A Multilevel Bayesian Framework for the Elaboration Phase

[An Internal Document for CAWG Discussion ***]***

Paul S. Rosenbloom

Draft 9 of November 9, 2009

I. At the top level consider the elaboration phase as yielding an interpretation of the current situation given what is known.
   A. A Bayesian perspective is that you want to compute *posterior probabilities* over a set of situation variables (S) given a set of evidence variables (E): P(S|E).
      1. This can be done directly via specification of P(S|E) or indirectly via Bayes Law, as P(E|S)P(S)/P(E).
         a. P(E|S) is called the *likelihood*, as it specifies the probability of the evidence given the situation.
         b. P(S) is the *prior probability* of the situation variables; i.e., their probability before attending to the evidence.
         c. P(E) is the probability of the *evidence*, and is considered as a *normalization constant* since it doesn't depend on the situation.
            i. So dropping the constant yields P(E|S)P(S).
      2. Can view these two distinct ways of specifying the function, directly or indirectly, as templates for the knowledge to be used in computing the posterior.
         a. The direct approach, using P(S|E), corresponds to a *procedural* encoding of knowledge that directly derives situation variables from evidence variables. Classical rules/productions are of this form.
         b. The indirect approach, using P(E|S)P(S), corresponds to a *declarative* encoding of knowledge, such as is provided by a *category* definition.
            i. For example, in Anderson's 1990 work on rational analysis (*The Adaptive Character of Thought*), he derives a model of categorization – the ability to determine the category of an object and to predict its other attributes from a partial specification of its attributes – that is based on the prior for the category, P(C), and the likelihoods of each of the individual attributes given the category: $P(A_i|C)$.
               a. This actually turns out to be naïve Bayes, although it doesn't appear to have been recognized at the time.
         c. Also maps onto distinction between discriminative (direct/procedural) and generative (indirect/declarative) learning?
      3. This suggests it might be reasonable to limit the expressibility of memory to a simple-yet-general conditional form that is sufficient to represent priors, likelihoods, and posteriors.

    a. Or, in other words, a single general form that covers both procedural and declarative memory.
    b. Rather than a full probabilistic logic, such as Markov logic, or even "whatever forms of knowledge with which you can efficiently/boundedly compute in graphs."

B. But it isn't logically necessary to limit these notions to their strictly probabilistic interpretations. In general, can think, at least metaphorically, of the elaboration phase as computing some form of a posterior interpretation of a situation based on generalized notions of partial posteriors (e.g., rules), priors, likelihoods, and evidence.

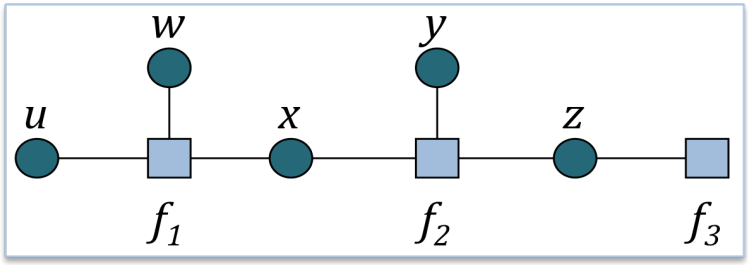
C. Implementation of this idea currently involves three levels rather than the two initially imagined (i.e., the *implementation* and *architecture* levels). From the bottom up, these are:
    1. *Graph*: Summary-product algorithm over continuous, N dimensional, region-based, arrays
    2. *Prœscript*: Expressions that jointly specify the overall function to be computed at the graph level
        a. Assumes a structuring of the graph into a working memory (WM) of ground instances, providing the *evidence,* and a long-term memory (LTM) of (variablized/first-order) *prœscripts*.
            i. A prœscript is a hybrid between a rule (a *prescript*) and a constraint (a *proscript*), with probabilistic annotation.
            ii. WM/evidence is compiled into factor node functions at the graph level and prœscripts are compiled into additional variable and factor nodes.
    3. *Faculty*: Architectural "capabilities" – such as procedural, semantic and episodic memories – based on appropriate varieties of knowledge (and settings of SUM versus MAX?).
        a. This level is not strictly needed for a Bayesian elaboration phase, but starts to get at how the elaboration phase is exploited to yield a range of architectural capabilities.

D. Graphical models are providing an interesting combination of *breadth of applicability* with *constraint on how the breadth is realized*.
    1. The more constraint there is, the more this effort takes the form of the development of a particular architecture as opposed to the creation of a general/neutral implementation level for architectures.
    2. My current approach is to explore a range of architectural capabilities while heading in the general direction of a hybrid (freely mixing cognitive/symbolic and perceptual-motor/signal processing within the inner loop), mixed (symbolic reasoning under uncertainty) variant of Soar 9.
        a. Possibly with additional capabilities when feasible, such as MDPs.

II. The *graph* level

A. Based on factor graphs over continuous variables.

1. Consisting of variable and factor nodes with processing via the summary product algorithm



2. There is nothing about this level that is either inherently symbolic or probabilistic/Bayesian.

B. Variable nodes
   1. Represent one or more variables
   2. Compute point-wise products of incoming messages from associated factor nodes to generate outgoing messages to factor nodes
      a. Can be thought of as combining constraints from associated factors

C. Factor nodes
   1. Represent functions on variables
   2. Compute point-wise products of incoming messages from variable nodes with factor's function and summarize – via summation/integration or maximization – out all variables not used in variable nodes to generate messages back to them
   3. Maps variable names when variable nodes use different name spaces but some of the variables must be equal across these spaces.

D. Each message and factor function is represented as a continuous function over an N dimensional (continuous) space
   1. Each dimension represents one or more equivalent variables
      a. When there are multiple, each comes from a different variable node, and keep them equal by the corresponding dimension of the factor node maintaining a list of such variables, and each mapping to that dimension when computing products
   2. A message/function is currently approximated as a piecewise linear function based on an N dimensional array of rectilinear regions in which each region has an associated N dimensional linear function
   3. In general, there are many ways to consider representing such functions – which will be a topic for a later session – but the criteria for a good representation are:
      a. *Compactness* in representing functions of interest.
      b. *Closure* over the operations that need to be performed.
         i. *Product* of functions
         ii. *Summation/Integration* over dimensions of a function

0	0	0
0	0	1
.2+.1x+.3y	0	0

iii. *Maximization* over dimensions of a function
iv. *Changing* the value of the function over a region of the domain

c. *Efficiency* of execution of the operations.

4. With respect to the criteria, the current implementation has the following properties:
   a. It is *compact* to the extent that the value of nearby elements can be captured via a single N-dimensional linear function. Having either very squiggly value curves or many isolated symbols leads to fragmentation.
      i. However, even when there is a small set of specified regions, the complementary set of regions can become unwieldily large.
         a. May need to enable implicit/default representation of complementary regions.
   b. *Closure*
      i. Closed over summation and changing of values.
      ii. Product yields quadratic results that must be reapproximated as linear functions.
      iii. Maximization yields new piecewise linear functions, but the partitions it requires need not be rectilinear.
         a. Have implemented maximization with reapproximation to maintain rectilinear regions.
         b. Longer term it would appear to imply a need for regions that are convex polytopes (i.e., N dimensional polygons).
5. Exploring shifting to a piecewise log-linear representation
   a. $\mathrm{Ln}(F(x,y,z)) = A + Bx + Cy + Dz$ (or $F(x,y,z) = e^{A+Bx+Cy+Dz}$)
   b. Yields a patchwork of exponentials rather than lines.
      i. Facilitates representing things like exponential decay along a temporal dimension.
   c. Appears to be closed under both integral and product (and max, but issues here not yet completely understood).
   d. Better two-region approximation to Gaussian's based on back-to-back exponentials rather than pyramids.

III. The *prœscript* level

A. At this level, the graph is organized into a working memory (WM) and a long-term memory (LTM) via a compiler that converts prœscripts and evidence into appropriate graph structures

B. Working memory
   1. Traditionally viewed as a short-term, limited capacity store of active information.
      a. Contains ground instances rather than variablized patterns, implying a match process rather than full unification
      b. In Soar, structured as object-attribute-value triples rooted in states
         i. `(state s1 ^operator o1)`
         ii. Corresponds to a sparse three dimensional Boolean array in graph
            a. One dimension each for object, attribute and value

b. Domains of dimensions/variables are constants and ids

c. Although Soar 9 does have an increasing number of special purpose WM sub-regions for interacting with the outside world and various LTMs.

2. Discrete numeric data (e.g., integers) and symbols are represented by discretizing the underlying continuous space along a dimension
   a. Integers (`:discrete t :numeric t`) correspond to appropriate unit intervals
      i. For example, '1' maps onto the region [1,2>.
   b. Symbolic values (`:discrete t :numeric nil`) apply arbitrary labels to unit intervals
      i. For example, `red` may correspond to [30,31>.
      ii. Assumes the need for a symbol table.
   c. The initial implementation followed Soar's approach of undifferentiated variables structured into triples but, with such an approach, arbitrary mixing of numeric and symbolic data along a single dimension can yield representational conflicts
   d. For this reason, and because of issues where need to be able to identify which values of a variable are competing, and because it should improve efficiency to have related items clustered along a dimension, have introduced *types* on dimensions/variables and have structured combinations of them into *predicates*
      i. Tried predicates corresponding to objects rather than triples, but this was problematic when dealing with partial information about objects, so switched to predicates corresponding to attributes
         a. *Type* examples:

```
(new-type 'color :constants '(silver brown white))
```
Automatically defaults to `:discrete t :numeric nil :min 0 :max 3`
```
(new-type 'weight :numeric t :min 0 :max 500)
```
Automatically defaults to `:discrete nil`

         b. *Predicate* example:

```
(predicate 'color :arguments '((id id) (value color)))
```

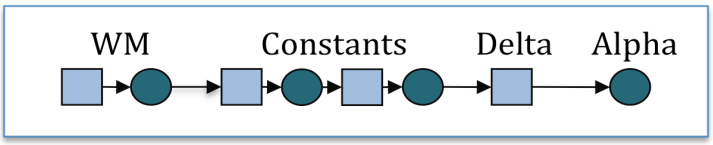
      ii. The result is rather like Alchemy's representation at this level

3. In the Bayesian interpretation of the elaboration cycle, WM is the natural home for evidence.
   a. Evidence is hard-and-fast/non-probabilistic knowledge specifying particular known values (groundings) for variables (although it is possible to add noise models).
      i. Priors, likelihoods and posteriors are all instead probabilistic patterns over sets of values.

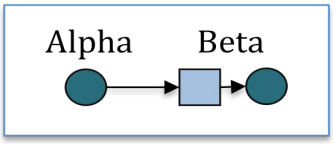
b. Could in fact conceive of WM as containing only evidence if willing to extend the concept of evidence to include the o-supported results of operator applications.
    i. I-supported results of elaborations will be discussed shortly.
c. One way to represent evidence in a graph is as a factor node's function, which is how WM is represented here
    i. There is a factor node for each predicate that stores its evidence
        a. WM is fragmented, and possibly even localized, across these nodes
        b. Should easily support specialized WM regions for perception and other purposes.
d. One question that has caused me continual grief is the meaning of a lack of evidence about groundings of a predicate
    i. Effectively means false in a classic rule system (weight is 0)
    ii. Means unknown in a probabilistic system (weight is 1 in initial messages)
    iii. I struggled for quite a while to find a single consistent semantics, including spending some time with what was essentially a multi-valued logic that added an *undefined* value as an alternative to a numeric value.
        a. No consistent way of viewing/treating *undefined* seemed to work in terms of whether it acts as an identity element during sum and product or causes the sum or product to become undefined.
    iv. I eventually decided to provide the ability to declare whether a predicate is open world or closed world – also similar to Alchemy – with the former meaning that any non-specified instance of the predicate defaults to a weight of 1 and the latter implying a default weight of 0.
        a. So far, this has let me make progress, but I may still learn more about this that destabilizes this solution; for example, should it prove necessary to use the same predicate in a condition/action and a pattern (see below).
    v. Am in the process of rethinking this based on trying to make sense out of a cluster of related distinctions:
        a. Prœscript
            1. o-supported versus i-supported
        b. Predicate (use)
            1. open world versus closed world
            2. fed by WM versus other predicates
            3. changeable during elaboration phase or only after
            4. used in conditions/actions versus in constraints
        c. Variable
            1. multiple versus unique

C. Long-term memory
    1. LTM consists of a set of *prœscripts*

a. Prœscripts were previously called *conditionals*, and the code still uses this term, so the examples embedded here also still use the old terminology.
b. Each prœscript is a list of patterns
    i. A pattern is a predicate, with constants or variables for the arguments (variables are in sublists)
        a. E.g.: `(color (id (i1)) (value brown))`
    ii. Each pattern corresponds to an *alpha memory* in a Rete network and becomes a variable node in the graph
        a. Sharing of alpha memories across prœscripts becomes important semantically when there are multiple constraints on the corresponding pattern, such as both priors and likelihoods
            1. This contrasts with the Rete model of sharing which is only for efficiency, because messages never go backwards from beta memories to alpha memories
            2. Sharing is signified explicitly by using *alpha variables* to remember and reuse patterns.
    iii. Patterns are connected by analogues of *beta nodes* in a Rete network; i.e., factor nodes that connect results from alpha nodes and earlier beta nodes.
c. Ended up splitting the list of patterns into three sublists, each with slightly different semantics
    i. *Conditions (generators)*
        a. Semantics correspond to those of a production condition
        b. The alpha (variable) node is constrained by the evidence; that is, by a match to working memory
            1. The working memory node is connected to the alpha memory through intermediate nodes that test for constants and map variable names from those used in the working memory to those used in the prœscript (*delta factors*)

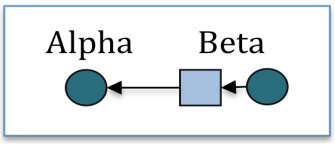


            2. Because this is simply propagation of evidence, and evidence is immutable, at least within a settling, no backwards messages are sent through this *alpha network*.
        c. Because this is a condition whose value is only to be determined by WM, propagation of information also does not happen backwards from the subsequent beta nodes.

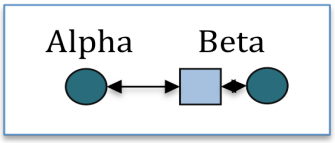


    ii. *Actions (absorbers)*
        a. Semantics correspond to those of a production action

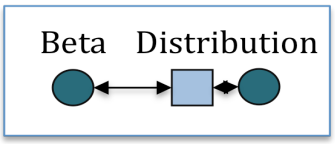
b. The alpha node is unconstrained by the evidence – i.e., it is not matched to WM – but it is constrained by any information propagated backwards from beta nodes.
c. No messages are sent from the alpha node to the attached beta nodes since actions do not provide information or constraint to any other patterns.



iii. *Constraints (propagators)*
a. In much of the rest of this note, and currently in the code, these are called *patterns*, but they are really constraints, and so I am moving to that nomenclature.
b. Semantics correspond to a node in constraint and probabilistic networks
c. The alpha node is constrained by WM, just as in a condition, including not sending messages backwards from the alpha node towards WM.
d. The alpha node is also constrained by any other messages propagating around in the beta network, so messages pass both ways between the alpha node and its associated beta nodes.



d. The condition list comes first in the graph, followed by the constraint list, and then the action list
i. A probability distribution is required over all of the variables appearing in the patterns that don't appear in conditions.
a. Distribution can be a constant '1' though.
b. Condition variables effectively generate distinct instantiations while pattern variables represent probabilistic attributes of these instantiations
c. Considering an alternative in which can add any number of distributions over combinations of variables in a prœscript.



ii. A *prior* is a prœscript with a single pattern and a distribution over its variables (italicized terms are variables; bold italicized terms are alpha variables; a distribution includes an entry/list for each region, with the first element being either a constant value or a linear function and the remaining elements either points or segments along the dimensions/variables (which are ordered as they are first used))

Example: P(time(*time*)/***time***)

```
(conditional 'time
   :patterns '((time time (value (time))))
```

```
   :distribution '((0 0) (0.032058604 1) (0.08714432 2)
                   (0.23688282 3) (0.6439142 4)))
```

Each entry in this distribution represents (1) a constant function on one variable/dimension: (2) *time.* So, for example, the first entry specifies that the value is 0 at time 0, and the second that it is 0.032058604 at time 1. The function as a whole specifies an exponential decay as time moves backward from the present.

iii. A *likelihood* is a prœscript with multiple patterns and a conditional probability distribution over the patterns.
   a. Evidence constrains the consequents, which in turn constrain the antecedents.

Example: P(alive(i1,*alive*)|time(*time*)/***time***)

```
(conditional 'time-alive
   :patterns     '((time time (value (time)))
                   (alive (id i1) (value (alive))))
   :distribution '((1 1 false) (1 2 true) (1 3 true) (1 4 true))
```

Each entry in this distribution represents (1) a constant function on the two variables over a region of the (2) *time* and (3) *alive* variables/dimensions. So, for example, the first entry says that the probability is 1 that at *time* 1 *alive* is false.

iv. A *posterior* can take one of two forms:
   a. A classic rule, comprising conditions and actions (the negative sign denotes a negated condition)

Example: next(*a*,*b*) & next(*b*,*c*) --> next(*a*,*c*)

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

Example: test-e(*id*,*t1*) & ¬test-n(*id*,*t1*) --> test-c(*id*,*t1*)

```
(conditional 'copy1
   :conditions '((test-e (id (id)) (value (t1)))
                 (test-n - (id (id)) (value (t1))))
   :actions    '((test-c tc (id (id)) (value (t1)))))
```

   b. Multiple patterns with a conditional probability distribution, just as with a likelihood, but with evidence constraining the antecedents, which in turn constrains the consequents.

Example: P(next(*a*,*c*)|next(*a*,*b*), next(*b*,*c*))

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

   c. I still need to understand this better.

v. Mixtures are showing up when, for example, conditions are used to match object identifiers but then the attributes of the objects are determined probabilistically (the * refers to the entire span of a dimension/variable; a linear function is defined by a list of weights, with the first being the constant and the subsequent ones being coefficients on the values of those variables)

Example: state(*state,id*)/***state-id***-->, P(concept(*id,concept)*/**id-concept)**

```
(conditional 'concept
  :conditions '((state state-id (state (state)) (id (id))))
  :patterns   '((concept id-concept (id (id)) (value (concept))))
  :distribution '((.25 *)))
```

Example: state(*state,id)*/***state-id***-->, P(weight(*id,weight*)|concept(*id,concept*)/***id-concept***)

```
(conditional 'concept-weight
   :conditions '((state state-id (state (state)) (id (id))))
   :patterns   '((concept id-concept (id (id)) (value (concept)))
                 (weight (id (id)) (value (weight))))
   :distribution '(((-2/15 0 2/75) walker (5 10))
                   ((4/15 0 -1/75) walker (10 20))
                   ((-2/1881 0 2/1881) table (1 20))
                   ((5/198 0 -1/3960) table (20 100))
                   ((-2/7301 0 2/7301) dog (1 50))
                   ((3/149 0 -1/7450) dog (50 150))
                   ((-2/59451 0 2/59451) human (1 150))
                   ((16/1995 0 -1/49875) human (150 400))))
```

Each entry/line in this distribution represents (1) a linear function on the two variables over a region of the (2) *concept* and (3) *weight* variables/dimensions. So, for example, the first entry is -2/15 + 0**concept* + 2/75**weight* for *concept*=walker and *weight* in [5,10>. Multipliers/weights for symbolic dimensions are almost always 0.

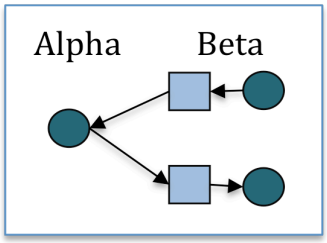
D. The key to how this provides first-order aspects of reasoning in factor graphs is that each element in a variable's domain can effectively become its own Boolean random variable that can take on a weight in [0,1]. Each message then contains information about all possible bindings, enabling match to compute variable bindings in parallel.
   1. Thus we are looking for all possible legal bindings of the variables rather than a single right/best answer.
      a. A bit like multi-attributes versus uni-attributes in Soar
   2. Can view this as either a non-Bayesian use of the graph or as a second-order (or first-order?) Bayesian approach.
      a. Need to declare such variables explicitly as *multiple* and always use MAX rather than SUM for them
         i. Each individual element of a multiple variable can weigh up to 1 simultaneously, whereas the weights on elements of a non-multiple variable would normally sum to 1 (if normalized).

3. Relates to the use of *templates* and *plates* in the graph world, but there – as with Alchemy – they end up compiling a ground graph from the templates/plates rather than sending around these more complex messages.

E. There are several possibilities for implementing negated conditions.
   1. Testing explicit negative information versus lack of positive information.
   2. Testing for a 0 value for positive element versus adding negative elements and testing for a 1 value for them.
      a. Need to extend standard graph inference mechanisms to do anything positive based on a 0 value.
   3. Implemented explicit negations
      a. Automatically generate a special Boolean attribute for objects: `inwm`
      b. Something in WM has a value of 1 for `(inwm true)` and 0 for `(inwm false)` while something that is absent has the inverse values
         i. Unknown items have both set to 1 to start (open world)
      c. A positive condition tests `(inwm true)` while a negated condition tests `(inwm false)`

F. How is working memory changed?
   1. Soar productions match to WM and generate changes to WM.
      a. One parallel cycle of match and firing is called an *elaboration cycle*.
      b. An *elaboration phase* comprises a sequence of elaboration cycles until *quiescence*; i.e., until no more productions can fire.
      c. Changes to working memory may be *i-supported* or *o-supported*, with the difference being that the former behave as in a justification-based truth maintenance system (JTMS); that is, changes are automatically retracted when the support for them goes away. O-supported changes are tied to *operator* implementation while i-supported changes result from *elaborations* (random rules not tied to operator implementation).
   2. In contrast, cycles in a graphical model involve sending messages, and quiescence occurs when there are no more messages waiting to be sent.
      a. It is possible to conceive of this all happening within each elaboration cycle, so that you wait for graph quiescence before modifying WM, but then have multiple such cycles within a single elaboration phase (and thus within a single decision).
      b. Alternatively, can reconceive of the elaboration phase as comprising the act of reaching quiescence in a graph.
         i. Changes to the values in WM would only occur at a decision, and would loosely correspond to o-supported changes to Soar's WM.
            a. Part of updating evidence prior to each new decision cycle.
         ii. Within an elaboration phase, messages would chain through variable nodes in the graph, without actually modifying WM. Such changes would be self-revising when WM changes, and thus would act something like i-supported changes to Soar's WM.

a. This is implemented by sharing an alpha node between an action in one prœscript and a condition in another (or between patterns in two prœscripts).
b. This would also be the locus of *trellises* for bounded forms of sequential (signal and theory of mind) processing during the elaboration phase.



G. A general concern to watch out for at the pattern level is whether the modifications of standard graph operations, such as in limiting the direction of flow of messages in conditions and actions, maintain the semantics of the factor graphs.
1. There is a temptation to do graph hacking to make things work
2. If the graphs can still be viewed as computing the product of subfunctions defined by the factors, then we are still within the semantics; otherwise we may still be doing something interesting, but it wouldn't quite be factor graphs.

IV. The *faculty* level
A. Pure procedural memory – traditional productions/rules – consists of prœscripts comprising just conditions and actions

Example: next(*a*,*b*) & next(*b*,*c*) --> next(*a*,*c*)

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

1. It is also possible to have posteriors with probabilities, based on patterns rather than conditions and actions, but I'm not sure how to think of these at the moment.

Example: P(next(*a*,*c*)|next(*a*,*b*), next(*b*,*c*))

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

B. Pure declarative (semantic and episodic) memory is based on priors and likelihoods, which themselves are based on patterns and distributions
1. It is possible to conceive of declarative memory being situated in working memory – if we remove the assumptions concerning its limited capacity and temporal extent – or in long-term memory.
a. Intriguing recent work by Brown, Neath and Chater shows that it is possible to model the traditional psychological working memory results without a distinction between short-term and long-term memories, and instead with a uniform approach based on a notion of

temporal discrimination of information that is analogous to spatial and other forms of perceptual discrimination.

2. I have done some thinking about, and experimentation with, WM versions of declarative memory, but not with great success so far, so I will concentrate here on LTM versions.
3. Semantic memory, of the categorization form discussed earlier, consists merely of one prœscript encoding the prior distribution over the concept labels, plus an additional prœscript for each attribute that encodes the likelihood of the attribute.
   a. The prior has a pattern for the concept label and a distribution over it.

Example: state(*state,id*)/***state-id***-->, P(concept(*id,concept)*/**id-concept)**

```
(conditional 'concept
 :conditions '((state state-id (state (state)) (id (id))))
 :patterns   '((concept id-concept (id (id)) (value (concept))))
 :distribution '((.25 *)))
```

   b. The attribute likelihoods have patterns for the concept label and the attribute, with a conditional distribution from the concept to the attribute.

Example: state(*state,id)*/***state-id***-->, P(weight(*id,weight*)|concept(*id,concept*)/***id-concept***)

```
(conditional 'concept-weight
   :conditions '((state state-id (state (state)) (id (id))))
   :patterns   '((concept id-concept (id (id)) (value (concept)))
                 (weight (id (id)) (value (weight))))
   :distribution '(((-2/15 0 2/75) walker (5 10))
                   ((4/15 0 -1/75) walker (10 20))
                   ((-2/1881 0 2/1881) table (1 20))
                   ((5/198 0 -1/3960) table (20 100))
                   ((-2/7301 0 2/7301) dog (1 50))
                   ((3/149 0 -1/7450) dog (50 150))
                   ((-2/59451 0 2/59451) human (1 150))
                   ((16/1995 0 -1/49875) human (150 400))))
```

   c. Given evidence on some of the attributes, sum-product generates a marginal over the concept label and predicts the other attributes by summing over all possible concepts.
4. In Soar 9, episodic memory is encoded by storing a snapshot of working memory once per decision into a special-purpose, long-term, episodic memory. This memory is accessed by creating a cue in a special part of WM, which results in retrieving the most recent best match from episodic memory into a special region of WM.
5. In the current approach, episodic memory is very much like semantic memory, except that:
   a. A time variable is used in place of the concept variable

i. The prior distribution on the time variable decays exponentially as you move further into the past

Example: P(time(*time*)/***time***)

```
(conditional 'time
  :patterns     '((time time (value (time))))
  :distribution '((0 0) (0.032058604 1) (0.08714432 2)
                  (0.23688282 3) (0.6439142 4)))
```

b. The attribute likelihoods contain instance rather than aggregate information.

Example: P(alive(i1,*alive*)|time(*time*)/***time***)

```
(conditional 'time-alive
  :patterns     '((time time (value (time)))
                  (alive (id i1) (value (alive))))
  :distribution '((1 1 false) (1 2 true) (1 3 true) (1 4 true))
```

c. You maximize rather than sum so that you retrieve information from the best matching episode rather than aggregating retrieval across all episodes.
   i. There is some additional trickiness to extracting the elements with the maximal values that is only partially implemented.

6. Other than the presence of the temporal variable – which will require some amount of architectural support – and its use in place of the concept variable, the key difference between semantic and declarative memory is the used of MAX rather than SUM.
   a. This could explain why they must be separate *modules* in some sense even though almost everything about their implementation is identical.
   b. Also implies that while it may be possible to retrieve attributes of multiple objects in parallel from declarative memory, can retrieve only one episode from episodic memory during any one settling?
      i. Because of the difference between computing all marginals and a single best answer.
7. One interesting thing that turned up while thinking about a WM implementation of semantic memory is that the difference between *analogical* and *inductive* transfer could fundamentally come down to whether you use MAX (to get the single best instance) or SUM (to get the generalizations across all instances).
   a. Shifting this concept to an LTM version of semantic memory, leads to an implemention as a set of instances encoded à la episodic memory, but with long-term object symbols rather than time in the condition, and the concept just one additional attribute

Example: P(object(*object*)/***object***)

```
(conditional 'object
```

```
  :patterns '((object object (value (object))))
  :distribution '((.125 *)))
```

Example: P(mobile(i1,*mobile*)|object(*object*)/***object***)

```
(conditional 'object-mobile
  :patterns ‘((object object (value (object)))
              (mobile (id i1) (value (mobile))))
  :distribution
             '((1 o1 true) (1 o2 true) (1 o3 true) (1 o4 true)
               (1 o5 true) (1 o6 false) (1 o7 true) (1 o8 true)))
```

b. Does appear to yield a memory which either returns best matched object(s) given evidence or probabilistic marginals for other attribute values

V. Peeking beyond the elaboration phase
   A. Decision making
      1. Have started to look at operator choice as the weighted selection of an id for the operator attribute of the state
         a. Preferences are implemented by manipulating the differential weights
         b. A few tricky issues remain to be addressed, such as:
            i. The creation of unique ids for dynamically generated operators
            ii. Multiple incompatible uses of a single alpha memory with differential variable (or constant?) bindings (also for *better* preferences)
         c. Impasses must arise from some form of inability to make such a decision.
      2. Or do we need to think in terms of multiple multivariate functions, each represented separately via graphs, or in some intermingled fashion?
         a. *Elaboration* is a function of state and represents joint distribution over it
         b. *Utility* is a separate function of the state and goal
         c. *Actions* are functions of operator plus prior and successor states
         d. *Preferences* are functions of state, operators and goal
   B. Problem solving
      1. Limited forms of search could occur within the elaboration phase (K-Search) via chaining in the graph.
         a. Limited in one sense, but expanded to handle trellises, etc.
      2. More traditional search (PS-Search) must occur across multiple elaboration phases (and accompanying decisions).
         a. One way to think about such search is by defining a multivariate function for the whole problem to be solved (achieving a goal or maximizing a numeric function).
            i. One model for (at least part) of this might then be the solution approach of *conditioning* in graphical models, where variables are heuristically set to concrete values in order to reduce the complexity of solving the graph, with an explicit combinatoric

search then being required over the possible combinations of variable values thus set.
    a. So, the basic idea is that message passing during the elaboration phase and conditioning across them.
  ii. In graphs in general, there are message-passing approaches and sampling (Monte Carlo) approaches. In analogy with constraint solving, it may make sense to think of all of the message passing as happening within the elaboration phase and sampling as occurring across decisions.

C. Learning
  1. Need to look for as general a set of mechanisms as possible for acquiring prœscripts of the appropriate forms from the knowledge that is available for learning.
     a. At least covering chunking and reinforcement learning for procedural knowledge plus acquisition of semantic and episodic knowledge.
  2. An alternative perspective is that need to learn each of the four(?) multivariate functions that may be needed as part of the decision cycle
     a. Elaboration learned from experience with evidence
     b. Utility learned from experience with reinforcers
     c. Actions learned from experience in performing them
     d. Preferences learned from experience with the action and utility functions
  3. A third possible perspective is that there are two general forms of learning possible and required
     a. Learning from snapshots to acquire data and regularities
        i. Would handle declarative and episodic learning
     b. Learning from traces/dependencies to (better) predict future
        i. Chunking records what is currently predicted
        ii. RL (and things like backprop) alters weights so as to better predict
     c. Is there a relationship between episodic memory and traces?
        i. Traces normally assume a causal/dependency structure which is not mandatory in episodic memories