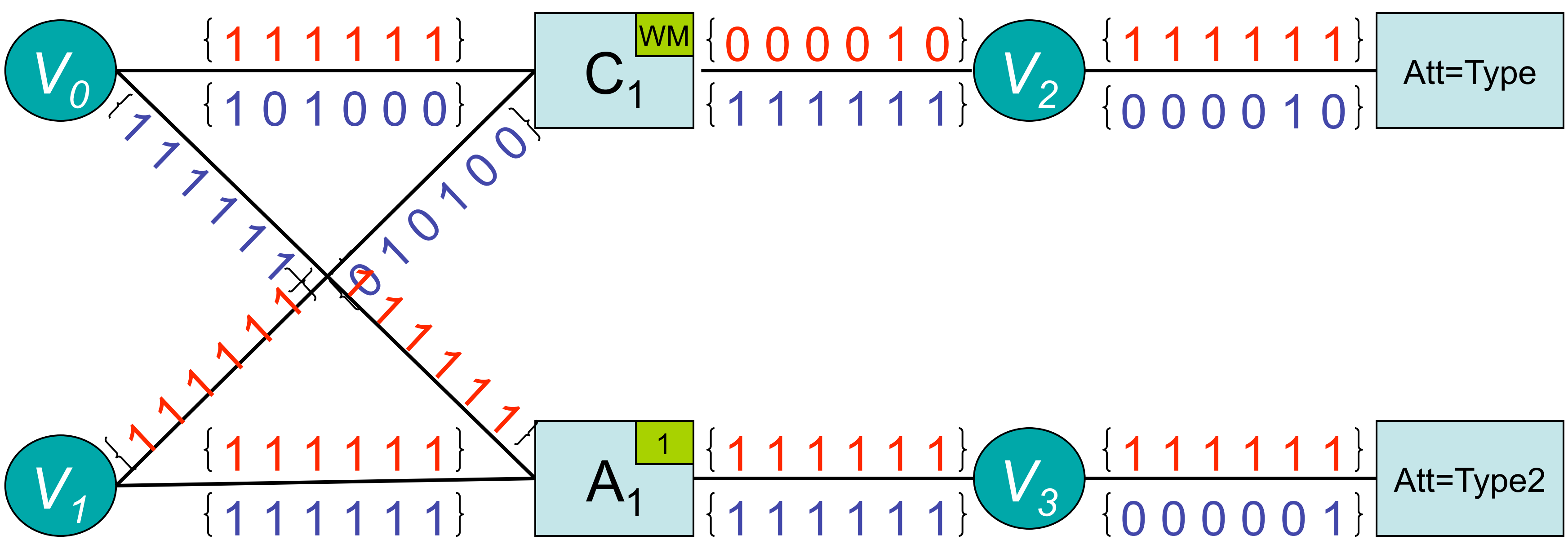
W2: (C ^type D)

[Really 6^3 Boolean array]

{ A, B, C, D, type, type2 }

Left-to-Right Message

Right-to-Left Message



Binding Confusion Trace

P2: Binding Confusion

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

-->

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

WM:

W1: (A ^type B)

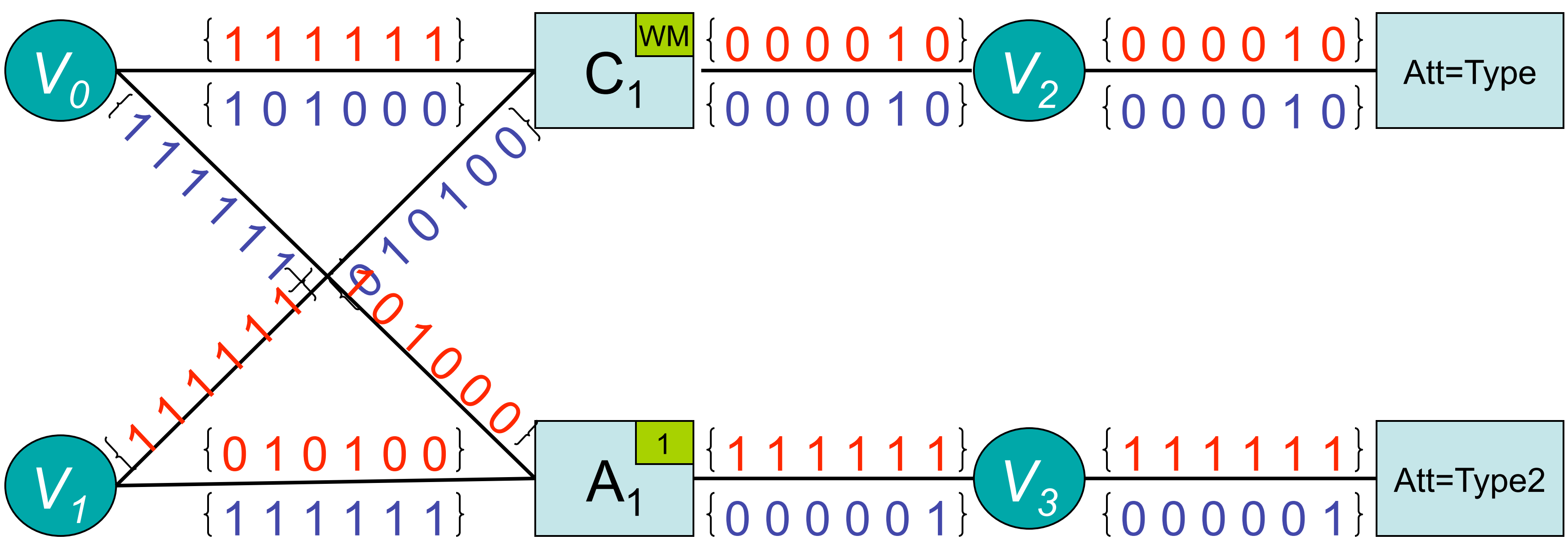
W2: (C ^type D)

[Really 6^3 Boolean array]

{ A, B, C, D, type, type2 }

Left-to-Right Message

Right-to-Left Message



Total of 22 Messages in Actual Implementation

Some Possible Solutions

- Alter processing in some way to yield full $f(v_1, v_2, \ldots)$
  - Extract instantiations after generate independent binding sets
  - <u>Alter factor graph to directly generate instantiations (à la Rete)</u>
- Redefine what production match is to achieve
  - Define match to generate what factor graph yields : $f_1(v_1)$, $\mathrm{f}_2(v_2)$, …
  - 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
  - <u>Compute marginalizations for groups of variables</u>
    - For action (<v0> ^type2 <v1>), compute $f_{12}(v_1,v_2)$, not $f_1(v_1)$ and $\mathrm{f}_2(v_2)$
      - Use factor graph transformation of *clustering* to combine variables
      - Or just do the summing out differently?
    - For other variables can still yield $f_i(v_i)$, or not generate marginal at all
      - Potential to significantly reduce complexity with respect to Rete?
- Other approaches?

Rete Match

P1: Inherit Color

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

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

-->

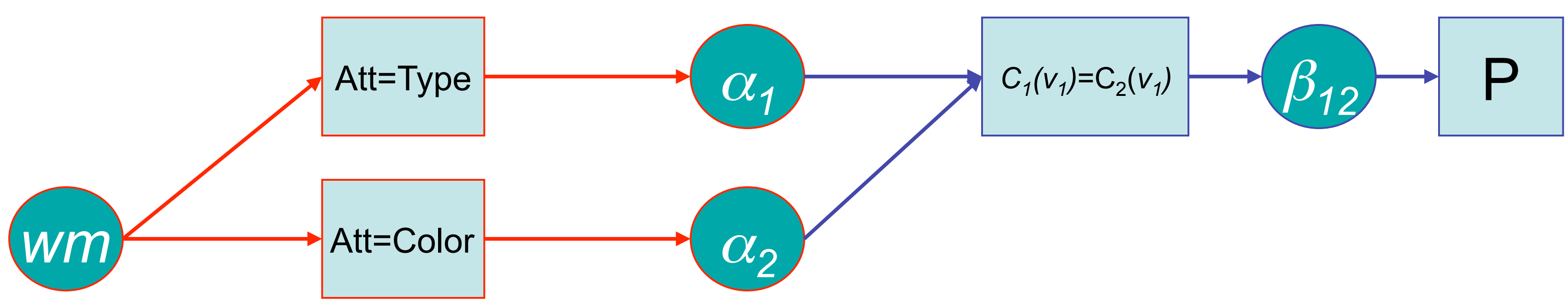
A1: (<v0> ^color <v2>)

Alpha network

- Discrimination network on constant tests
- Fills α memories w/ wmes matching conditions

Beta network

- Join network on variable binding tests
- Fills β memories with partial instantiations



Rete in Factor Graphs

P1: Inherit Color

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

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

-->

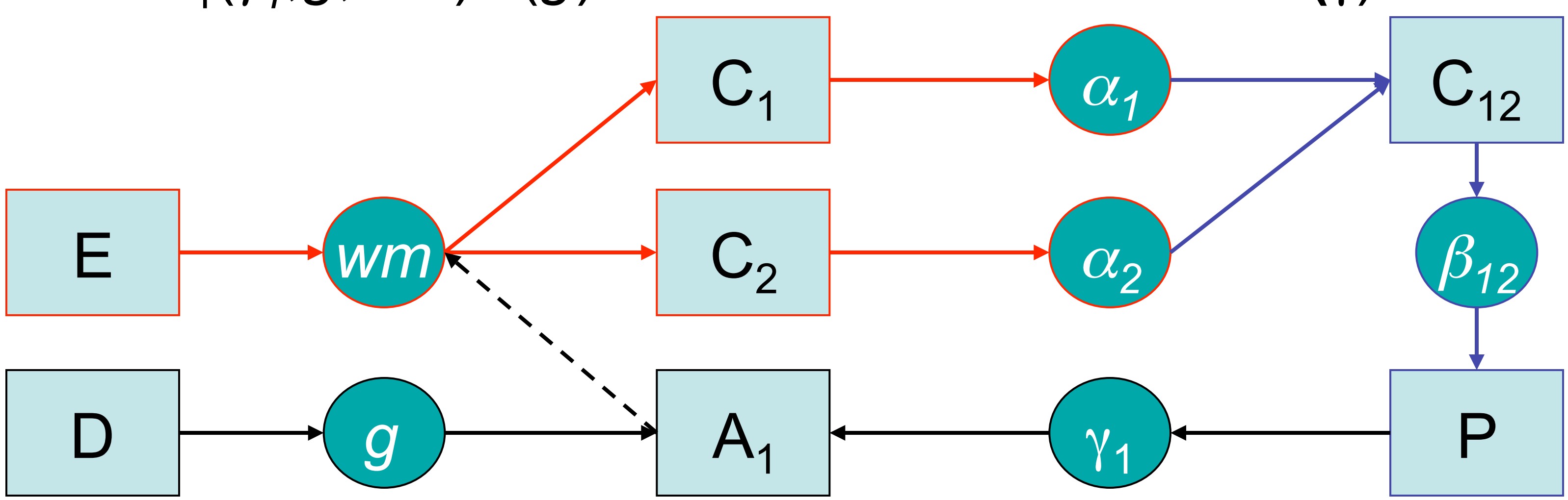
A1: (<v0> ^color <v2>)

$$P_1(wm, v_0, v_1, v_2, g) = E(wm)C_1(wm,\alpha_1)C_2(wm,\alpha_2)\,C_{12}(\alpha_1,\alpha_2,\beta_{12})P(\beta_{12},\gamma_1)\,A_1(\gamma_1,g,wm)D(g)$$

α network

β network

Action (γ) network



Comments on Rete in FG

- Combines Rete's α and β networks, plus actions (γ), into a single graph structure processed in a uniform manner
- Graph can be evaluated as an expression tree
  - Can think of duplicating unchanging WM across C nodes to make tree
  - Guaranteed solution with no iteration required, as with Rete
- Size of WM domain is n^3 for n max symbols and size of β message is n^{3k} for k conditions
- Need to use sparse or hierarchical message structures
  - Sparse messages just include elements that are 1 (instantiations)
  - Hierarchical structure does nested array-region specification
    - E.g., start with stating whole array is 0 and then specify which subregions are 1
      - Can also then nest this further with exceptions that are 0 for subsubregions, etc.
    - Generalization over standard/Rete sparse/instantiation-based representation
      - More efficient when instantiations are clustered in array
    - Also transfers better to mixed case (used in some Bayes net approaches)
- Should be extensible to sharing of work across rules and cycles

Clustering Action Variables

P1: Inherit Color

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

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

-->

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

P2: Binding Confusion

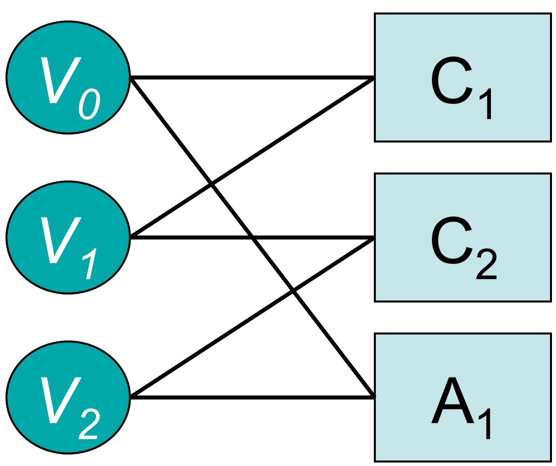
C1: (<v0> ^type <v1>)

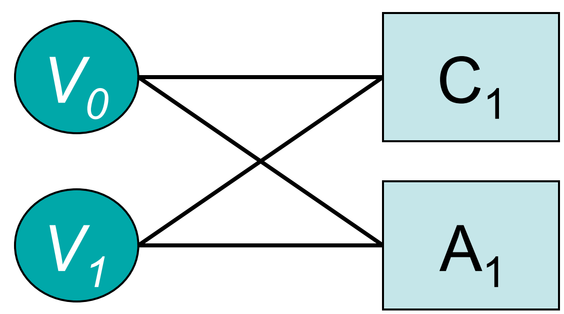
-->

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

$$P_1(v_0,v_1) = C_1(v_0,v_1)\,A_1(v_0,v_1)$$

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





Clustering Action Variables

P1: Inherit Color

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

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

-->

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

P2: Binding Confusion

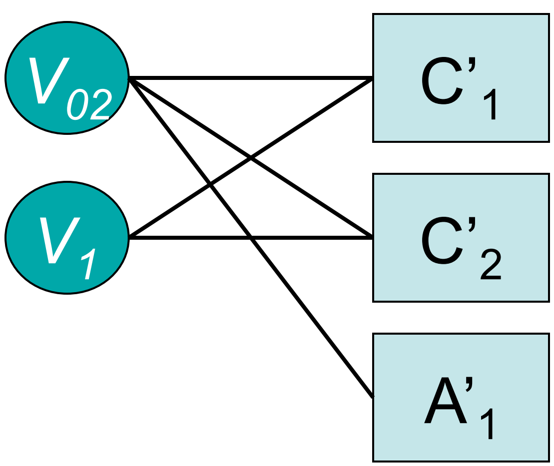
C1: (<v0> ^type <v1>)

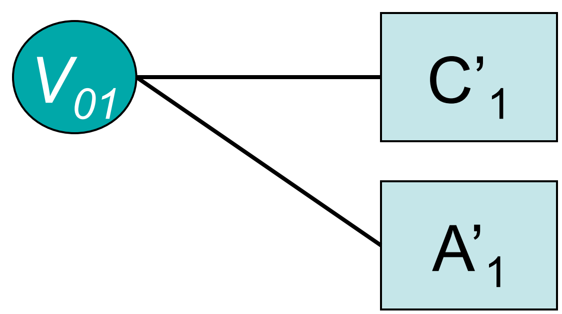
-->

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

$$P_2(v_0,v_1) = C_1(v_0,v_1)\,A_1(v_0,v_1)$$
$$= C'_1(v_{01})\,A'_1(v_{01})$$

$$P_1(v_0,v_1,v_2) = C_1(v_0,v_1)C_2(v_1,v_2)A_1(v_0,v_2)$$
$$= C'_1(v_{02},v_1)C'_2(v_{02},v_1)A'_1(v_{02})$$





Only introduces combinatorics where needed

Beyond Forward Chaining

- Use rule definitions in new ways
  - Backward chaining
    - Use Goal array to constrain bindings of action variables to what want and thus indirectly constrain bindings of condition variables
    - Propagate constraint backwards by adding to goal array wmes that will enable rules to fire in forward direction
  - Hybrid forward/backward chaining
    - Unconstrained goal array yields forward chaining
    - Tightly constrained goal array yields backward chaining
    - Moderately constrained goal array yields some kind of mixed behavior
  - Abduction
    - Allow backward chaining to change WM at select times
- Combine rule definitions with other symbolic graphs
  - Constraints are fully symmetric graphs
    - But what would/should they actually mean/do as part of LTM?
  - Facts and examples for declarative memory and analogy?
    - Will actually talk a bit more about declarative under mixed
  - Others?

Bayesian Network (BN) Example

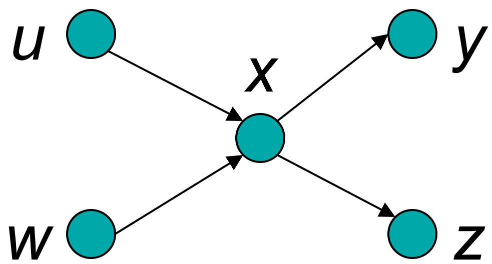
- Probabilistic reasoning involves computations of various quantities from *joint probability distributions* over random variables
  - E.g., compute the *marginal probability* $p(u)$ from the joint probability distribution $p(u,w,x,y,z)$ by summing out/over all of the other variables
    - $p(u) = \mathrm{SUM}_{w,x,y,z}\,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

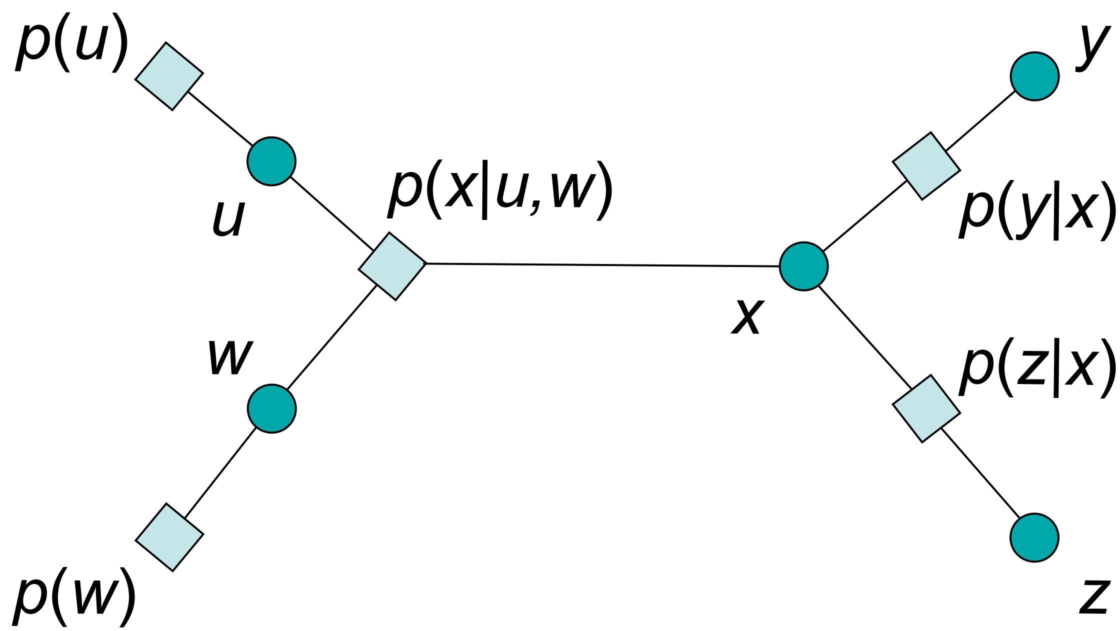
BN Example (cont.)

- 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

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





Towards Mixed Graphs

- Existing work combines constraints and probabilities
  - Essentially hard and soft constraints
    - Hard constraints involve probabilities of 0 and 1 only
  - Standard BP in Bayesian networks works for such mixtures, but you can increase efficiency by preprocessing hard constraints
- What would mixtures do in our case?
  - Extend WM to be prior distribution on contents of WM
    - Move from Boolean to numeric values in array
  - Extend rules to be conditional probabilities (CPs)?
    - CP of wmes generatable by actions given wmes bound to conditions?
      - But may not provide full parents (in Bayes net sense) if other rules can generate same wmes
    - What is connectivity among these CPs in Bayes net sense?
      - Only a part of the domain of an action corresponds to a part of a domain of another condition
      - Can we represent whole elaboration phase as a single Bayes net (trellis)?
  - Probabilities of rules being valid/accurate?
    - Meta-information on rules rather than, or in addition to, being CPs
  - What else can be supported?
    - E.g., other probabilistic information, decision-theoretic decision making, statistical learning, declarative memory, dialogue management?

Thoughts on Declarative Memory (DM)

- Given memories and a probe, retrieve memory best matching probe
- Symbolic/Prolog view of DM
  - A fact is a horn clause with a true body: Mortal(Paul) :- T.
  - Retrieve by stating goal of what is to be known: Mortal(x)
  - Retrieves all exact matches rather than best partial match
  - Can incorporate other knowledge into retrieval by generalizing to full backward chaining on rules
- Probabilistic view of DM
  - Retrieve most probable memory given probe: $\text{Argmax}_m\ P(M|\textit{probe})$
  - Can conceive of probe as corrupted version of a memory, with the problem being to figure out exactly which memory
  - Probe can equivalently be thought of as output of a communication channel, as in coding theory, with the task being to determine input
    - $\text{Argmax}_m\ P(M|\textit{probe}) = \text{Argmax}_m\ \alpha P(\textit{probe}|M)P(M)$
    - Models a priori likelihood of memories, P(M), as well as process of generating probes from memories, $P(\textit{probe}|M)$
- Can approaches be fused if map P(M) to fact memory and $P(\textit{probe}|M)$ to (backward) rules, and then use factor graphs to compute Argmax?
  - Akin to abduction to best explanation for probe?