*** Draft Submitted to AAAI10 Special Track on Integrated Intelligence on 1/15/10. ***
*** Not for quotation or distribution. ***

An Integrated Implementation of Rule, Semantic, Episodic and Constraint Memories via Factor Graphs and a Bayesian Decision Cycle

Paul S. Rosenbloom

Department of Computer Science & Institute for Creative Technologies
University of Southern California
13274 Fiji Way, Marina del Rey, CA 90292
rosenbloom@usc.edu

Abstract

The raison d'être of cognitive architectures is to *implement* and *integrate* the set of capabilities required for intelligent behavior. This dual task is approached here by harmonizing *diversity* and *uniformity*; using *factor graphs* to uniformly provide broad functionality, a *Bayesian decision cycle* as a unifying construct across diverse memories, and a *hybrid*, *mixed* variant of Soar 9 as a driving application. The result is a novel approach to integrated architectural diversity that effectively marries simplicity with broad functionality, as witnessed by an integrated implementation of four distinct long-term memories, one procedural (rules) and three declarative (semantic, episodic and constraint).

Introduction

Cognitive architectures provide a coherent integration of mechanisms necessary for intelligent behavior, whether as a tightly constrained model of human intelligence or a looser model of human-level artificial intelligence. The two key challenges in developing such models are: (1) providing the requisite diversity of intelligent capability; and (2) integrating this diversity into a coherent whole. The result ideally should be a simple elegant model that produces the full range of requisite capability. However, simplicity and diversity are usually at odds, making it difficult to develop architectures that meld broad functionality with the simplicity and constraint that are not only theoretically desirable but also critical for architectural integrability, extensibility and maintainability.

To resolve this *diversity dilemma* (Rosenbloom, 2009c), a strategy is being explored of developing a uniform *implementation level* that blends the generality needed for diverse capabilities with the simplicity and constraint necessary for integrated models. In turn, to drive and test this strategy, a *hybrid* (combining discrete and continuous processing), *mixed* (combining Boolean and Bayesian reasoning) variant of the Soar 9 architecture (Laird 2008) is being developed. Soar 9 combines Soar's traditional rule-based long-term memory, symbolic working memory, knowledge-based decision cycle, impasse-driven reflection, and learning by chunking with new semantic and episodic memories; semantic, episodic and reinforcement learning; imagery; and affective capabilities. The goal for the hybrid, mixed variant of Soar 9 is a new architecture that is simultaneously simpler and more functional.

Work to date has yielded a simpler yet more functional hybrid, mixed *elaboration phase*, the portion of Soar's decision cycle within which parallel cycles of long-term memory access repeat until quiescence. It is hybrid in supporting both discrete and continuous variables and mixed in supporting first-order probabilistic reasoning. It also incorporates in an integrated manner four distinct kinds of long-term memories, one procedural (rules) and three declarative (semantic, episodic and constraint). Yet the underlying implementation complexity is more comparable to that of the elaboration phases in earlier versions of Soar that were limited to accessing a symbolic rule memory than to the elaboration phase in Soar 9 with its additional modules for semantic and episodic memory.

How this works is explained bottom up across three system layers: (1) the *graph layer* exploits the uniform breadth of factor graphs (Kschischang, Frey, and Loeliger 2001) to move from symbolic to hybrid, mixed processing; (2) the *memory layer* uses Bayes law as a guide towards a uniform approach to handling procedural and declarative knowledge; and (3) the *mechanism layer* defines, and integrates together, the four long-term memories. Together these layers yield significant integrated architectural diversity that is grounded naturally in a broadly functional yet uniform implementation.

Extending this approach to a full architecture raises many additional issues for future attention, but these interim results already strongly suggest the feasibility of new models that are significantly more functional than today's best while exhibiting core complexity that is comparable to that of much simpler models.

The Graph Layer

Factor graphs, along with the broader class of graphical models (Jordan 2004) that also includes Bayesian (Pearl 1988) and Markov networks, provide an intriguing foundation for cognitive architecture because of how, based on a single representation (e.g., factor graphs) and reasoning algorithm (e.g., *summary product*), they can produce state-of-the-art algorithms for symbol, probability and signal processing; such as production match algorithms (Rosenbloom 2009a), loopy belief propagation (Pearl 1988), and hidden Markov models. This ability to yield diversity from uniformity is key to resolving the diversity dilemma. It is also crucial to integrating the resulting diversity, enabling for example the demonstration of hybrid, mixed reasoning (Gogate and Dechter 2005). Graphical models yield a striking combination of generality in the breadth of capabilities they can readily effectuate, with constraint in how these capabilities can naturally be implemented.

Factor graphs are undirected graphical models that were developed in coding theory for efficient computation over multivariate functions via their decomposition into products of reduced subfunctions. They are akin to Markov networks, but instead of representing subfunctions as clique potentials, factor nodes incorporate them directly into the network. Links exist between factor nodes and variable nodes wherever subfunctions draw on variables.

Factor graphs are typically used to compute either marginals on individual variables or the most likely single (MAP) hypothesis. Both computations can be performed via either some form of message passing or sampling. Message passing algorithms such as summary product, the focus here, pass messages between nodes about the values of variables. Nodes compute outgoing messages as the pointwise product of incoming messages, with factor nodes also multiplying in the factor's function and then summing out all variables not included in the target variable node. Although defined here via product and sum, summary product actually works for any pair of operations defining a *commutative semi-ring*, where both operations are associative and commutative and have identity elements, and the distributive law exists. Computing marginals involves sum-product while MAP uses max-product.

Rosenbloom (2009a) first proposed factor graphs as a uniform implementation level for cognitive architecture, and showed that they can yield a state-of-the-art production match algorithm, a capability at the heart of Soar and other architectures. However, this implementation was limited to symbol processing. Factor functions were represented as Boolean arrays, and messages among nodes were Boolean vectors with a 1 for any potentially legal element of the variable's domain and a 0 elsewhere. For efficiency, these arrays/vectors were structured as *exptrees*, nD generalizations of quad/octrees that represent uniform regions unitarily and decompose inhomogeneous regions into subregions. Functionally, the result was a piecewise constant (Boolean) WM representation. The algorithm yielded correct match, with worst-case cost reduced from exponential in conditions, as in the state-of-the-art Rete algorithm (Forgy 1982), to exponential in *treewidth*.

The key extension to the graph layer here is support for hybrid, mixed processing. Continuous functions replace Boolean arrays at the core of the implementation. Sum is replaced by integration while product and max remain unchanged. To implement this, a representation is wanted that can *compactly* and *accurately* approximate continuous functions of interest while enabling *efficient* computation that is *closed* over the relevant operations. Piecewise constant functions are efficient and closed, but yield a poor tradeoff between compactness and accuracy. Gaussians work well with probability densities, but are awkward for other functions. An interesting compromise is *piecewise linear functions* (PLFs). The nD space defined by the cross product of the variables' domains is partitioned into rectilinear regions, with each region specifying its own linear function over the variables (Figure 1).

y\\x	[0,10>	[10,25>	[25,50>
[0,5>	0	.2y	0
[5,15>	.5x	1	.1+.2x+.4y

Figure 1: 2D piecewise linear function (PLF).

PLFs are compact to the extent that large regions can be approximated accurately by linear functions; efficient because of the ease of computing with linear functions; and closed under summation/integration. Technically they are also closed under maximization, but only if the resulting regions aren't limited to rectilinear boundaries. If regions could be bounded by convex polytopes (nD polygons) then maximization would be closed. At present this is dealt with by reapproximating the results of maximization as new linear functions within rectilinear regions. The product of two linear functions is quadratic in general, so closure fails here. As with max, this is handled via reapproximation. Too much reapproximation can degrade accuracy, but this has not so far been an issue. Longer term, alternatives such as piecewise log-linear/exponential functions may improve closure while yielding a better combination of compactness and accuracy.

PLFs are inherently continuous in both domain and range, yielding natural representations for signals and probability densities. But individual domains can also be *discretized* – allocating unit intervals to integral values – to provide discrete probability distributions. Compounding continuous and discrete domains then supports hybrid processing. When range *Booleanization* – a restriction to 0/1 – is added to domain discretization, symbols ensue (with a symbol table provided for mapping domain integers onto symbolic labels). Compounding continuous and Boolean ranges supports mixed processing.

It is important to note that the summary product algorithm remains in complete ignorance of any discretization or Booleanization that might exist. It still processes such variables as if they were continuous, thus remaining simple and uniform across signals, probabilities and symbols. This contrasts sharply, e.g., with how mixed processing occurs in a system such as Alchemy, an implementation of Markov logic that combines first-order logic with Markov networks for first-order reasoning under uncertainty (Domingos et al. 2006). Alchemy first compiles input sentences expressed in a weighted first-order logic to a ground network, with variables replaced by all possible constant bindings, and then solves this ground network for probabilities. In a mapping of Soar's decision cycle onto Alchemy, in which WM mapped onto evidence in a database file and rules mapped onto implications, rule match occurred during graph compilation rather than graph solution, leading to an inhomogeneity between the processing of symbols and probabilities that conflicts with the uniform approach sought here (Rosenbloom 2009b).

For a classic random variable, a message reflects the probabilities of domain elements providing the unique correct value, with the domain probabilities all summing to 1. For a match variable, any subset of the domain elements may instead be correct, implying that each may vary independently in [0,1]. Alchemy deals with this by adding a ground node and a Boolean random variable for each domain element. In contrast, with general factor functions, each domain element in a message can effectively act as its own Boolean variable whose value ranges independently in [0,1], enabling single nodes and messages to cover all possible bindings at once. The graph layer uses this latter approach to enable uniform within-graph processing of both symbols and probabilities.

Although the graph layer can be difficult to grasp initially for those unfamiliar with the intricacies of graphical models, the overall implementation complexity of summary product is comparable to that of Rete, which is also at heart a message-passing algorithm. The only significant increase in complexity arises in moving from symbolic variables to continuous ones, and thus from piecewise constant to piecewise linear functions.

The Memory Layer

The memory layer organizes graphs into a working memory (WM) and a long-term memory (LTM), while defining the basic representations used in both. In so doing it engenders a Soar-like elaboration phase in which quiescence becomes a settling of the graph. In the prior work, the memory layer was closely modeled after Soar. WM was specified as a factor function; in particular, a 3D Boolean array with a 1 for every object-attribute-value triple present and a 0 elsewhere. Rules were compiled into graph structures and then matched via message passing in the graph. This demonstrated a new form of symbolic processing in graphical models – match – but yielded neither hybrid nor mixed processing. It also lacked a clear path towards bidirectional message passing across rules, as needed for correct probabilistic reasoning in general graphs and the kinds of *trellises* – chained repetitions of graph structures – used in speech and elsewhere.

In the current memory layer, WM comprises a set of nD, continuous, piecewise linear, factor functions. This is mostly just a natural consequence of the new graph layer, but in addition the traditional monolithic WM becomes partitioned into multiple local memories to avoid potential across-variable conflicts. Since the domains of variables can now be either numeric or symbolic, and symbols are represented as arbitrary mappings onto integers, the allocation of an integer to a symbol for one variable may conflict in WM with its concurrent use as a number for another. A solution to this problem has been adopted from the mapping of Soar's decision cycle onto Alchemy. Each attribute in Soar's representation becomes a predicate with typed variables – e.g., the condition `(<o> ^concept <c>)` becomes `Concept(`*`o`*`,`*`c`*`)`, with *`o`* and *`c`* typed symbolic variables over objects `[O1 O2 O3]` and concepts `[Walker Table Dog Human]` – and each predicate maintains its own factor function as part of WM.

LTM is a bit more complex. The ideal would be a single memory, with a general but simple and uniform knowledge representation that provides the capabilities needed by the procedural and declarative memories listed in the introduction. What exists at this point comes close to this ideal, but does not quite reach it. There is a single memory with a single general representation that subsumes the non-learning aspects of the four memories, but the complexity of the representation still leaves something to be desired.

The representation employed in LTM exploits the abstraction of a *Bayesian decision cycle* to compound procedural and declarative knowledge. Bayes law computes a situation's posterior probability given evidence P(S|E) as a function of its prior probability P(S), the probability of the evidence P(E), and the evidence's likelihood given the situation P(E|S): P(S|E) = P(E|S)P(S)/P(E). But since P(E) is constant, this reduces to P(S|E) = C*P(E|S)P(S). Mapping this onto Soar's decision cycle, based on hints from the Alchemy mapping, yields WM at the start of a cycle as evidence (E) and WM at the end of a cycle as the situation (S). The left side of the equation then specifies the direct, forward, computation of S from E; i.e., just what the existing procedural rule knowledge does. In contrast, the right side computes the result indirectly via likelihoods and priors. As will be seen, this is just what is needed for declarative memory.

Bayes law thus suggests a fundamental equivalence between procedural and declarative knowledge – analogous to, although not nearly as potent as, Einstein's equation for the equivalence between mass and energy – with the implication that a general representation subsuming the two sides of the equation could enable both flavors of knowledge and so provide a uniform auto-compatible basis for LTM diversity. Such a representation can be based on *conditionals*; generalized rules comprised of standard conditions and actions plus *condacts* (a neologism) and a function. A condact behaves like a combination of a *cond*ition and an *act*ion, matching existing elements *and* generating new ones. A function can be a probability distribution, or any other arbitrary PLF, over a subset of variables in a conditional. For example, given the concept predicate above, and a predicate for objects in the current state, Figure 2 defines a prior distribution over object O1's concept. A condact rather than an action is used for the concept here to enable the prior to be overridden by evidence in WM.

```
CONDITIONAL ConditionPrior
   Condition: Object(s,O1)
   Condact: Concept(O1,c)
```

Walker	Table	Dog	Human
.1	.3	.5	.1

Figure 2: Concept prior over object O1.

Conditions, actions and condacts are all *predicate patterns* that compile into factor nodes. Match occurs for conditions and condacts – but not actions – based on Rete-like *discrimination* and *join* networks (Figure 3), but with both networks implemented uniformly in the graph layer (and a treewidth match bound). The discrimination network computes all possible matches for its patterns via paths from predicate WMs to pattern nodes, with intermediate factor nodes performing constant tests. Messages flow unidirectionally along discrimination paths because WM/evidence is fixed during graph processing so that reverse messages can have no effect. The join network uses additional factors to combine matches across patterns. It only affects bindings for condacts and actions, not conditions. Condition and action patterns thus maintain unidirectional connections with join nodes – *to* and *from* join nodes, respectively – while condact connections are bidirectional. When there is a distribution, a factor node is added to the join network for it that links to all of the variables over which it is defined. Distributions can thus affect actions and condacts, but not conditions.

Conditionals can directly encode both sides of Bayes law, but encoding true rules implies more than just representing the law's left side. Distinguishing conditions and actions from condacts is one part of this. The other part is enabling a *closed-world assumption* for WM. Both constraint and probability processing assume an open world, where the truth of anything not mentioned is unknown. But rules assume that anything not in evidence/WM is false. Despite considerable effort searching for a single semantics for unmentioned elements usable for both forms of processing, in the end both possibilities were provided as options. When a predicate is defined, it is declared as either open or closed world. Condacts are generally open world while conditions and actions are generally closed world.

How this memory layer realizes, and integrates together, the four memories mentioned in the introduction, is described in the next section. But first a subtle issue concerning *locality* in the memory layer needs a brief discussion. Several aspects of the processing of Soar's rules are global, including interaction with WM and (implicit) use of negation-as-failure in negated conditions. Yet a layered analysis of Soar derived from the Alchemy mapping implies that rules should be restricted to local processing, with global processing the demesne of the decision cycle. To reflect this, *pattern variables* have been added that bind to the results of conditional patterns – whether conditions, actions or condacts – which, when reused either within or across conditionals, cause their associated patterns to compile to the same nodes in the graph. This means, e.g., that the action of one conditional can directly feed the condition of another without going through WM. Likewise, condacts can be linked directly across conditionals to yield chains/trellises that embody the kind of bidirectional message passing mentioned at the top of this section. The global WM is only accessed directly at the beginning of the elaboration phase and changed at the end of it. All communication within the elaboration phase happens via shared nodes in the graph. A local form of negated condition has also been added, based on explicit representation of what is not in WM, rather than depending on the global semantics of negation as failure.

The Mechanism Layer

At the mechanism layer, the uniform lower layers implement integrated architectural diversity. To date this has included (non-learning) variants of: (1) a rule-based procedural memory, (2) a concept-based semantic memory, (3) an event-based episodic memory, and (4) a constraint memory. The distinction between procedural and declarative memory is a familiar one in cognition, and has been the long-term basis for architectures such as ACT-R (Anderson 1993). The distinction between semantic and episodic memory also has a long history (Tulving 1984), and is reified in such architectures as Soar 9. Constraints (Dechter 2003) are familiar and very useful declarative structures in AI, but have not so far played a prominent role in architecture. The original plan for this work only included the first three memories, but constraints came along essentially for free, so they are included here as well.

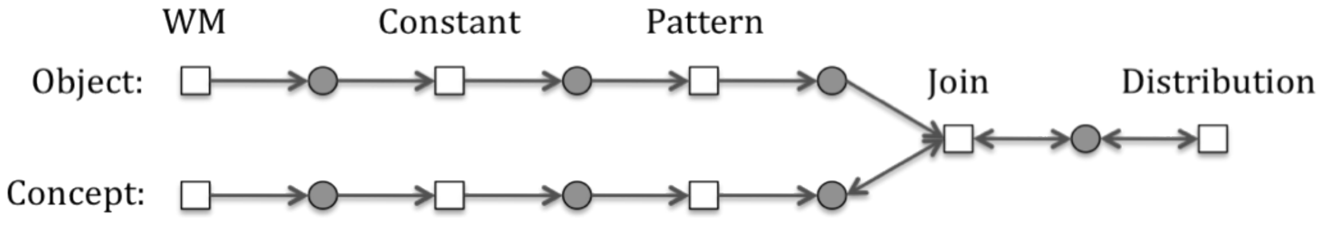

Figure 3: Factor graph for conditional in Figure 2. Factor nodes are white boxes and variable nodes are grey circles.

Implementing these memories via the lower layers does not yield distinct modules with hard boundaries between them; rather, each is defined by a set of conditionals that combine to yield the appropriate functionality, along possibly with substituting *max* for the default *sum*. Rule and constraint memories are trivial to implement given conditionals. Rule memory uses conditions and actions on closed-world predicates (Figure 4), while constraint memory uses condacts on open-world predicates plus a Boolean function (Figure 5).

```
CONDITIONAL Transitive
   Condition: Next(a,b)
              Next(b,c)
   Action: Next(a,c)
```

Figure 4: Transitive rule.

```
CONDITIONAL TwoColorConstraint12
   Condact: Color(R1,c1)[α1]
            Color(R2,c2)[α2]
```

c1\c2	Red	Blue
Red	0	1
Blue	1	0

Figure 5: Two-color constraint between regions R1 & R2. Pattern variables *α1 & α2* connect with other constraints.

Semantic memory encodes knowledge about objects in terms of prior probabilities on concepts P(C) and conditional probabilities of features given concepts P(F|C), à la Anderson's (1990) rational analysis. Feature prediction for a new object uses conditional probabilities backwards from its cued features along with prior concept probabilities to compute a distribution over its concept. This is in turn used with conditional distributions for its uncued features to determine their predicted distributions. The implementation involves condacts and functions. There is one conditional for the concept prior – like the one in Figure 2, but without the condition – and one for each feature's conditional distribution (Figure 6). Cues appear as evidence in WM, and all predicates are open world. Together with sum-product, this yields probabilistic predictions of uncued object features from cued ones.

```
CONDITIONAL ConceptWeight
   Condact: Concept(O1,c)[α3]
            Weight(O1,w)[α4]
```

w\c	Walker	Table	…
[1,10>	.01w	.001w	…
[10,20>	.2-.01w	"	…
[20,50>	0	.025-.00025w	…
[50,100>	"	"	…

Figure 6: Conditional probability of weight given concept.

In Soar 9, episodic memory retrieves the most recent episode that best matches the cues, effectively acting as a temporal instance-based semantic memory. It can thus also be implemented via prior and conditional probabilities, but with alterations for recency and retrieval of the best individual episode rather than prediction of likely feature values. A discrete temporal variable replaces the concept variable, with a prior distribution that tails off exponentially as time recedes into the past. Each feature conditional specifies its actual values over past history. Max-product is used to retrieve the best match given the specified cues, the feature conditionals, and the recency bias defined by the temporal prior.

It turns out it is also possible to implement a form of semantic memory in such an instance-based fashion, with object instances represented explicitly and the concept just another feature. With sum-product, this dynamically computes distributions over features, but at present without leveraging numerical relationships for (non-constant) linear generalizations over regions. With max-product, it instead retrieves the individual object best matching the cues. One intriguing implication is that the causative difference between generalization and analogy may reduce to whether sum-product or max-product is used over an instance-based memory. The former generalizes over all instances, while the latter retrieves the single best instance.

But what about integration across these memories? Conditionals by themselves enable combining procedural and declarative capabilities within individual memory elements. Semantic memory provides a simple example. In addition to condacts and a function, each conditional can also include a condition that matches multiple objects in WM, each with their own cues. The prior then resembles the conditional in Figure 2, but with the constant object O1 replaced by a variable. Feature conditionals resemble Figure 6, but with the condition added and the variable substituted (Figure 7). Like Soar 9, there can be only one cycle of semantic memory retrieval per decision here – because the graph must settle during retrieval – but unlike Soar 9, features of many objects can be predicted in parallel within this single cycle. In a similar manner, it is possible to combine aspects of rules and constraints within conditionals, and to have rules with probabilistic aspects.

```
CONDITIONAL ConceptWeightGeneral
   Condition: Object(s,o)[α5]
   Condact: Concept(o,c)[α6]
            Weight(o,w)[α7]
```

Figure 7: Conditional distribution for semantic memory with condition to match objects (shown without function).

To integrate across the four memories implemented at the mechanism layer requires going beyond this though, to enable conditionals from different memories to interact. Within the elaboration phase, shared pattern variables are the key. The rule in Figure 8, for example, matches the results of Figure 7's semantic retrieval via pattern variables and generates a new ConceptWeight predicate. This also exploits within-conditional integration, but here in service of across-memory integration.

To the extent that the elements of a memory can be accessed independently of each other – as is true, e.g., of

```
CONDITIONAL ConceptWeightRule
   Condition: Object(s,o)[α5]
   Condact: Concept(o,c)[α6]
            Weight(o,w)[α7]
   Action: ConceptWeight(c,w)
```

Figure 8: Accessing result of semantic retrieval in a rule.

rules in a system like Soar – interaction across memories can happen at the level of individual memory elements. When access is a global process over a memory – such as with the declarative memories implemented here – it only makes sense to consider interaction among memories as a whole, even though this is still mediated by individual conditionals and pattern variables. Interaction among memories can also occur across decision cycles when changes in WM enacted by one memory on one cycle trigger activity in another on the next.

Conclusion and Future

Previous work proposed resolving the diversity dilemma by building diverse architectures on a uniform, graphical, implementation level. Here, the graph layer preserves much of what was proposed for the implementation level, and the mechanism layer does the same for the architecture level, but the memory layer has been slotted in between to link these two earlier concepts. Via the memory layer, and its abstraction of a Bayesian decision cycle, the hybrid, mixed capability now implemented in the graph layer has been harnessed to produce an integrated implementation of four distinct memories, one procedural (rules) and three declarative (semantic, episodic and constraint); demonstrating in the process the impact of harmonizing diversity and uniformity on integrated architectural diversity. Although the examples are admittedly toy, they do reveal how simple uniform LTM structures that are capable of broad functional integration within themselves can naturally implement, and integrate together, multiple distinct mechanisms.

The immediate future stratifies by layer. At the graph layer, the push is towards improved closure and accuracy in function approximation. At the memory layer, the focus is on reducing the complexity of conditionals through a deeper combination of their procedural/rule and declarative aspects. At the mechanism layer, the emphasis is on adding the other capabilities required for completing a hybrid, mixed version of Soar 9. Tapping the full potential of hybrid processing, for example, implies enabling fine-grained interaction between perception and cognition through direct incorporation of perception into the elaboration phase, rather than leaving it marginalized in peripheral processes. This should be feasible given that today's state-of-the-art speech and vision systems are based on graphical models; that is, on hidden Markov models and Markov random fields, respectively.

Over the longer term, the applicability of this uniform foundation for integrated architectural diversity must be extended beyond Soar, to other existing architectures and to the creation of radically new architectures enabled by the unique synergies provided by graphical models.

Acknowledgements. This effort has been sponsored by the USC Institute for Creative Technologies and the U.S. Army Research, Development, and Engineering Command (RDECOM). 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. I would like to thank Bill Swartout for helpful comments on restructuring this work for publication.

References

Anderson, J. R. 1990. *The Adaptive Character of Thought.* Hillsdale, NJ: Erlbaum.

Anderson, J. R. 1993. *Rules of the Mind.* Erlbaum.

Dechter, R. 2003. *Constraint Processing*. San Francisco, CA: Morgan Kaufmann.

Domingos, P., Kok, S., Poon, H., Richardson, M., and Singla, P. 2006. Unifying logical and statistical AI. In *Proceedings of the 21st National Conference on Artificial Intelligence*, 2-7. AAAI Press.

Forgy, C. L. 1982. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence* 19(1): 17-37.

Gogate, V. and Dechter, R. 2005. Approximate Inference Algorithms for Hybrid Bayesian Networks with Discrete Constraints. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, 209-216.

Jordan, M. I. 2004. Graphical models. *Statistical Science* 19: 140-155.

Kschischang, F. R., Frey, B. J., and Loeliger, H. 2001. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory* 47: 498-519.

Laird, J. E. 2008. Extending the Soar cognitive architecture. In *Artificial General Intelligence 2008: Proceedings of the First AGI Conference*. IOS Press.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA: Morgan Kaufman.

Rosenbloom, P. S. 2009a. Towards a new cognitive hourglass: Uniform implementation of cognitive architecture via factor graphs. In *Proceedings of the 9th International Conference on Cognitive Modeling*.

Rosenbloom, P. S. 2009b. A graphical rethinking of the cognitive inner loop. In *Proceedings of The IJCAI International Workshop on Graph Structures for Knowledge Representation and Reasoning*.

Rosenbloom, P. S. 2009c. Towards uniform implementation of architectural diversity. In *Proceedings of the AAAI Fall Symposium on Multi-Representational Architectures for Human-Level Intelligence*, 32-33.

Tulving, E. 1984. Precis of Elements of Episodic Memory. *Behavioural and Brain Sciences* 7: 223 – 268.