# Rethinking Cognitive Architecture via Graphical Models

**Paul S. Rosenbloom**
Department of Computer Science and Institute for Creative Technologies
13274 Fiji Way, Marina del Rey, CA 90292 USA
Email: rosenbloom@usc.edu; Phone: (310) 448-5341; Fax: (310) 301-5009

To appear in *Cognitive Systems Research* (2010)

## Abstract

Cognitive architectures need to resolve the *diversity dilemma* – i.e., to blend diversity and uniformity – in order to couple functionality and efficiency with minimality, integrability, extensibility and maintainability. Building diverse architectures upon a uniform implementation level of graphical models is an intriguing approach because of the homogeneous manner in which such models produce state-of-the-art algorithms spanning symbol, probability and signal processing. To explore this approach a *hybrid* (discrete and continuous) *mixed* (Boolean and Bayesian) variant of the Soar architecture is being implemented via graphical models. Initial steps reported here, including a graphical implementation of production match and the beginnings of a mixed decision cycle incorporating a simple semantic memory, begin to show the potential of such an approach for cognitive architecture.

**Keywords:** Cognitive architecture; graphical models; factor graphs; production match; Soar; Markov logic

A *cognitive architecture* is a hypothesis about the fixed mechanisms underlying intelligent behavior. It seeks to provide a coherent integration of capabilities sufficient for human-level artificial intelligence, whether as a detailed model of human cognition or a system more loosely tied to the specifics of human behavior. Such an architecture requires the integration of a wide range of cognitive capabilities for, among other things, representation and memory, problem solving and planning, learning, reflection, interaction (including perception and motor control, use of language, etc), and the social aspects of cognition (such as emotion, collaboration, etc.). Functionally an architecture can play a pragmatic role as an executable model of cognition – when combined with knowledge upon which the fixed mechanisms can operate – but it can also play a theoretical role as a hypothesis concerning the sufficiency and/or necessity of the combination of mechanisms it embodies.

A key issue for such architectures is what can be called the *diversity dilemma*: they must simultaneously be both diverse *and* uniform. Diversity of functionality is required to support intelligent behavior in a complex and uncertain world and to model the range of experimental results exhibited by human cognition. As an architectural strategy, diversity implies openness to a proliferation of mechanisms, and appeals to notions of specialization and optimization. In caricature it is the biologist's approach, in which many specialized structures, each locally optimized, jointly yield a robust and coherent whole. Uniformity on the other hand is critical for architectural minimality, integrability, extensibility and maintainability. As an architectural strategy, it implies conservatism towards the addition of mechanisms, and appeals to notions of elegance and simplicity. In caricature it is the physicist's approach, where a broad diversity of phenomena emerges from interactions among a small set of general elements. Ockham's razor

can be viewed as a preference for uniformity in theories to the extent that the same set of phenomena can be explained via a more minimal basis.

Traditionally architectures have had to choose one approach or the other. Among architectures for cognitive modeling, Soar (Rosenbloom, Laird & Newell, 1993) has been a canonical example of the uniformity strategy and ACT-R (Anderson, 1993) of the diversity strategy. However, Soar 9 (Laird, 2008) has recently shifted strongly towards diversity. Looking back, the history of Soar (Laird & Rosenbloom, 1996; Laird, 2008) could be said to illustrate a *uniformity-first* research strategy (a temporal variant of Ockham's razor): begin by assuming uniformity and accept diversity only upon overwhelming evidence. As long as uniformity is possible, such a strategy yields simplicity and elegance while facilitating unification and extension. Beginning instead with diversity removes the pressure to search for hidden commonality, and may lead down an irrevocable path of both theoretical and systems complexity. For years Soar had a single procedural, rule-based, long-term memory and a single learning mechanism, while investigations were pursued into the ability of this combination to support a diversity of memory (e.g., procedural, semantic, and episodic (Rosenbloom, Newell & Laird, 1991)) and learning (e.g., skill and knowledge acquisition, generalization and transfer, and learning from observation (Rosenbloom, 2006)) behaviors. A wide range of such behaviors proved feasible, but they never could be fully unified with the rest of the system to yield pervasive utility across all activity. This evidence against the existing uniformity, amassed over years of experimentation, inspired the development of Soar 9, a diverse architecture that adds new long-term memories (semantic and episodic (Nuxoll & Laird, 2004)) and learning mechanisms (semantic, episodic and reinforcement (Nason & Laird, 2005)), while also incorporating other new capabilities (emotion (Marinier & Laird, 2004) and imagery (Lathrop & Laird, 2007)).

But the acceptance of architectural diversity leaves open the question of how to deal with issues of minimality, integrability, extensibility and maintainability. Diverse architectures, such as Soar 9 and the various flavors of ACT, can be difficult to integrate, risking an end product that is more of a toolkit than an architecture. Software engineering techniques, such as modularization, provide one path for coping with the pragmatic issues. ACT-R, along with cognitive modeling frameworks such as Storm (Pearson, Gorski, Lewis & Laird, 2007) and Cogent (Cooper & Fox, 1998), are exploring this path. But there is an alternative path, based on continued adherence to uniformity-first – combining an acceptance of the need for diversity in the architecture with a continued search for uniformity elsewhere in cognition – that shows potential for yielding the desirable attributes that are characteristic of uniformity in what is otherwise a diverse system.

Newell (1990) proposed understanding cognition in terms of a hierarchy of spatiotemporal levels. Given such a hierarchy, it is possible to ask about the *girth* – that is, the variety of mechanisms – at each level. Diversity always exists across levels, but individual levels may consist of anything from a small number of very general elements to a wide diversity of more specialized ones. Across a hierarchy of levels, there is no a priori reason to assume they are all of comparable girth. While physicists and biologists may expect uniformity within their fields, the networking community trumpets the *Internet hourglass* to explain their protocol stack (Deering, 1998). At the narrowed waist is the Internet Protocol (IP). Above is an increasingly diverse sequence of levels enabling "everything on IP". Below is an increasingly diverse sequence of levels enabling "IP on everything". The hourglass yields a diversity of applications and implementations that are united via a core of *mesoscale uniformity*.

In analogy, we can ask whether there remains a core of mesoscale uniformity in cognition, and whether it can be the key to resolving the diversity dilemma. There must clearly be diversity at the top of the cognitive hierarchy; the extraordinary range of behaviors and applications of which humans are capable is one of the core phenomena cognitive architectures are developed to explain. At the bottom, the mind is grounded in the diverse biology of the brain and, at least according to *strong AI*, could also be grounded in a diversity of alternative technologies. But is there an hourglass or a rectangle in between? Domingos's recent call for an *interface layer* in AI (Domingos & Lowd, 2009) amounts to an appeal for a *cognitive hourglass* in the building of AI systems in general, although not directly focused on architecture. Within cognitive modeling, the question of the existence of a cognitive hourglass has historically been cast in terms of whether the cognitive architecture is uniform. The approach here is instead to accept the need for diversity in the architecture, as in ACT-R and Soar 9, while continuing the search for uniformity at the *implementation level* below the architecture. The goal is still an hourglass, albeit one with a lower waistline.

Architectural implementation has traditionally been considered mere "implementation details," of pragmatic importance for efficiency and robustness – and potentially of use in exploring and understanding (the space of) architectures (Cooper, Fox, Farringdon & Shallice, 1996) – but of little theoretical interest. The one notable exception has been when the implementation level is based on neural networks, as in Neuro-Soar (Cho, Rosenbloom & Dolan, 1991) and SAL (Jilk, Lebiere, O'Reilly & Anderson, 2008). In such cases, theoretical hypotheses that are otherwise limited to the architecture can expand down to include the workings of the neural level that implements it. Due to the computational model embodied by neural implementations, they may also constrain or influence what assumptions and mechanisms occur at the architecture level, thus indirectly impacting the theory at that level too. However, neural networks are not the only approach to the implementation level. If we are concerned with either modeling cognition in the abstract – i.e., independently of the details of human cognition – or modeling human cognition but based on more abstract models of what might be going on in the brain, there are other possibilities that may more effectively yield fruitful results of both pragmatic and theoretic import.

In this article, *graphical models* (Koller & Friedman, 2009) are explored as a promising alternative that fits both of these forms of abstraction. They show significant potential for modeling intelligence in the abstract while also being reasonably viewable as abstractions of neural networks. Graphical models are based on local processing in networks of nodes, as with neural networks, but their essence concerns efficient computation with complex multivariate functions via their decomposition into products of simpler subfunctions. The original multivariate function fixes the meaning of the graph, and thus what it computes, while the decomposition into subfunctions determines the structure of the graph used in the computation, and thus how the computation plays out. Bayesian networks (Pearl, 1988) are directed graphical models over random variables that have revolutionized probabilistic reasoning. Markov networks (aka Markov random fields) are undirected analogues of Bayesian networks that go beyond probabilities to allow general weights, or *clique potentials*, on groups of variables. They are at the core of many state-of-the-art algorithms for visual perception (Li, 2009). Factor graphs (Kschischang, Frey & Loeliger, 2001) are undirected, like Markov networks, but were developed in coding theory – where they underlie the "astonishing performance" of turbo codes – to represent and reason efficiently with general multivariate functions. They differ from Markov networks in replacing clique potentials with factor nodes directly in the graph, and can subsume

both Bayesian and Markov networks (Frey, 2003). While neural networks are obviously graphical in structure, only some variants – such as supervised Boltzmann machines, radial basis functions, and unsupervised learning algorithms – actually map onto graphical models in this sense (Jordan & Sejnowski, 2001).

What is particularly tantalizing about the idea of basing the implementation level for cognitive architectures on graphical models is how they combine: (1) *generality* in the range of capabilities they can support in a state-of-the-art manner; (2) *uniformity and simplicity* in how they implement these capabilities; and (3) *constraint* in the ways that these capabilities can reasonably be supported. Their generality is impressive, subsuming such state-of-the-art algorithms as arc consistency in constraint satisfaction (symbol processing), loopy belief propagation in Bayesian networks (probability processing), and Kalman filters and the forward-backward algorithm in hidden Markov models (signal processing). But what is even more striking from an architectural perspective is how these well known algorithms that span symbol, probability and signal processing are all simple consequences of a single uniform inference algorithm – the *sum-product algorithm* (Kschischang et al., 2001) – when it is applied to straightforward graphs of the appropriate kinds.

The resulting combination of generality and simplicity can yield substantial pragmatic benefits, supporting the development of simpler systems with broader functionality, while simultaneously easing the challenges of integration, extension and maintenance. It can also contribute to cognitive theory by: (1) providing more minimal explanations for the diversity seen in existing architectures; and (2) opening up new theoretical possibilities in cognition by enabling the development of architectures with new varieties of mechanisms and means of combining them. For example, one of the long-term goals of this effort is to develop a new more simply implemented architecture that has mechanisms comparable to those in Soar 9, but which is pervasively *hybrid* (combining discrete and continuous processing) and *mixed* (combining Boolean and Bayesian reasoning). Such an approach could elegantly and more minimally explain the diversity seen in existing architectures while going beyond them to yield a simple yet effective basis for: unifying cognition with perception and motor control by breaking down the barriers between central and peripheral processing, bringing the latter within the cognitive inner loop and making each form of processing potentially penetrable by the other; and fusing symbolic and probabilistic reasoning to provide general reasoning under uncertainty.

As with any computational model, graphical models make some kinds of computations easy to implement and others more complex. Implementation complexity in graphical models depends in particular on such factors as the decomposability of the multivariate function, whether the computation over the resulting graph is local or global, and whether there are loops in the structure of the graph. The result is that apparently complex calculations, such as those implemented by the state-of-the-art algorithms mentioned above, may end up being as simple as, if not simpler than, an apparently straightforward bit of global communication. If the use of graphical models at the implementation level is taken seriously as a theoretical claim about cognition, instead of merely viewing them as a convenient implementation language, such patterns should influence the nature of the architectures implemented.

The overall research strategy therefore includes reimplementing existing architectures to help better understand them and the pragmatic and theoretical implications of implementing them via graphical models. It also includes going beyond existing architectures by hybridization and simplification, both within and across architectures; integrating in new capabilities that don't mesh well with existing architectures, such as perception and motor control; and exploring

radically new architectures enabled by the unique strengths of graphical models. Ultimately the hope is for both a better understanding of the space of architectures and the development of radically new architectures embodying heretofore-unseen combinations of functionality and uniformity.

The initial focus is on a graphical reimplementation and extension of Soar, with the aim of developing a simple uniform implementation of a mixed hybrid variant. Soar is particularly useful as a starting point because it: is one of the longest standing – over 25 years – and most thoroughly investigated cognitive architectures; has been explored as both a unified theory of human cognition and as an architecture for intelligent agents and virtual humans; and exists in both uniform (versions 1-8) and diverse forms (version 9), enabling a strategy of starting reimplementation with the initial uniformity while seeking opportunities for a more uniform integration of the later diversity. Concurrently we can explore extensions beyond Soar's predominant symbol processing paradigm, through the deep integration of probability and signal processing in support of improved reasoning about, and interaction with, the real world. Lending encouragement to such an enterprise is the success of existing work on hybrid mixed graphical models (Gogate & Dechter, 2005; Wang & Domingos, 2008), plus the recent development of a variety of general languages for mixing probabilistic and symbolic reasoning. FACTORIE (McCallum, Rohanemanesh, Wick, Schultz & Singh, 2008), for example, combines factor graphs with an imperative programming language to support relations and other capabilities, while BLOG (Milch, Marthi, Russell, Sontag, Ong & Kolobov, 2007) and Alchemy (Domingos, Kok, Poon, Richardson & Singla, 2006) combine probability and logic via Bayesian and Markov networks, respectively.

This article reports initial steps along this path. Section 1 presents the essence of "uniform" Soar (versions 3-8) across a hierarchy of three cognitive layers (at time scales of 10ms-1s). Section 2 provides an introduction to factor graphs as an exemplar of graphical models, and as the key technology for a graphical reimplementation of Soar's lowest layer, the *elaboration cycle*. This reimplementation, as described in Section 3, establishes the initial applicability of graphical models to production match, a central component in Soar and other architectures. Section 4 then uses Alchemy to explore the implications of a general mixed capability for extending Soar's *decision cycle*, the next layer up in the architecture, to reasoning under uncertainty and a more uniform integration of Soar 9's additional capabilities; in particular, a simple form of semantic memory. Results of this exploration include a new functional analysis of Soar's layers that has potential implications across a much wider range of cognitive models. Section 5 concludes with next steps.

## 1 Cognitive Scales and Soar

Newell's notion that scale counts in cognition provides a key component of Soar's theoretical background as a model of human cognition. The core idea is that the phenomena of interest in cognition change as the focus shifts from small spatiotemporal scales to larger ones. Newell discusses time scales from $10^{-4}$ sec (100 μs) up to $10^{7}$ sec (months), stratifying them into four bands in human cognition: biological ($10^{-4}$-$10^{-2}$ sec), cognitive ($10^{-1}$-$10^{1}$ sec), rational ($10^{2}$-$10^{4}$ sec) and social ($10^{5}$-$10^{7}$ sec). In the biological band in particular there is also a spatial aspect to these scales, since signals within the brain are limited in how far they can travel within such small time periods. Organelles ($10^{-4}$ sec), neurons ($10^{-3}$ sec) and neural circuits ($10^{-2}$ sec) yield spatial scales within the biological band, before primitive deliberate acts ($10^{-1}$ sec) and operations ($10^{0}$ sec) are reached at the base of the cognitive band.

Architectural mechanisms in uniform versions of Soar were traditionally mapped onto a subset of these time scales, as in Figure 1, starting with the *elaboration cycle* at 10 ms (neural circuits), the *decision cycle* at 100 ms (deliberate acts), and *problem space activity* at 1 sec (operations). In the context of Soar, the implementation level implements these mechanisms. It resides within the biological band in the human cognitive hierarchy, but just as Soar's elaboration cycle abstracts away from the details of neural circuits at the 10 ms scale, an implementation level based on graphical models may abstract away from biological details at multiple scales.

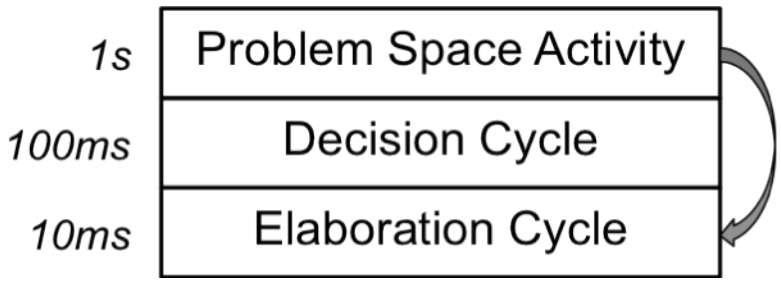


Figure 1: Soar's architectural levels (with chunking).

Soar's elaboration cycle involves parallel match – via a variant of the Rete algorithm (Forgy, 1982) – and firing of productions based on the contents of a global working memory (WM). Functionally, it achieves one round of parallel associative retrieval of information relevant to the current situation. Production actions specify knowledge for retrieval while production conditions specify when it should be retrieved. Conditions also bind variables for use in actions.

The decision cycle begins with repeated cycles of elaboration until quiescence; i.e., until no more productions can fire. This *elaboration phase* is followed by a decision based on preferences retrieved during elaboration. The elaboration phase yields an interpretation of the current situation, while the decision either selects an operator to apply or generates an *impasse* if no operator can be selected. Impasses engender *reflection*, enabling processing to recur at the meta-level on the problem of making the decision. The decision cycle is Soar's cognitive inner loop; it accesses whatever knowledge is immediately available about the current situation and then attempts to decide what to do next.

A sequence of decisions yields activity in a problem space, amounting to some form of search if knowledge is limited and impasses occur. Search in problem spaces (ps-search) is: *slow*, with each decision occurring at the 100 ms level; *serial*, via a sequence of operator selections and applications; and potentially *combinatoric*, yielding trees that grow exponentially in the depth of the search. However, ps-search is open to control by any knowledge accessible during the decisions that occur as part of it. When the knowledge is sufficient to uniquely determine the outcome of each decision, behavior is more accurately characterized as algorithmic, or knowledge-driven, than as search.

Determining which knowledge to access during a decision can also be viewed as a search process – termed knowledge search (k-search) – but one that contrasts strongly with ps-search in character. K-search is: *fast*, with a 10 ms cycle time; *parallel*, both in match and firing of productions; and *subexponential*, at least in theory, if not in reality in most implementations. K-search occurs over a closed, extensionally defined, set of structures – the knowledge/productions in the system – rather than dynamically generating an open search space in the manner of ps-search. It is inherently algorithmic, rather than using an open cognitive loop, and is thus not itself penetrable by additional control knowledge. In the work described here, the focus is on implementing k-search via graphical models.

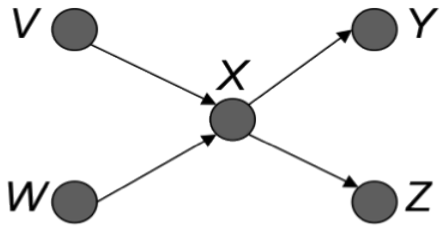
*Chunking* (Laird, Rosenbloom & Newell, 1986) is a learning mechanism in Soar that generates new productions based on the results of problem space activity during impasses. It

compiles knowledge that is initially only available through activity at time scales of 1 second or more down to knowledge that is "immediately available" for use at the 10 ms time scale. It automatizes behavior (Schneider & Shiffrin, 1977) while speeding up overall performance. Although chunking, in combination with the flexibility of Soar's knowledge and problem solving, has been shown to yield a much wider range of learning behaviors than just automatization (Rosenbloom, 2006) – such as concept acquisition and episodic learning – speeding up behavior remains its most essential functionality. Responsibility for most other learning activities has been taken over by Soar 9's new learning mechanisms.

## 2 Factor Graphs

Factor graphs provide a form of divide and conquer with nearly decomposable components for reducing the combinatorics that arise with functions of multiple variables. The function could be a joint probability distribution over a set of random variables; e.g., $\mathbf{P}(V,W,X,Y,Z)$, which yields the probability of $V=v \wedge W=w \wedge X=x \wedge Y=y \wedge Z=z$ for every value $v$, $w$, $x$, $y$ and $z$ in the variables' domains. Or the function could represent a constraint satisfaction problem – e.g., $\mathbf{C}(A,B,C,D)$ – over a set of variables, yielding 1 if a combination of values satisfies the constraints and 0 otherwise. Or the function could represent a discrete-time linear dynamical system, as might typically be solved via a Kalman filter. The problem formulation here would involve a *trellis* structure, where the graph for one time step is repeated for each, with four variables per time step (*State*, *Input*, *Output* and *Noise*): $\mathbf{K}(S_0,I_0,O_0,N_0,\ldots, S_n,I_n,O_n,N_n)$ (Kschischang et al., 2001).

The prototypical factor graph operation is the computation of *marginals* on variables. For a joint probability distribution, this is simply the marginal distribution of a random variable, as computed by summing out the other variables; e.g., $\mathbf{P}(Y) = \Sigma_{v,w,x,z}\mathbf{P}(v,w,x,Y,z)$. The key to tractability is avoiding the explicit examination of every element of the cross product of the variables' domains. For probabilities this is achieved by decomposing the joint distribution into a product of conditional and prior probabilities over subsets of variables; e.g., $\mathbf{P}(V,W,X,Y,Z) = \mathbf{P}(V)\mathbf{P}(W)\mathbf{P}(X|V,W)\mathbf{P}(Y|X)\mathbf{P}(Z|X)$. Such decompositions derive from the *chain rule* plus *conditional independence* assumptions. A Bayesian network can then be generated from this equation by mapping each random variable into a node and each conditional probability distribution into a set of directed links from the conditioning variables to the conditioned variable (Figure 2). The marginal of a random variable is computed by summing out all other variables from this expression: $\mathbf{P}(Y) = \Sigma_{v,w,x,z}P(v)P(w)P(x|v,w)\mathbf{P}(Y|x)P(z|x)$. However, the commutative and distributive laws can first be utilized to improve the efficiency of this calculation by enabling variables to be summed out as soon as possible: $\mathbf{P}(Y) = \Sigma_x\mathbf{P}(Y|x)\Sigma_z P(z|x)\Sigma_v P(v)\Sigma_w P(x|v,w)P(w)$. The *belief propagation algorithm* performs this computation in Bayesian networks by passing messages among the nodes. This algorithm becomes *loopy belief propagation* when there are cycles in the graph. The *sum-product algorithm*, which subsumes loopy belief propagation, gets its name from the fact that it computes sums of products.



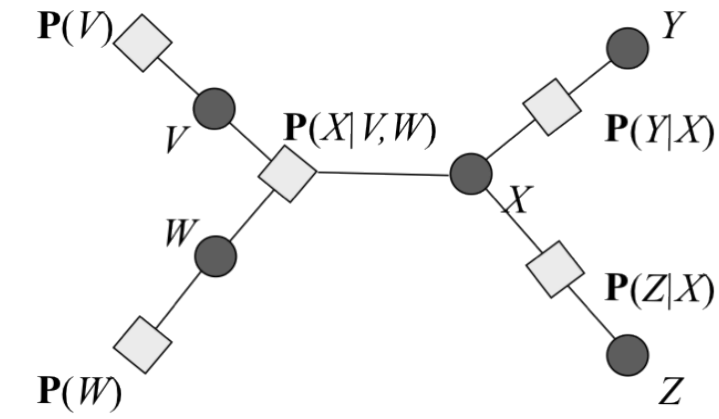

$$\mathbf{P}(V,W,X,Y,Z) = \mathbf{P}(V)\mathbf{P}(W)\mathbf{P}(X\,|V,W)\mathbf{P}(Y\,|X)\mathbf{P}(Z\,|X)$$

Figure 2. Sample Bayesian network.

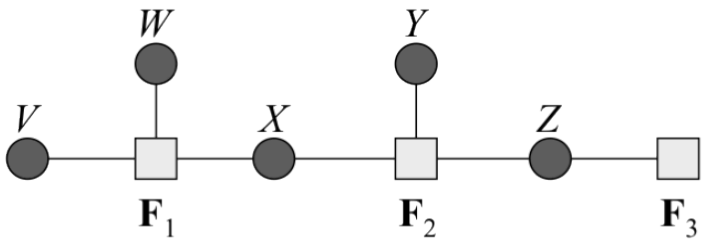
Factor graphs generalize the type of function decomposition seen in Bayesian networks to

arbitrary multivariate functions; e.g., $\mathbf{F}(V,W,X,Y,Z) = \mathbf{F}_1(V,W,X)\mathbf{F}_2(X,Y,Z)\mathbf{F}_3(Z)$. They get their name from the fact that each subfunction call is a *factor* in the resulting multiplicative expression. As with Bayesian networks, a correct decomposition here still depends on independence assumptions being met by the decomposition, but the resulting graphs are now *bipartite*; that is, they are composed of two classes of nodes, where all edges link one node of each type. A *variable node* is defined for each variable in the original function and a *factor node* for each factor in the resulting expression. Each factor node incorporates a *factor function* for the subfunction that maps onto the node. An undirected link is added between a variable node and a factor node whenever the corresponding variable is part of the corresponding factor. Figure 3 shows what the Bayesian network in Figure 2 looks like as a factor graph, along with the factor graph for the example of a more general function decomposition.

The sum-product algorithm operates by passing messages along links in factor graphs. Messages propagate in both directions along the links, from variable nodes to factor nodes and from factor nodes to variable nodes. All messages on a link, irrespective of direction, provide information about the values of the function variable mapped to the link's variable node; e.g., by specifying a probability distribution over the variable's domain elements. Each node computes a distinct output message for each of its neighbors, based on the input messages the sending node has itself received from all of its neighbors except for the one that is to receive the output message. An output message from a variable node is computed as the *pointwise product*[1] of the relevant input messages. An output message from a factor node starts with this same pointwise product of the relevant input messages, but also then includes the factor node's function in the product. All variables other than the one mapped to the receiving variable node are finally summed out to form the factor node's output message. A key optimization here, as in Bayesian networks, is to use the commutative and distributive laws to redistribute multiplicative factors outside of summations.




(a) $\mathbf{P}(V,W,X,Y,Z) = \mathbf{P}(V)\mathbf{P}(W)\mathbf{P}(X\,|\,V,W)\mathbf{P}(Y\,|\,X)\mathbf{P}(Z\,|\,X)$




(b) $\mathbf{F}(V,W,X,Y,X) = \mathbf{F}_1(V,W,X)\mathbf{F}_2(X,Y,Z)\mathbf{F}_3(Z)$

Figure 3. Sample factor graphs.

For tree-structured factor graphs in which only a single marginal is desired, the graph can be reduced to an *expression tree* in which the products and sums are computed in one pass from the

[1] A pointwise product of two vectors is akin to an inner product, with the corresponding values at each point of the two vectors being multiplied, but instead of then summing over these individual products to yield a single inner product, the products are assembled into a new vector that is the pointwise product.

leaves towards the root of the tree. Beyond this simplest case the algorithm must operate iteratively, repeatedly sending output messages from nodes as they receive new input messages. For *polytrees*, which have at most one path between any two nodes, this iterative algorithm always terminates and yields the correct answer. For arbitrary graphs with loops, neither correct answers nor termination are guaranteed. However, the algorithm does often work quite well in practice, as has been most strikingly evident for turbo codes.

The sum-product algorithm utilizes two specific arithmetic operations: *sum* and *product*. However, the same generic algorithm works for any pair of operations forming a *commutative semi-ring*, where both operations are associative and commutative and have identity elements, and the distributive law exists. *Max-product*, for example, is key to computing maximum a posteriori (MAP) estimations. *Or-and* also works, as do a range of other operation pairs.

To improve the efficiency of the sum-product algorithm, additional optimizations can also be applied, and alternative algorithms can be used (such as survey propagation (Mézard, Parisi & Zecchina, 2002) and Monte Carlo sampling (Bonawitz, 2008)). A connection exists between factor graphs and statistical mechanics, revealing that the sum-product algorithm minimizes the *Bethe free energy* and yielding further algorithmic innovations (Yedidia, Freeman & Weiss, 2005).

## 3 Reimplementing Soar's Elaboration Cycle via Factor Graphs

The core computational process in Soar's elaboration cycle is production match, as implemented by the Rete algorithm. Rete consists of a discrimination network for sorting working memory elements to matching production conditions; a join network to determine which combinations of working memory elements yield production instantiations while respecting across-condition variable equality constraints; and support for both incremental match across cycles and shared match across productions. Most individual productions match efficiently, although worst-case match cost is exponential in the number of conditions.

The purpose of this experiment was to determine whether graphical models in general, and factor graphs in particular, could straightforwardly implement a state-of-the-art production match algorithm – i.e., one that is as good as or better than Rete – and in so doing: (1) expand the known repertoire of graphical models to a form of symbol processing that is used in many cognitive architectures and that is absolutely central to Soar; and (2) establish the plausibility of the use of rules in an architecture based on a graphical implementation level. Although the functionality described in this section does not go beyond what already exists in Soar, such an experiment is an essential enabling step for the whole research program.

A mapping of Rete onto factor graphs exists on paper, in which factor nodes handle discriminations and joins, variable nodes effectively represent both the working memory elements that match production conditions and the instantiations that match combinations of conditions – analogous to Rete's $\alpha$ and $\beta$ *memories*, respectively – and unidirectional message passing over an expression tree enables incremental and shared match. However, rather than beginning with the assumption that Rete is what should be (re)implemented, it makes sense to start from the beginning with the semantics of production match as a multivariate function and see what form of factor graph results from decomposing it.

Consider the rule in Figure 4, for simple color inheritance. This is not exactly Soar's representation, although it does retain Soar's object-attribute-value representation for working memory elements with conditions testing these elements via constants and variables (denoted in angle brackets). The match process for this production can be represented as a Boolean function

```
P1: Inherit Color
    C1: (<v0> ^type <v1>)
    C2: (<v1> ^color <v2>)
    →
    A1: (<v0> ^color <v2>)
```

Figure 4. Sample color inheritance rule.

of the three production variables – $\mathbf{P_1}(v_0,v_1,v_2)$ – which, for each combination of variable values, yields 1 or 0 depending on whether or not the combination defines a legal instantiation of the conditions.

The question then becomes how to decompose this function into a product of subfunctions. The most obvious way is to model each condition as a factor, yielding the decomposition equation $\mathbf{P_1}(v_0,v_1,v_2) = \mathbf{C_1}(v_0,v_1)\mathbf{C_2}(v_1,v_2)$ and the graph in Figure 5. In such a graph, working memory is a 3D Boolean array – objects × attributes × values – with 1s for every element in working memory and 0s elsewhere; and messages are Boolean vectors with 1s for valid bindings of the link's variable and 0s elsewhere. In essence, productions define graphs while working memory defines distributions over graph variables.

Although useful as a didactic device, for introducing the core idea of how to decompose the match function and map it onto a graph, such an approach is fundamentally naïve, failing even to guarantee that the resulting graph will produce correct results. The core of the problem is that it ignores the essential correctness criterion for such function decompositions, that the product of the factors must yield the same result as would the original function, and thus misses a major constraint on how functions can be decomposed for appropriate implementation in factor graphs. The specific issue in this case is one of *binding confusion*, caused by independently tracking the legal bindings of each variable – called *instantiationless match* – rather than maintaining Rete's explicit combinations of condition instantiations (Tambe & Rosenbloom, 1994). Suppose `(A ^type B)`, `(C ^type D)`, `(B ^color Red)` and `(D ^color Blue)` are in working memory. The match binds $v_0$ to `A` & `C`, $v_1$ to `B` & `D`, and $v_2$ to `Red` & `Blue`, but it can't, for example, distinguish which color ($v_2$) to associate with object `A` ($v_0$), even though a correct match requires `Red` rather than `Blue`.

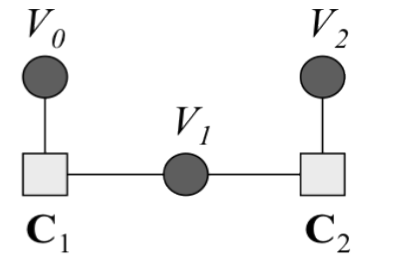



Figure 5. Sample rule graph.

To solve this problem we could conceivably post-extract correct instantiations from the graph (Dechter & Pearl, 1987), or explore a Rete-like instantiation-based approach to match, or devise a different way of decomposing the function, or even redefine the match so as to produce what is yielded by this decomposition. The approach actually taken has been to redefine the function's variables so as to yield correct match results from an instantiationless match over a condition-based decomposition. The key idea is for individual variable nodes in the graph to represent cross products of production variables, as needed to track combinations of variable bindings, rather than individual production variables. The resulting function, along with its decomposition and graph, can be seen in Figure 6. A factor node has been added to the graph for the production's

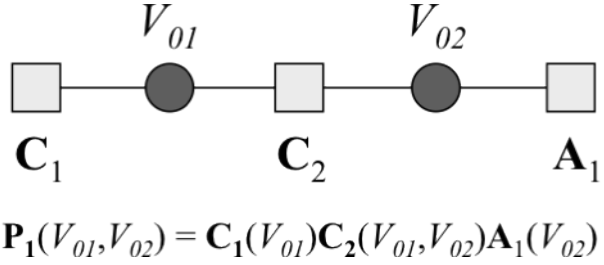



Figure 6. Modified rule graph.

action, to act as a consumer of the variable bindings generated by the condition match, and the three production variables have been replaced by two cross-product variables: $v_{01}$ for the combination of $v_0$ and $v_1$, and $v_{02}$ for the combination of $v_0$ and $v_2$. Each possible binding of such a cross-product variable corresponds to a combination of a binding for each of the production variables of which it is composed. For example, one possible binding of $v_{02}$ is `A`/`Red`. This combined element would only be assigned a value of 1 if $v_0$ and $v_2$ could compatibly be assigned `A` and `Red`, respectively.

This approach eliminates binding confusion by explicitly representing the needed combinations of bindings for action variables, while still avoiding creation of Rete's full production instantiations. In the process, it alters the worst-case match cost for a production from exponential in the number of conditions, as in Rete, to exponential in its *treewidth*; that is, the maximum number of production variables at a node. It also avoids the redundancy, along with the resulting extra cost and potential confusion, engendered when Rete creates instantiations that differ in values of variables appearing only in conditions while the values of variables appearing in actions are identical. For expression trees, this approach also appears capable of incorporating Rete's other key optimizations, such as sharing of work across productions and cycles, but the situation is less clear for loopy graphs.

To map a production into a graph via this approach, we start by imposing an ordering on the production's conditions and actions to yield a sequence of factor nodes, one for each condition and action. A variable node is then added between each successive pair of factor nodes. Finally, to determine which production variables are combined in each variable node, the first and last condition or action that uses each production variable is determined, and the variable is added to each variable node between the corresponding factor nodes. This approach is based on the idea of *stretching* in factor graphs, which itself maps onto junction trees (Kschischang et al., 2001).

Since individual variable nodes in the graph may now represent multiple production variables, messages can be multi-dimensional arrays that are expensive to process without further optimization. The most critical optimization here is the kind of factor rearrangement that is enabled by the distributive law. Without it, the full factor graph for the rule in Figure 4 – comprising 8 factor nodes and 8 variable nodes after several other less significant conceptual changes have been made to it – exhausts heap space before match completes (in LispWorks PE). With factor rearrangement, match takes only 1.7 sec. A second critical optimization leverages the uniformity of WM and message arrays – they are almost all 0s or 1s – via an N-dimensional generalization of *region quad/octrees* called *exptrees*, which are akin to the CPT-trees used in Bayesian networks (Boutilier, Friedman, Goldszmidt & Koller, 1996). If an array is uniform, it becomes a single-valued unit. Otherwise, each dimension is bisected to yield $2^N$ sub-arrays, and the process recurs. Implementing sum and product is trickier over this kind of a representation, but has been done. With this optimization, match time is reduced by a further factor of ~7 (from 1.7 to .25 sec). Using exptrees enables match of the rule in Figure 4 to complete even without factor rearrangement, but it then still requires 132 sec. This implies that factor rearrangement, at least for this rule with exptrees, speeds up match by a factor of ~500. It is too early to worry about the absolute magnitude of these match times, as this is still largely unoptimized research code (a more recent implementation, for example, is showing match times that are two orders of magnitude faster than this). What is relevant though are the relative times.

One interesting implication of representing working memory as an exptree is that it effectively becomes a piecewise constant function. Extending this to piecewise linear functions, as is being investigated in follow on work, holds the potential for effectively and uniformly

handling the kinds of continuous variables that are necessary for the processing of probabilities and signals. This in turn contributes to the hope that a single representation and inference algorithm will prove capable of spanning the processing of symbols, probabilities and signals.

## 4 Revisiting Soar's Decision Cycle via Alchemy

Soar's decision cycle, comprising an elaboration phase and a decision, is the lowest level at which knowledge may affect decisions, at which multiple fragments of knowledge may combine, and at which k-search may involve more than one cycle of match and firing. It is also the key scale at which extending Soar beyond symbol processing could lead to radically expanded functionality and at which it first makes sense to consider incorporation of Soar 9's diversity. The goal here is threefold, to: (1) investigate whether an existing mixed language that uses graphical models to combine general symbolic and probabilistic reasoning might provide a quicker path to the desired implementation level than is possible by starting from scratch with factor graphs; (2) see what insight can be obtained from such a system about incorporating probabilities deeply into Soar's decision cycle; and (3) begin exploring how to leverage graphical models to expand from uniform Soar's production-based elaboration phase to a more uniform implementation of Soar 9's multiple-memory approach.

Alchemy, which combines first-order logic and Markov networks in the form of *Markov logic*, was selected for this experiment. It supports forms of both symbolic and probabilistic processing along with nascent signal processing (Wang & Domingos, 2008), and was earlier explored within the Icarus cognitive architecture (Langley & Choi, 2006) with a specific focus on the implementation of inference (Stracuzzi, 2009). It also has been suggested as a uniform interface layer for AI systems (Domingos & Lowd, 2009). In Alchemy, a *Markov logic network* (MLN) is first defined via first-order predicates and formulas, with weights assigned to the formulas; for example, the Alchemy sentence "`(Color(o1,c)^Type(o2,o1)) => Color(o2,c)`." expresses simple color inheritance, à la Figure 4, while the sentence "`10 Color(F, Green)`" states that the particular object F is green with a weight of 10. The MLN is then compiled into a *ground Markov network* with binary nodes for each ground predicate, such as the fact that F is green; links among nodes that appear in common formulas, such as would be generated between `Color(F,Green)` and `Type(E,F)` by their co-occurrence in at least one grounding of the implication; and features for each possible ground formula. Graphical inference is performed on this ground Markov network, unless additional optimizations such as laziness (where grounding only occurs for variables that take on non-default values (Poon, Domingos & Sumner, 2008)) or lifting (where multiple ground nodes are combined into single nodes when they can be guaranteed to pass the same messages during belief propagation (Singla & Domingos, 2008)) are included.

Several small-scale experiments have been run, including: (1) reimplementing the simple production systems that were previously implemented via factor graphs; (2) adding a simple form of semantic long-term memory to the production memory; and (3) exploring an implementation of the eight puzzle, one of the earliest tasks investigated in Soar (Laird & Newell, 1983) and the basis for early learning experiments with it (Laird et al., 1986). Several of these experiments have also been replicated with BLOG, but the results are not fundamentally different from those described here based on Alchemy. Because the details of these experiments are not terribly interesting, the strategy here is to ignore the nitty-gritty and instead focus on the high level lessons learned from, and issues exposed by, the experiments.

The first lesson is that it is possible to implement a simple mixed decision cycle in this fashion, yielding an elaboration phase that combines Soar's standard rule-based capabilities with probabilistic reasoning and semantic memory (fact storage). Rules and facts become part of a Markov logic network. Since Alchemy provides the full expressibility of first order logic, rules can be represented via the approximately equivalent logical construct of conditional formulas; for example, the color-inheritance sentence above can approximate the rule in Figure 4. Facts become ground predicates in the network, as with the one presented above for the color of object F. Probabilities can then be added as weights on these sentences, as in the color fact. The state of working memory at the beginning of the decision cycle is not part of the Markov logic network. It instead acts as *evidence* that clamps the values of particular variables in the network. Alchemy compiles the Markov logic network and its associated evidence into a ground Markov network that has nodes corresponding to working memory elements and facts in semantic memory; and links corresponding to co-occurrence of these elements in production instantiations. This ground Markov network effectively provides a graph for the entire elaboration phase, rather than just for a single elaboration cycle, enabling a single invocation of Alchemy's inference procedure to compute the results of an entire phase.

One major issue with this approach to the implementation level though concerns its uniformity and minimality. In Alchemy, match actually occurs during the compilation of the (first-order) Markov logic network to the ground Markov network, rather than proceeding via the kind of within-network message passing that occurred in the factor graph implementation. In essence, the Markov logic network corresponds to the definition of the production system while the ground Markov network corresponds to working memory elements (the ground nodes) and production instantiations (the ground formulas). In contrast to the factor graph implementation, working memory elements are represented by distinct nodes in this network rather than simply serving as the basis for messages among nodes. Given the ultimate goal of combining functionality with uniformity, the inhomogeneity resulting from this match-as-compilation approach raises a significant theoretical issue at the implementation level. To attempt to resolve it, subsequent work focuses back on factor graphs, where the question can be asked as to whether it is possible to unify match – i.e., the computation of ground instances from first-order formulas – with the other necessary forms of probability and signal processing into a single graph that is processed in a uniform manner, or whether it will be necessary to go with something like Alchemy's dual graph/network approach in which match occurs via a first-order graph that generates a ground graph within which probabilistic inference occurs.

Despite this fundamental difference in how match is implemented, it turns out that exptrees serve a role in the factor graphs that is analogous to the use of laziness and lifting in Alchemy. The latter mechanisms eliminate unnecessary computation, either by avoiding the processing of default values or by grouping items that can be treated the same. With exptrees, defaults are identified naturally and neighboring items are grouped by region if their values are identical. Exptrees appear to be a coarser approach, but it may ultimately be possible to bring these ideas more into alignment as differences between them are better understood.

The other major issue worth mentioning is a broad one concerning search, times scales, locality of processing, and non-monotonicity. Systems like Alchemy that combine general first order reasoning with probabilistic inference perform global propagation of information in their knowledge structures in service of reaching a global minimum in a solution space. In other words, they are engaged in search; and, in particular, combinatoric search that can easily get stuck in local minima (Stracuzzi, 2009). This clearly has the essential character of problem

space search rather than knowledge search, although it has been mapped here onto Soar's decision cycle. Even though there is no distinction between k-search and ps-search in such systems, this mapping suggests that perhaps there should be. Combinatoric search needs to be controlled by knowledge, which itself must be processed in a more tractable fashion. This leads to the hypothesis that Alchemy, and in fact this whole class of systems, might be more effective in solving real problems if they bounded what they tried to compute within a single settling of the graph – essentially only striving for local minima based on global propagation of information – while adding an explicit capability for controlled search over a sequence of such local minima to find a global minimum.

Taking this a step further, such an approach enables assigning simple yet principled functional characterizations to the three time scales at which Soar is defined: activity in a problem space ($\geq$ 1 sec) finds global minima; the decision cycle (100 ms) finds local minima based on global information propagation; and the elaboration cycle (10 ms) only provides local propagation of information. Spatially, the 10 ms scale corresponds to neural circuits, so this last choice is plausible from this perspective. Such an assignment of functionality provides a simple yet powerful view of these cognitive scales that could be applicable to any cognitive architecture. It also serves as an excellent example of how research at the implementation level can yield new theoretical hypotheses about the architecture.

Beyond merely stating these hypotheses, it is also possible to examine their implications on particular architectures, such as Soar. Any lack of consistency with this mapping of functionality onto architecture levels may reveal incoherence in the architectural definition; and in Soar it does in fact reveal some potential issues. The biggest issue concerns the use of working memory to enable global communication across elaboration cycles. If the mapping is correct, then this is too powerful for this time scale. Global communication instead should be limited to decision cycles. Interestingly, although rules are used in many architectures, this conflict only exists in those like Soar, where rules map onto the 10 ms level. In most architectures rules map onto the 100 ms level, where global communication is unproblematic. In ACT-R, for example, rules are the objects of selection, and execute sequentially. It has been known for some time that ACT-R's rules map functionally onto Soar's operators rather than Soar's rules, with Soar's rules mapping onto ACT-R's subsymbolic processing rather than its rules. Yet, prior to this analysis there was no reason to select between these two distinct alternative levels for rules. It is important to note though that this analysis does not suggest an outright ban on rules at the 10 ms level, only global rather than local communication among them.

One less momentous aspect of this issue concerns non-monotonic reasoning. Architectures such as Soar embody non-monotonic reasoning in various forms. Operator application, for example, is inherently non-monotonic in the changes it makes to the current state. Soar also maintains an implicit closed world assumption with respect to working memory – that anything not present is false – enabling negated conditions to implement negation as failure, a form of default reasoning. Non-monotonic reasoning is difficult in general in a first order reasoner such as Alchemy. It is also problematic within rules from a levels perspective because it is implicitly global. Default reasoning, for example, draws conclusions from a lack of evidence; that is, based on a global assumption about what does not exist. Operator application in Soar does not engender a level conflict because it occurs at the 100 ms level, but negated conditions do because they are defined at the level below.

If we now look beyond the elaboration phase to the decision process, there is a need to combine comparative information about alternatives to decide on one. Limited experiments with

such decisions have also been performed, leveraging Alchemy's provision of weights on formulas to encode comparative information, and most-probable-explanation (MPE) inference to select operators based on this information. Although this has proven adequate for simple examples, developing a full decision mechanism is left for future work.

## 5 Conclusion and Next Steps

The ultimate goal for work in cognitive architecture should be architectures that are comprehensive models of human cognition and/or effective bases for constructing human-like artificial intelligence. To reach this goal, it is important to resolve the diversity dilemma, enabling the development of architectures that are functional and efficient yet elegant, minimal, extensible, integrable and maintainable. This article proposes achieving this by combining architectural diversity with a uniform implementation level based on graphical models, and then exploring this combination through experiments with existing and new architectures. The particular focus in this article has been on a uniform graphical implementation of a hybrid mixed variant of the Soar 9 architecture. Experiments have been described on reimplementing Soar's lowest two architectural levels – the elaboration and decision cycles – via graphical models. The former shows how graphical models can yield a match algorithm for productions with an improved per-rule worst-case bound over the state-of-the-art Rete algorithm. The latter demonstrates the beginnings of a mixed decision cycle and the addition of a simple semantic memory. It also yields new insights into the structure of Soar, and what will be necessary for a truly uniform graphical implementation of a Soar-like architecture.

Many outstanding issues remain with these partial reimplementations, but they start to reveal the potential of a uniform, graphical implementation level to support diverse cognitive architectures. Current work is focused on completing a uniform implementation of a hybrid mixed decision cycle based on factor graphs that is capable of combining symbol, probability and signal processing to integrate together procedural knowledge, declarative knowledge (e.g., semantic and episodic memories) and perceptual knowledge. Learning from the experiments with Alchemy, and the resulting analysis, this work is paying careful attention to locality of computation. It is, for example, investigating how to keep the elaboration cycle a local rather than a global process, by directly linking actions of some productions with conditions of others, rather than going through a global working memory. There is also a major focus on understanding how to perform production match and probabilistic processing compatibly via message passing, rather than relegating the former to a precompilation phase.

Future work will include reimplementing much of the rest of Soar 9 – including semantic and episodic memories, imagery, reflection and learning – while enhancing it with additional capabilities, such as decision-theoretic planning, Markov decision processes, and theory of mind. Although such extensions may expand the diversity even further at the architectural level, the goal will always be to unify them all through the common graphical implementation level. Beyond Soar, it will also be essential to explore: reimplementation of other leading architectures, as well as hybrids among them; new architectures that are more directly inspired by the uniform multipotency of a graphical implementation level; and the potential of graphical models to bridge the gap between symbolic and neural architectures.

## Acknowledgments

This article is based on three papers published during 2009, a conference paper that introduced the cognitive hourglass and focused on a graphical reimplementation of the elaboration cycle

(Rosenbloom, 2009a), a workshop paper that explored a mixed elaboration phase via Alchemy (Rosenbloom, 2009b), and a symposium paper that introduced the diversity dilemma (Rosenbloom, 2009c). This body of work was made possible by sabbatical support from the USC Viterbi School of Engineering plus funding from USC's Institute for Creative Technologies (ICT). ICT's Cognitive Architecture Working Group has been invaluable for semi-public exploration of these ideas. My attention was first attracted to factor graphs by (Kschischang et al., 2001), which also served as the basis for much of the general material here on them. I would also like to thank the Alchemy group at the University of Washington for their help in installing Alchemy and working through various issues that arose during experimentation with it; and John Anderson, Rick Cooper and Dario Salvucci for helpful comments on an earlier draft of this article.

## References

Anderson, J. R. (1993). *Rules of the Mind*. Hillsdale, NJ: Erlbaum.

Bonawitz, K. A. (2008). *Composable Probabilistic Inference with Blaise* (Doctoral dissertation). Department of EECS, MIT, Cambridge.

Boutilier, C., Friedman, N., Goldszmidt, M. & Koller, D. (1996). Context-specific independence in Bayesian networks. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence* (pp. 115-123).

Cho, B., Rosenbloom, P. S. & Dolan, C. P. (1991). Neuro-Soar: A neural-network architecture for goal-oriented behavior. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society* (pp. 673-677). Erlbaum.

Cooper, R. & Fox, J. (1998). COGENT: A visual design environment for cognitive modeling. *Behavior Research Methods, Instruments, & Computers*, *30*, 553-564.

Cooper, R., Fox, J., Farringdon, J. & Shallice, T. (1996). A systematic methodology for cognitive modeling. *Artificial Intelligence*, *85*, 3-44.

Dechter, R. & Pearl, J. (1987). Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, *34*, 1-38.

Deering, S. (1998). Watching the waist of the protocol hourglass, Keynote at ICNP '98.

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

Domingos, P. & Lowd, D. (2009). *Markov Logic: An Interface Layer for Artificial Intelligence*. San Rafael, CA: Morgan & Claypool.

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

Frey, B. J. (2003). Extending factor graphs so as to unify directed and undirected graphical models. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence* (pp. 257-264).

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

Jilk, D. J., Lebiere, C., O'Reilly, R. C. & Anderson, J. R. (2008). SAL: An explicitly pluralistic cognitive architecture. In *Journal of Experimental and Theoretical Artificial Intelligence*, *20*, 197-218.

Jordan, M. I. & Sejnowski, T. J. (2001). *Graphical Models: Foundations of Neural Computation*. Cambridge, MA: MIT Press.
Koller, D. & Friedman, N. (2009). *Probabilistic Graphical Models*. Cambridge, MA: MIT Press.
Kschischang, F. R., Frey, B. J. & 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 1st AGI Conference.* IOS Press.
Laird, J. E. & Newell, A. (1983). A universal weak method: Summary of results. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence* (pp. 771-773).
Laird, J. E. & Rosenbloom, P. S. (1996). The evolution of the Soar cognitive architecture. In D. M. Steier & T. M. Mitchell (Eds.) *Mind Matters: A Tribute to Allen Newell*. Mahwah, NJ: Erlbaum.
Laird, J. E., Rosenbloom, P. S. & Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11-46.
Langley, P. & Choi, D. (2006). A unified cognitive architecture for physical systems. In *Proceedings of the 21st National Conference on Artificial Intelligence* (pp. 1469-1474).
Lathrop, S. D. & Laird, J. E. (2007). Towards incorporating visual imagery into a cognitive architecture. In *Proceedings of the 8th International Conference on Cognitive Modeling*.
Li, S. Z. (2009). *Markov Random Field Modeling in Image Analysis*. London: Springer.
Marinier, R. P. & Laird, J. E. (2004) Toward a comprehensive computational model of emotions and feelings. In *Proceedings of the 6th International Conference on Cognitive Modeling*.
McCallum, A., Rohanemanesh, K., Wick, M., Schultz, K. & Singh, S. (2008). FACTORIE: Efficient probabilistic programming via imperative declarations of structure, inference and learning. In *Proceedings of the NIPS Workshop on Probabilistic Programming.*
Mézard, M., Parisi, G. & Zecchina, R. (2002). Analytic and algorithmic solution of random satisfiability problems, *Science*, *297*, 812-815.
Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D. L. & Kolobov, A. (2007). BLOG: Probabilistic models with unknown objects. In L. Getoor & B. Taskar (Eds.) *Introduction to Statistical Relational Learning*. Cambridge, MA: MIT Press.
Nason, S. & Laird, J. E. (2005). Soar-RL: Integrating reinforcement learning with Soar. *Cognitive Systems Research*, *6*, 51-59.
Newell, A. (1990). *Unified Theories of Cognition*. Cambridge, MA: Harvard Press.
Nuxoll, A. & Laird, J. E. (2004). A cognitive model of episodic memory integrated with a general cognitive architecture. In *Proceedings of the 6th International Conference on Cognitive Modeling*.
Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufman.
Pearson, D. J., Gorski, N. A., Lewis, R. L. & Laird, J. E. (2007). Storm: A framework for biologically-inspired cognitive architecture research. In *Proceedings of the 8th International Conference on Cognitive Modeling*.
Poon, H., Domingos, H. & Sumner, M. (2008). A general method for reducing the complexity of relational inference and its application to MCMC. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence* (pp. 1075-1080).
Rosenbloom, P. S. (2006). A cognitive odyssey: From the power law of practice to a general learning mechanism and beyond. *Tutorials in Quantitative Methods for Psychology, 2*, 43-51.

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 Multirepresentational Architectures for Human-Level Intelligence.*
Rosenbloom, P. S. Laird & J. E. Newell, A. (1993). *The Soar Papers: Research on Integrated Intelligence*. Cambridge, MA: MIT Press.
Rosenbloom, P. S., Newell, A. & Laird, J. E. (1991). Towards the knowledge level in Soar: The role of the architecture in the use of knowledge. In K. VanLehn (Ed.) *Architectures for Intelligence*. Erlbaum.
Schneider, W. & Shiffrin, R. M. (1977). Controlled and automatic human information processing: I. Detection, search, and attention. *Psychological Review*, *84*, 1-66.
Singla, P. & Domingos, P. (2008). Lifted first-order belief propagation. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence* (pp. 1094-1099).
Stracuzzi, D. (2009). Personal communication.
Tambe, M. & Rosenbloom, P. S. (1994). Investigating production system representations for non-combinatorial match. *Artificial Intelligence*, *68*, 155-199.
Wang, J. & Domingos, P. (2008). Hybrid Markov logic networks. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence* (pp. 1106-1111).
Yedidia, J. S., Freeman, W. T. & Weiss, Y. (2005). Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, *51*, 2282-2312.