Advances in Cognitive Systems 6 (2018) 1–18 Submitted 06/2018; published 08/2018

# Exact, Tractable Inference in the Sigma Cognitive Architecture via Sum-Product Networks

**Himanshu Joshi** HJOSHI@USC.EDU
**Paul S. Rosenbloom** ROSENBLOOM@USC.EDU
Department of Computer Science, University of Southern California, Los Angeles, CA 90089 USA
Institute for Creative Technologies,12015 E Waterfront Drive, Los Angeles, CA 90094 USA

**Volkan Ustun** USTUN@ICT.USC.EDU
Institute for Creative Technologies,12015 E Waterfront Drive, Los Angeles, CA 90094 USA

## Abstract

Sum-product networks (SPNs) are a new kind of deep architecture that support exact, tractable inference over a large class of problems for which traditional graphical models cannot. The Sigma cognitive architecture is based on graphical models, posing a challenge for it to handle problems within this class, such as parsing with probabilistic grammars, a potentially important aspect of language processing. This work proves that an early unidirectional extension to Sigma's graphical architecture, originally added in service of rule-like behavior but later also shown to support neural networks, can be leveraged to yield exact, tractable computations across this class of problems, and further demonstrates this tractability experimentally for probabilistic parsing. It thus shows that Sigma is able to specify any valid SPN and, despite its grounding in graphical models, retain the desirable inference properties of SPNs when solving them.

## 1. Introduction

Cognitive architectures (Langley, Laird, & Rogers, 2009) model the fixed mechanisms underlying human-like cognition and yield cognitive systems when combined with the necessary knowledge and skills. The efficiency of such architectures has long been a major concern when there is significant anticipation of their being applied to large-scale and/or real-time problems. An early important example was the incorporation of the Rete match algorithm (Forgy, 1982) into the Soar architecture, when the rule system underlying Soar was switched from XAPS2 to OPS5 (Laird & Newell, 1983). Beyond this, there is also a more specific need for tractability, or even boundedness, in human-like cognitive architectures that stems from their need to be driven by a cognitive cycle that, in humans, runs at ~50 msec./cycle (Laird, Lebiere & Rosenbloom, 2017).

The Sigma cognitive architecture is being developed towards achieving a quartet of desiderata (Rosenbloom, Demski & Ustun 2016a). One concerns *sufficient efficiency*, the ability to execute quickly enough for whatever ends are to be accomplished, which hits squarely on the needs just mentioned. The other three desiderata are: *grand unification*, combining cognitive and key sub-cognitive abilities; *functional elegance*, enabling this diverse behavior from a core set of general mechanisms; and *generic cognition*, constructing artificial and modeling natural intelligence. The

overall approach to satisfying these desiderata in Sigma is based on the *graphical architecture hypothesis*, that "key to progress on them is combining what has been learned from over three decades' worth of separate work on *cognitive architectures* and *graphical models*."[1]

Many forms of graphs exist in cognitive architectures, from the discrimination networks found in Rete to the relational graphs (or semantic networks) that form the basis for the declarative long-term and working memories in some architectures (e.g., Anderson et al., 2004; Laird, 2012) to the neural networks used in others (e.g., Jilk et al., 2008; Sun, 2016). However, the graphical models used in Sigma are of a very specific sort, which leverage forms of independence to decompose arbitrary multivariate functions into products of simpler functions that can be mapped onto graphs for efficient computation of quantities such as marginals and modes (Koller & Friedman, 2009).

Such graphical models provide the dominant paradigm for probabilistic computation, in forms such as *Bayesian networks* (Pearl, 1988), but in the more general form of *factor graphs* (Kschischang, Frey, & Loeliger, 2001) – as used in Sigma – they have yielded state-of-the-art algorithms across the processing of signals, probabilities and symbols. It is this breadth and efficiency from a single underlying formalism that provides the core potential for achieving three of Sigma's four desiderata, with only generic cognition at this point seeming unrelated.

It has, however, recently been shown that there are significant classes of important problems – such as probabilistic parsing – for which such graphical models yield exponential time and only approximate answers whereas a new kind of deep architecture – *sum-product networks* (SPNs) – is able to solve them in an exact and tractable fashion (Poon & Domingos, 2011). This class of problems has since been enhanced further to include constraint satisfaction, optimization, and satisfiability, among others, by generalizing the ideas behind SPNs (Friesen & Domingos, 2016).

The core hypothesis examined in this paper is that Sigma, although grounded in traditional graphical models, already embodies all that is necessary to encode and solve any valid SPN with the tractability and exactness expected of them. This hypothesis will be approached by: (1) proving, with the aid of a translation algorithm, that Sigma's existing cognitive language is expressive enough to encode any valid SPN; (2) proving that an extension to Sigma that was originally developed to support Rete-like rule match (and actions) in Sigma, and which later proved key in implementing neural networks, can enable solving any valid SPN with the exactness and tractability expected of them; and (3) demonstrating experimentally that solving SPNs for probabilistic context free grammars (PCFGs) in Sigma retains the tractability and exactness implied by the inside-outside algorithm. These results bear directly on sufficient efficiency as well as functional elegance while introducing more generally a novel means of combining graphical models and SPNs. They may even provide a new route to solving the long-standing issue of limiting the cognitive cycle to tractable or even bounded inference (see, e.g., Tambe, Newell & Rosenbloom, 1990).

The rest of this article elaborates on these themes. Section 2 reviews Sigma along with a brief introduction to graphical models. Section 3 discusses sum-product networks and how they map onto Sigma. Included in this is more necessary detail on inference in graphical models, an algorithm for the mapping of SPNs onto Sigma, a proof that the algorithm works for any valid SPN, and a proof that the resulting SPN retains the expected exactness and tractability when solved in Sigma. Section 4 discusses probabilistic context free grammars, the challenges faced by a factor graph

[1] Although originally stated in this form, it is now actually over four decades worth.

representation of them, and experimental results from an SPN-based Sigma implementation that demonstrates the requisite tractability. Section 5 summarizes and concludes.

## 2. The Sigma Cognitive System

As implied by the graphical architecture hypothesis, Sigma is an approach to cognitive systems that reflects a number of lessons from earlier cognitive architectures. At a high level, Sigma borrows much from Soar. Early Soar was symbolic and functionally elegant, with one symbolic long-term memory and one symbolic learning mechanism (Laird, Newell & Rosenbloom, 1987). Soar has since changed to incorporate multiple long-term memories and learning mechanisms and several forms of subsymbolic representation (Laird, 2012). It also freely appends, as necessary, external modules when intensive subsymbolic processing is required. From Soar, Sigma has leveraged:

- the distinction between long-term memory and working memory;
- the use of problem spaces to structure cognitive behavior
- the cognitive cycle structure and its division into two major phases (Figure 1), although with differences in the details; and
- the functional elegance of its nested three-layer control structure (reactive via a single cognitive cycle, deliberative via a sequence of operator selections and applications over multiple cognitive cycles, and reflective via impasse-driven generation of metalevels).

In the process, Sigma has attempted to sustain the kind of functional elegance that was evident in the early Soar while providing the necessary diversity of long-term memory and learning behaviors via *idioms* above the architecture (Rosenbloom, Demski & Ustun, 2016a). Sigma has also opted for the more pervasive style of subsymbolic representation – or *quantitative metadata* (Laird, Lebiere, & Rosenbloom, 2017) – that is at least partially seen in ACT-R and the later versions of Soar, but via a strategy based on the broadly state-of-the-art approach of graphical models.

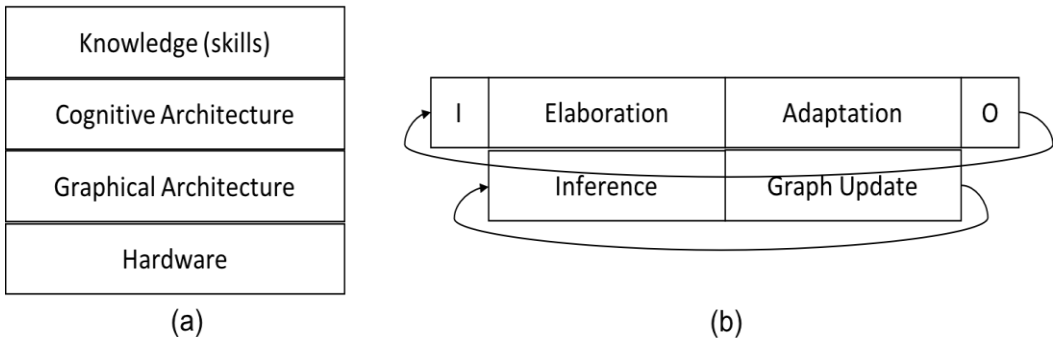


*Figure 1*. Sigma operationalizes the graphical architecture hypothesis in a layered format. In (a), the Cognitive architecture is supported by the graphical architecture below it. The cognitive architecture consists of the cognitive cycle and the cognitive language. The cognitive cycle is shown in (b) at the top with two major phases – elaboration and adaptation – corresponding to inference and update phases in the graphical architecture below. The minor phases *Input (I) & Output (O)* are also shown and are required for interfacing to the outside world.

Figure 1 shows the operationalization of the graphical architecture hypothesis via a layered design. Long-term knowledge fragments are specified via Sigma's cognitive language and then compiled to a *generalized* factor graph, representing the memory and structuring its reasoning. The resulting graph is then solved using a generalized form of the sum-product algorithm. A review of how Sigma has extended the sum-product algorithm is covered in Rosenbloom, Demski and Ustun (2016b). The remainder of this section describes Sigma's cognitive language, factor graphs, and the unidirectional extension to Sigma's factor graphs that enables SPN inference.

Sigma's cognitive language consists of *predicates* and *conditionals*. Predicates provide relational data structures for cognitive processing that join together *typed arguments* representing objects, entities or concepts, plus optional *functions*. Each predicate induces a region in the *working memory* (WM) for temporary storage of the state of the system and may also induce a region in the system's *perceptual buffer* for input from the outside world. The types themselves can be discrete – symbolic or probabilistic – or continuous, thus allowing the predicates to conjointly represent richly structured representations that are both mixed (symbolic + probabilistic) and hybrid (discrete + continuous).

Relationships between predicates are specified using *conditionals* that represent generalized knowledge fragments in Sigma's long-term memory (LTM) by blending concepts from rule-based systems and probabilistic networks. They are built from predicate patterns – *conditions*, *actions*, and *condacts* – plus optional *functions*. Conditions and actions are analogous in their behavior to the respective parts of rules and provide the forward momentum characteristic of procedural memory – conditions match to evidence in working memory and actions generate proposed changes in WM. Condacts support bidirectional processing – both matching to WM and suggesting changes to it – as needed for general probabilistic reasoning and traditional factor-graph semantics. The results of matching condacts and conditions within conditionals are combined in a multiplicative manner, while reuse of the same actions for the same predicate, either within or across conditionals, results in an additive combination.

Factor graphs and the message passing *sum-product algorithm* (Kschischang, Frey, & Loeliger, 2001) are used to ground the knowledge and processing thus specified. As seen in Figure 2, factor graphs are bipartite graphs composed of *variable nodes* and *factor nodes*. There is a variable node for each variable in the global function and a factor node for each sub-function in its decomposition. Each variable node is connected to all the factor nodes in whose function it participates. The sum-product algorithm operates by computing messages along these links in both

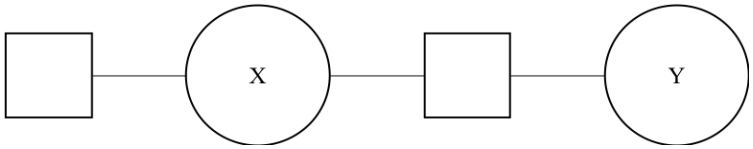


*Figure 2*. A factor graph for the function $f(X, Y) = f_1(X)f_2(X, Y)$, with two variables $X,Y$ shown in variable nodes and the two factors into which the function decomposes.

directions. A message along a link from a variable node is a product of messages to the node along all other links, while the message along a link from a factor node also multiplies in the local factor function.

Although the bidirectionality of the sum-product algorithm, as supported by condacts in Sigma's cognitive language, is required in order to meet the semantics of graphical models, certain forms of processing, such as for rules in procedural memories, require flow of information in one direction only. This limitation was overcome early on in Sigma by providing the option for unidirectional message passing along particular links, as specified in the cognitive language by using conditions and actions in conditionals. This breaks the traditional factor graph semantics for the affected portions of the overall graph, but it increases the scope of what can be represented and computed, including enabling rule conditions (and actions) to map onto the generalized factor graphs produced by Sigma's compiler in ways that mimic the Rete algorithm (Rosenbloom, 2010). It has also since been shown to enable mapping of feedforward neural networks onto portions of these graphs (Rosenbloom, Demski & Ustun, 2016b).

One other rule-based extension to Sigma's factor graphs is also leveraged in this work: messages converging on a single node from multiple actions for the same predicate are combined via summation rather than multiplication (Rosenbloom, Demski & Ustun, 2016b). Use of this extension is not logically necessary to encode SPNs exactly and tractably in Sigma, as it is possible to instead directly leverage the sum aspect of the sum-product algorithm, but it simplifies the mapping from SPNs onto Sigma and is thus easier to understand. For simplicity, in the next section we will refer to these as sum nodes to contrast them with normal product nodes.

## 3. Sum-Product Networks and Their Mapping onto Sigma

Sum-product networks (SPNs) are a relatively new form of graph-based (actually tree-based in this case) computational model that represent complex functions via sums and products (Poon & Domingos, 2011). Although the name here is quite similar to that of the sum-product algorithm, the two are actually distinct in both history and usage, with the biggest difference being that inference in SPNs is guaranteed to be exact and linear in the size of the network. SPNs can encode any graphical model or factor graph where inference is tractable. However, the converse is not true; a function that can be represented as an SPN does not necessarily lend itself to tractable, non-exponential inference when expressed as a factor graph. In fact, several classes of problems exist that can be solved tractably and exactly as SPNs, but that lose their exactness and tractability when represented as factor graphs (Demski, 2015; Gens & Domingos, 2013). Probabilistic context free grammars (PCFGs) provide one key example, for which in fact the SPN representation over a sentence of bounded length encodes the well-known inside-outside algorithm, supporting inference (inside) and a prerequisite for learning (outside) (Poon & Domingos, 2011). Friesen and Domingos (2016) have further generalized earlier results on SPNs to show that inference can remain tractable for a larger class of problems – characterized by summing over semirings – that "includes satisfiability, constraint satisfaction, optimization, integration, and others."

Factor graphs were chosen for Sigma because they subsume many different graphical models and narrow AI algorithms. Whereas SPNs focus on representing what computations are to be performed and how to perform these computations, factor graphs specify the goals of the

computation only, with the sum-product algorithm providing one possible algorithmic implementation. For this reason, the decision was made here to focus on direct extensions to the sum-product algorithm in implementing sum-product networks within Sigma rather than more indirectly finding a way of extending the semantics of factor graphs that would in turn guarantee that any algorithm for solving them would have the right properties. Here, in fact, we specifically leverage the sum-product algorithm with the two extensions mentioned in the previous section for unidirectional message passing and sum nodes.

In the rest of this section we prove that Sigma's cognitive language is able to specify any valid SPN while retaining the desirable inference properties of SPNs in the graphs that result from compiling these specifications into extended factors graphs. This is accomplished by first considering how inference works in factor graphs, which form the inner loop of the elaboration phase of the cognitive cycle, and in SPNs. An algorithm to convert an SPN into an equivalent set of Sigma conditionals is then presented. The effectiveness of this algorithm in producing Sigma conditionals that retain the desirable inference properties of SPNs is proven by showing that for the smallest valid SPN (and the corresponding Sigma model): (1) the posteriors calculated in the corresponding sum and product nodes in Sigma's working memory are the same as in the original SPN, (2) the messages incoming at these nodes are the same as in the original SPN, and (3) there are no cycles in the resulting Sigma graph. These properties can then be generalized over any valid decomposable SPN recursively created using the definition of SPNs presented by Gens and Domingos (2013).

### 3.1 SPNs and Graphical Models

We borrow notation from Darwiche (2009) and Koller and Friedman (2009) to describe SPNs and graphical models respectively. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graphical model with variable nodes in set $\mathcal{V}$ and edge nodes in edge set $\mathcal{E}$. Each variable node is a discrete[2] random variable $X$ and takes on values $x^i$ in its domain $\Delta(X)$. Figure 3 shows an example of a Bayesian network, the corresponding factor graph representation and the SPN corresponding to them.

An indicator variable typically takes on the value 1 if the supporting variable takes on the corresponding value. Here, we extend the definition of indicator variable as done in Gens and Domingos (2013). We define an indicator variable $\mathcal{I}(X^i)$ for every $x^i \in \Delta(X)$. $\mathcal{I}(X^i)$ takes on the value 1 if the corresponding variable $X$ is observed in evidence $E$ and it takes value $x^i$, or if $X$ is not observed as part of $E$.

$$\mathcal{I}(X^i) = \begin{cases} 1, & \text{if } X \text{ is in } E \text{ and } E \text{ indicates } X = x^i \text{ or } X \notin E \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

The networks shown in Figure 3 encode the distribution:

$$P(X,Y) = \sum_{X,Y} P(X)P(Y|X) \quad (2)$$

Here, P(X) is the prior on $X$ and P($Y$|$X$) is the conditional probability distribution on $Y$ given $X$. The factor graph is a bipartite representation $\mathcal{G} = (\mathcal{V}, \mathcal{F})$ where the variable nodes $\mathcal{V}$ correspond to

[2] We consider discrete random variables for the sake of simplicity; however, these concepts can be applied to continuous variables with suitable modifications.

variables and the factor nodes $\mathcal{F}$ encode functions over variables. The prior and the conditional distribution from Equation 2 are shown as local factors with the joint distribution being the global function represented by the graph. Inference is carried out via message passing, with Figure 3b showing the messages sent over the links from variable nodes. In particular, the factor graph in Figure 3b encodes the distribution:

$$P(X,Y) = \sum_{x^i \in \Delta(X), y^j \in \Delta(Y)} \mathcal{I}(X^i)\, \mathcal{I}(Y^j) P(X) P(Y|X) \tag{3}$$

The value of the factor graph is determined by the evidence provided, as applied by the definition of the indicator variables to perform inference via message passing. Here we present a brief description of the message passing algorithm using our notation and interpretation. The messages incident on variable nodes are from factor nodes and represent beliefs over the domains of the respective variables. Messages over different links at variable nodes are combined via multiplication. At the variable nodes, we provide evidence by multiplying the respective indicator variables with the local distribution over that variable. If the variable is observed as part of evidence, then this reduces the message to a non-zero value for the particular value of the variable observed. The calculation performed at the variable nodes is a pointwise product over the variable's domain and an update message is generated to other links. The outgoing message from a variable node to a factor node thus includes a product of all incoming messages except the one from that particular factor node:

$$\mu_{x \to f; x^i \in \Delta(X)} = \prod_{f \in \mathcal{F}\ that\ are\ neighbors\ of\ X \setminus \mu_{f \to x; x^i \in \Delta(X)}} \mathcal{I}(X^i) \mu_{f \to x; x^i \in \Delta(X)} \tag{4}$$

The factor nodes receive messages from variable nodes that are their neighbors and multiply these messages together along with their local functions. Outgoing messages to variable nodes are then generated by summing out the other variables not in this message:

$$\mu_{f \to x; x^i \in \Delta(X)} = \sum_{y} \prod_{y \in \mathcal{V} that\ are\ neighbors\ of\ f \setminus \mu_{x \to f; x^i \in \Delta(X)}} F(x,y) \mu_{y \to f; y^i \in \Delta(Y)} \tag{5}$$

This has the effect of selecting a particular state from each of the local factors and multiplying them to obtain the global function, consistent with any provided evidence. For the variables that are not observed, this has the effect of yielding their marginals at the respective variable nodes, given the evidence (Kschischang, Frey, & Loeliger, 2001). It is important to note that the number of calculations at the factor nodes is exponential in the number of variables in the largest factor and their domain size, in particular it is $O(m^n)$, where $m$ is the size of the largest variable domain and $n$ is the number of variables. It is also important to note that if certain states in the domains of the variables participating in that factor do not participate, the factor graph has no way of specifying that. An example of this will be seen later in the domain of PCFG parsing. In this example, the grammar itself specifies which sums and products are necessary, forgoing those that are not needed.

The factor graph shown in Figure 3b is a tree, but this overall procedure can be generalized to graphs with cycles. However, inference may not be exact in graphs with cycles and the cost is typically proportional to the *treewidth* of the graph – a measure of the connectedness of the graph. In such cases, the inference is approximate as well as exponential in nature. Since the solution of such graphs is at the very heart of Sigma's cognitive cycle, this raises not only efficiency issues,

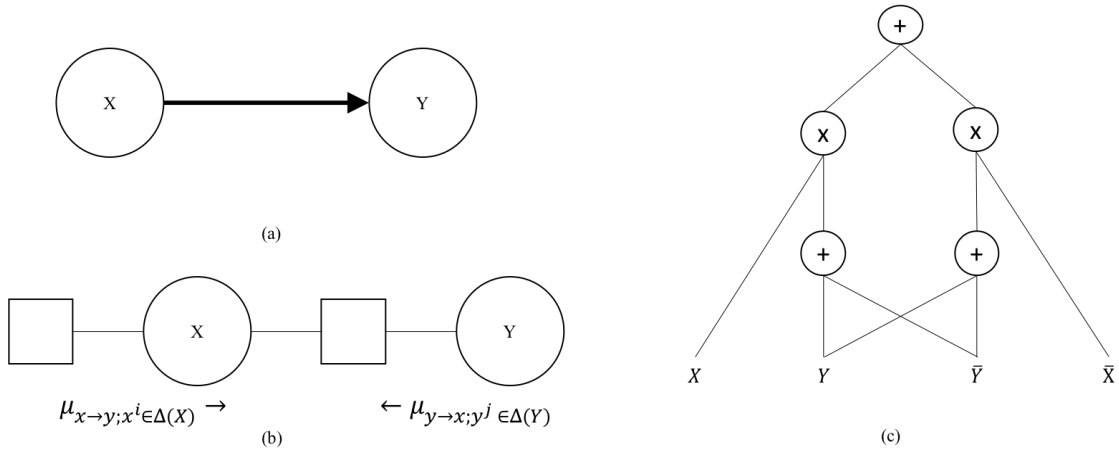


*Figure 3*. A Bayesian network shown in standard notation (a) and the corresponding factor graph (b). Each variable here is Boolean. The factor graph version shows the messages that are to be exchanged along each link. The corresponding SPN is shown in (c).

but also ones of tractability and boundedness. Even worse, there are major cognitive problems, such as the parsing of probabilistic context free grammars (PCFGs) – an important formalism characterizing linguistic structure (Jurafsky & Martin, 2008) that has been argued recently is important for cognitive architectures (Demski, 2015) – for which polynomial (cubic) algorithms are known, but which when mapped onto pure graphical models yield exponential computation (Pynadath & Wellman, 1998); and even then they only provide approximate solutions rather than the exact solutions provided by the polynomial algorithms. To work around this problem, Smith and Eisner (2008) and Naradowsky, Vieira, and Smith (2012) designed a special factor that embedded dynamic programming – i.e. the structure of the problem – inside the factor itself, allowing for exact inference. However, as shall be seen subsequently, inference remains inefficient in the size of the grammar and requires a special, non-standard message passing schedule.

SPNs avoid these problems by being able to specify only the operations that are needed to compute the marginal efficiently. This is achieved via the use of a network polynomial (Darwiche, 2009) and postulating hidden variables to express the network polynomial efficiently. An SPN is a tree rooted in a sum node, encoding a partition function over a probability distribution. More formally, we use the definition of a decomposable SPN from Gens and Domingos (2013) because it helps show how to compose SPNs from basic elements. Consider a set of variables $\mathcal{X} = \{X_1, X_2, \ldots, X_N\}$ and let $\mathcal{X}_K$ be partition such that $\mathcal{X}_K = X_1 \cup X_2 \ldots \cup X_K$. An SPN $S(X)$ is recursively defined and constructed by (repeated) application of the following rules:

1. An indicator variable $\mathcal{I}(X_i^j)$ is an SPN $S(\{X_i\})$. The previous definition of indicator variables presented in Eq. 1 applies.
2. A product $\prod_{k=1}^{K} S(X_k)$ is an SPN, with the SPNs $\{S_k(X_k)\}_{k=1}^{K}$ factors.
3. The weighted sum $\sum_{k=1}^{K} w_k S_k(X)$ is an SPN, where $\{w_k\}$ are non-negative weights and sum to 1 for probability distributions and they combine the SPNs $S_k(X)$.

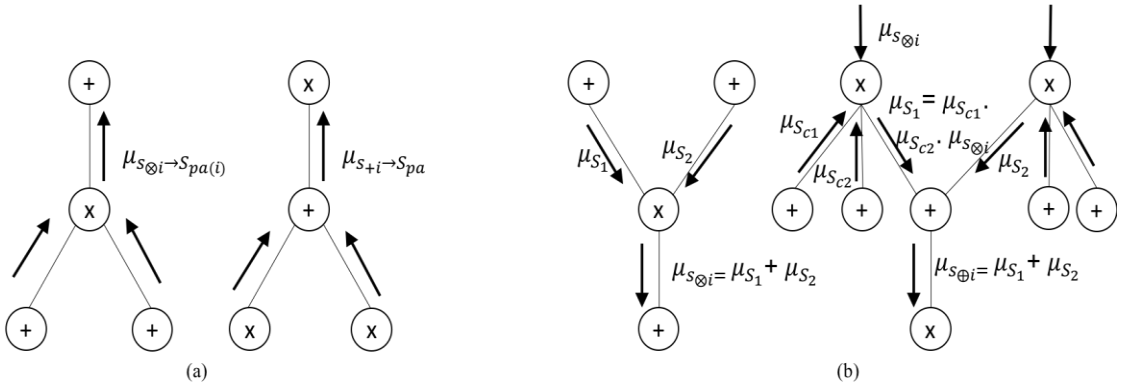


*Figure 4*. Messages computed by SPNs in the bottom-up pass in (a). Messages computed by SPNs in the top-down pass in (b), from equations 6 and 7 respectively.

The above definition also implies a structure of alternating sum and product nodes with the topmost root node being a sum node. SPNs are evaluated in two distinct passes. There is a bottom-up pass for evaluating the probability of particular evidence, as applied via the indicator variables. To obtain the most-probable-explanation assignment of unobserved variables, a top-down pass is also needed, selecting the most likely branch at each sum node. The top-down and bottom-up messages are calculated via the application of differentiation in arithmetic circuits. Interested readers may refer to Darwiche (2009) for a detailed explanation of how these messages are obtained. Here we present just a high-level overview of them in the interest of showing how they can be computed in Sigma.

Figure 4 shows the messages exchanged between child and parent nodes, in both directions. In the bottom-up direction, the message going from a Sum node $S_{i\oplus}$ to its parent node is simply the weighted addition of its child nodes, and similarly the message going from a Product node $S_{i\otimes}$ to its parent node is simply the product of its children. These outgoing messages are the values of the SPN rooted in those nodes. In the top-down direction, the value of a product node is simply the weighted addition of its parents' values ("product" is an accurate label for the node only in the bottom-up direction):

$$S_{i\otimes} = \sum_{k \in pa(i)} w_{ki} \delta S(X) / \delta S_k(X) \tag{6}$$

whereas the value of a sum node is:

$$S_{i\oplus} = \sum_{k \in pa(i)} w_{ki} \delta S(X) / \delta S_k(X) \prod_{l \in Ch(pa(i)) - i} S_l(X) \tag{7}$$

It is important to note that by specifying computations directly, in a computational trellis, SPNs provide a compact representation of all the operations needed to perform exact inference.

### 3.2 Converting SPNs to Sigma Conditionals

As discussed previously, long term knowledge is specified in Sigma using the language of conditionals. In this section we describe an algorithm to translate an SPN into an equivalent Sigma model (Table 1). We focus on valid, decomposable SPNs only because we are interested in

Table 1. Algorithm 1, which generates a Sigma model from an input SPN. The SPN is specified in terms of the sum and product nodes with associated links. The algorithm generates the corresponding Sigma model in time and space linear in the size of the SPN.

Input: An SPN $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with $\mathcal{V} = \{S_{\oplus}, S_{\otimes}, \mathcal{I}(X^i)\}$.
Output: A Sigma model with associated predicates, and conditionals.

1. Declare a set of perception predicates $perceive(X^i)$ for $\mathcal{I}(X^i)$ indicator nodes.
2. Declare two sets of predicates $\{sum\_alpha(S_{\oplus})\}$ and $\{sum\_beta(S_{\oplus})\}$ for bottom-up and top-down values of SPN $S_{\oplus}$nodes.
3. Declare a set of predicates $prod\_alpha\ (S_{\otimes})$, $prod\_beta(S_{\otimes})$ for top-down values of SPN $S_{\otimes}$nodes.
4. Declare a set of perception predicates $perceive(X^i)$ for $\mathcal{I}(X^i)$ indicator nodes.
5. Declare two sets of predicates $\{sum\_alpha(S_{\oplus})\}$ and $\{sum\_beta(S_{\oplus})\}$ for bottom-up and top-down values of SPN $S_{\oplus}$nodes.
6. Declare a set of predicates $prod\_alpha\ (S_{\otimes})$, $prod\_beta(S_{\otimes})$ for top-down values of SPN $S_{\otimes}$nodes.
7. Declare a set of predicates $\{sum\_gamma(S_{\oplus})\}$ for the posteriors.
8. In the bottom-up direction, from indicators to root, for each element of $sum\_alpha(S_{i\oplus})$:
   i. For each product child node $S_{j\otimes}$ of $S_{i\oplus}$, create a conditional such that:
      a. The predicate in the condition is $prod\_alpha(S_{j\otimes})$.
      b. There is an action for the predicate $sum\_alpha(S_{i\oplus})$.
      c. There is a function in the conditional that corresponds to the weight $w_{ij}$.
9. In the bottom-up direction, indicators to root, for each element of $prod\_alpha(S_{i\otimes})$:
   i. Create a conditional such that:
      a. The predicates in the conditions are $sum\_alpha(S_{j\oplus})$, corresponding to the children $S_{j\oplus}$ of $S_{i\otimes}$.
      b. There is an action for the predicate $prod\_alpha(S_{i\otimes})$.
10. In the top-down direction, root to indicators, for each element $sum\_beta(S_{i\oplus})$:
   i. For each parent node $S_{j\otimes}$ of $S_{i\oplus}$ create a conditional such that:
      a. The predicates in the conditions are $sum\_alpha(S_{c\oplus})$, where each $S_{c\oplus}$is a child of $S_{j\otimes}$. Exclude the $sum\_alpha(S_{i\oplus})$ predicates from the conditions.
      b. Add the predicate $prod\_beta(S_{j\otimes})$ to the conditions.
      c. There is an action for the predicate $sum\_beta(S_{i\oplus})$.
11. In the top-down direction, root to indicators, for each element $prod\_beta(S_{i\otimes})$:
   i. For each parent node $S_{j\oplus}$ of $S_{i\otimes}$, create a conditional such that:
      a. The predicate in the condition is $sum\_beta(S_{j\oplus})$ .
      b. There is an action for the predicate $prod\_beta(S_{i\otimes})$.
      c. There is a function in the conditional that corresponds to the weight $w_{ji}$.
12. Initiate the bottom-up pass by providing evidence via Sigma's perception mechanism for the indicator variables in accordance with their definition.
13. Initiate the top-down pass by providing evidence of "1" for the root node $sum\_beta(S_{root\oplus})$.

translating only such SPNs into Sigma. Learning a valid, decomposable SPN from a dense SPN is left for future work.

The number of predicates required in Sigma is at most thrice the number of total nodes in the original SPN. This is because we compute the values of each node in two separate passes, based on separate trees for bottom-up and top-down computations, and then combine them via a third set of predicates. Deconstructing the graph in this fashion breaks the loops that would otherwise occur among the bottom-up and top-down messages along individual links. An example of this shall be seen in the next section. Furthermore, the messages computed by Sigma are a superset of the messages computed by the SPN, but scale by a constant factor as shall be seen in the next section. Finally, selection of the most likely path – as required by some algorithms such as Viterbi (Rabiner, 1989) – can be performed via the selection/decision process provided by Sigma's cognitive cycle. Although the selection process has not yet been implemented here, as the focus has been instead on inferential tractability and exactness, something very similar was previously implemented for continuous speech processing in Joshi, Rosenbloom and Ustun (2016).

**Proposition 1**: Algorithm 1 produces a set of conditionals that computes the same posteriors as the sum and product nodes in the original SPN.

**Proof**: Recall from the earlier discussion of Sigma's graphical architecture that messages are combined via product for predicates that are in conditions and combined via addition when the predicate appears in actions of multiple conditionals. For example, consider the SPNs shown in Figure 4. It can be established due to the conditionals created in steps 5 and 6 that the messages generated towards the $sum\_alpha(S_{i\oplus})$ and $prod\_alpha(S_{i\oplus})$ predicates are the same as those shown in Figure 4a. Similarly, the messages generated towards predicates $sum\_beta(S_{i\oplus})$ and $prod\_beta(S_{i\oplus})$ using conditionals from steps 7 and 8 are those shown in Figure 4b. Since Algorithm 1 can thus generate Sigma conditionals to calculate the messages for basic valid SPNs, then by induction, all decomposable SPN messages – i.e. messages exchanged in decomposable SPNs recursively created using the SPN definition from Section 3.1 – can be generated by conditionals from steps in Algorithm 1. ■

**Proposition 2**: Algorithm 1 induces a Sigma graph that is tractable.

**Proof**: This compilation process yields a set of predicates and conditionals that compile into a graph composed of bottom-up and top-down trees that are isomorphic to the corresponding SPN trees, up to the addition of a constant factor of additional nodes and messages. The sum-product algorithm will respect this tree structure in computing the same results in the same manner as the SPN models. ■

## 4. Probabilistic Grammar Parsing in Sigma

Parsing of probabilistic context free grammars (PCFGs) (Jurafsky & Martin, 2008) defines an important class of problems in computational linguistics. Formally, a context-free grammar (CFG) consists of a set of rules, called productions, expressed over a set of non-terminal symbols, including a special start symbol and terminal symbols. A PCFG extends the notion of a CFG by

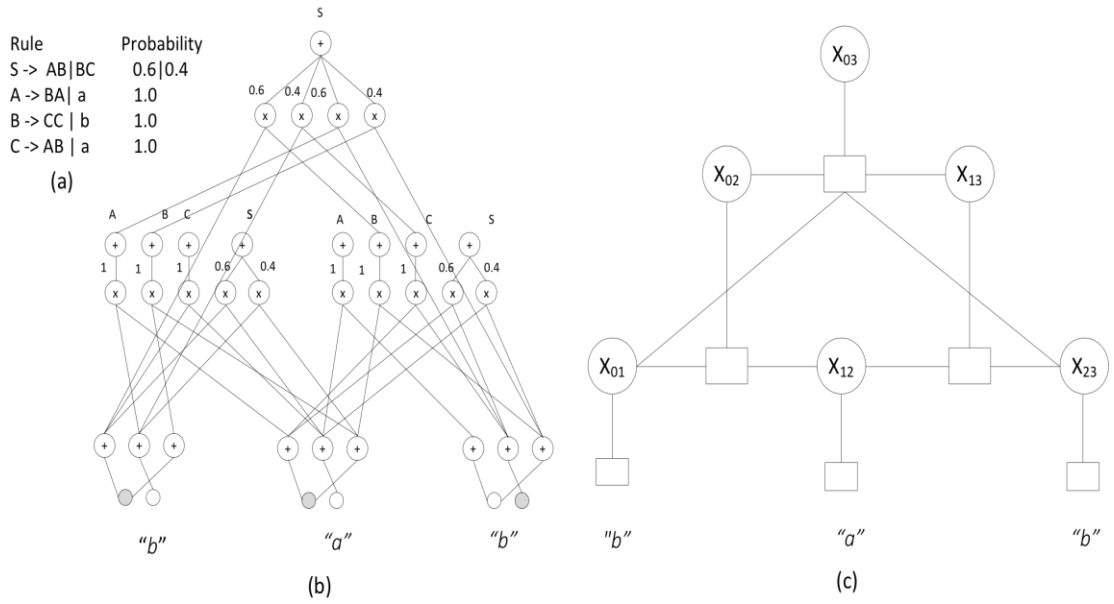


*Figure 5*. (a) A PCFG specified in Chomsky Normal Form. (b) Corresponding SPN, specifying precisely sums and products necessary for inside-outside algorithm, with all invalid trees not represented. (c) Corresponding factor graph version, with each variable node representing a distribution over a set of non-terminals and factor nodes encoding the grammar rules along with their associated probabilities.

assigning probabilities to each rule. A context free language is one that consists of all possible sentences that are derived from a particular context free grammar. Intuitively, the non-terminals model consecutive words as a group – also referred to as a *constituent* – with evidence existing that constituency plays a role in human language processing (Jurafsky & Martin, 2008). Due to its importance in language processing, and its potential architectural importance in this context, we use it here to demonstrate the tractable use of SPNs in Sigma.

Table 2. Example conditionals generated by Algorithm 1. (a) Conditional for grammar rule S−> A B with probability 0.6. Head corresponds to root sum-node in Figure 4b. Conditional corresponds to leftmost link of four in bottom direction toward root sum node. Function represents weight on link between sum and product node. (b) Conditional for downward message from root node to an intermediate sum node, corresponding to grammar rule S−> A B with probability 0.6. Head corresponds to root sum-node in Figure 4b. Downward message to an intermediate node also includes bottom up from sibling, as from Left here. Conditional is sending downward message to right child of rule.

```
CONDITIONAL Bottom-Up-S_AB
   Conditions: Left(non-terminal:A)
               Right(non-terminal:B)
   Actions: Head(non-terminal:S)
   Function: 0.6
```

(a)

```
CONDITIONAL Top-Down-S_AB
   Conditions: Head(non-terminal:S)
               Left(non-terminal:A)
   Actions: Right(non-terminal:B)
   Function: 0.6
```

(b)

The problem of parsing consists of assigning structure to a sentence from the language described by the PCFG. This is typically accomplished by the well-known CKY algorithm from computational linguistics. The CKY algorithm considers an entire sentence as input and generates a set of possible parse trees in a structure called a *parse chart*.[3] The parse chart encodes all valid parse trees that can give rise to the input sentence. Naively representing the parse chart as a graphical model and using the belief propagation algorithm gives rise to exponential inference (exponential in the treewidth). The size of the largest clique provides a measure of how connected the graph is and is used to characterize the efficiency of inference. Additionally, inference is not exact here due to the presence of loops in the graph (Figure 5c). A similar approach was used in Pynadath and Wellman (1998) where PCFGs were mapped onto graphical models by representing the parse chart as a Bayesian network. To avoid the exponential and approximate nature of inference, more recently Naradowsky, Vieira, and Smith (2012) introduced a special factor, *CKYTree*, that encapsulated the dynamic programming necessary to perform cubic time parsing. This special purpose factor requires a modification to the belief propagation message passing schedule to handle the requirements of this factor. Our work here also extends the message passing algorithm but in a way that requires no special purpose factors, other than the nodes which sum across the messages from multiple actions for a single predicate, and which have previously been mentioned to not be logically necessary.

When the probabilities associated with the rules are to be estimated, the inside-outside algorithm (Jurafsky & Martin, 2008) is used. The inside portion of the algorithm is similar to CKY's bottom-up pass and calculates the probability of a substring rooted in a certain non-terminal, whereas the outside portion of the algorithm calculates the probability of a substring rooted in a particular non-terminal in the context of the rest of the sentence. An SPN encoding the inside-outside algorithm for a toy language is shown in Figure 5b. The SPN is cubic in the size of the sentence. Due to the absence of loops, inference is tractable and exact. To understand why inference

[3] Although CKY parsing is not incremental, there are other approaches to PCFG parsing, such as predictive shift reduce (Shieber, Schabes, & Pereira, 1995), that ultimately might better support the needs of architecture-based cognitive systems. In unpublished work, Kenji Sagae explored incremental shift-reduce parsing in Sigma.

is efficient, note that the SPN encodes only the sums and products that are necessary in the context of the grammar.

Table 2a shows a Sigma conditional generated by Step 5 of Algorithm 1, in the bottom-up direction ('inside' in the context of PCFGs). The conditions correspond to the RHS of the grammar rule *S-> A B* of the grammar shown in Figure 5a and are generated according to Step 5.ii. The action represents the LHS of the rule and is generated by Step 5.iii. The function corresponds to the probability of the rule and is generated by Step 5.iv. Table 2b shows the analogous conditional in the top-down direction ('outside' in the context of PCFGs) as generated by Step 7 in the algorithm.[4] It is straightforward to note that these conditionals produce a graph that when operated upon by Sigma's sum-product algorithm yields inference with the desired properties of the underlying SPN. Exactness of inference is established by Proposition 1 and tractability is established by Proposition 2, both from section 3.2.

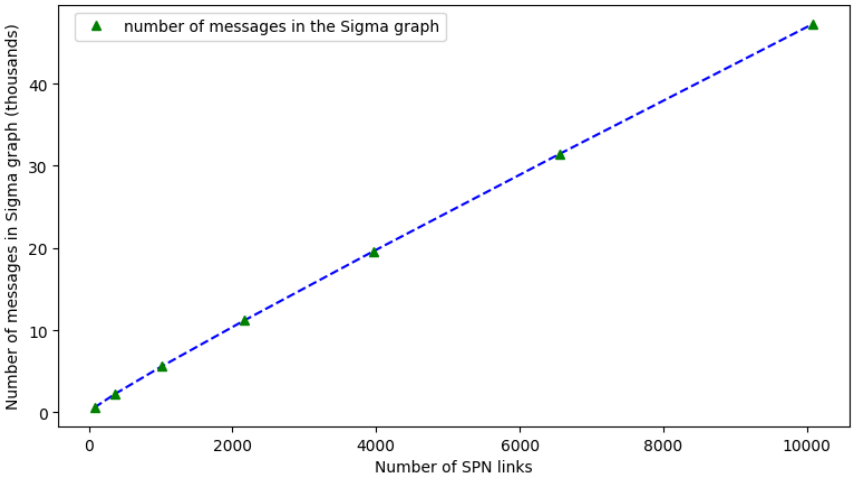


*Figure 6.* The number of messages exchanged in a Sigma graph is linear in the size of the underlying SPN in terms of the number of links in the corresponding SPN.

To explore these claims empirically, a version of Algorithm 1 was implemented for PCFGs and applied to the one in Figure 5a, with the resulting graphs then solved in Sigma. SPNs corresponding to sentence lengths varying from three up to fifteen words were generated. Figure 6 shows the number of messages exchanged in the Sigma SPN model as a function of the number of links in the underlying SPN. It is clear here that the number of messages exchanged in the Sigma SPN graph is linear ($R^2$ of 0.99) in the size of the underlying SPN, as opposed to the exponential growth that would be expected for a pure factor graph (Naradowsky, Vieira, & Smith, 2012).

These results provide a concrete demonstration of what was proven abstractly in the previous section, demonstrating that the inner loop of Sigma's cognitive cycle is able to solve an important

[4] The Sigma-SPN parser code for a set of sample sentences from grammar of Figure 5a is available: `https://bitbucket.org/hima_cogarch/acs_spn/`.

class of problems – parsing for PCFGs – in an exact and tractable fashion. In the process, it raises the possibility of combining this work on PCFGs with other work (Joshi, Rosenbloom & Ustun, 2014; Joshi, Rosenbloom & Ustun, 2016) towards a uniform supraarchitectural integration of speech, language and cognition via conditionals in Sigma, rather than as separate architectural modules, as a major step forward in both grand unification and functional elegance. It also raises the possibility of incorporating other efficient algorithms for important problems, such as optimization and satisfiability, within Sigma.

## 5. Summary and Conclusion

Sum-product networks (SPNs) provide a new graph-based computational model that shows great potential for performing tractable and exact calculations on important problems for which traditional graphical models are exponential and approximate. The core hypothesis examined in this paper has been that Sigma, although grounded in traditional graphical models, already embodies all that is necessary to encode and solve any valid SPN with the tractability and exactness expected of them. This hypothesis was approached via three key steps. First, an algorithm was developed that was proven able to convert any valid SPN into Sigma's cognitive language of conditionals. Second, it was proven that when the resulting cognitive expressions are compiled down to factor graphs that have been extended to allow unidirectional message passing along links – an extension to Sigma that was originally developed to support Rete-like rule match (and actions), and which later proved key in implementing neural networks – then the resulting SPNs retain the exactness and tractability expected of them. Exactness was proven by showing Sigma SPNs calculate the same posteriors as the underlying SPN. Tractability was proven by showing that processing is within a constant factor of the SPN. Third, we demonstrated experimentally that solving SPNs in Sigma for probabilistic context free grammars (PCFGs) retains this expected tractability and exactness. The number of messages required in the Sigma implementation was linear in the number of links in the underlying SPNs.

These results directly bear on Sigma's *sufficient efficiency* desideratum by showing how a major cognitive problem – of parsing PCFGs – can be solved tractably within its graphical processing. They also imply that Sigma may be able to solve a variety of additional major cognitive problems tractably within the cognitive cycle – such as optimization, constraint satisfaction and satisfiability – that would not be possible with pure graphical models. These results also bear on the desideratum of *functional elegance* by reusing for SPNs the core sum-product algorithm that was implemented in Sigma for factor graphs plus the same unidirectional extension that underlies rule match and neural networks. For convenience, this work also reused another extension previously made for rules that sums the messages arriving from multiple actions for the same predicate. These results also provide a path towards bearing on the desideratum of *grand unification* through combining tractable parsing of PCFGs with other work in Sigma on spoken language understanding, including the encoding of speech processing itself via conditionals, to yield a tight coupling of the cognitive and subcognitive aspects of this overall problem.

Moving beyond Sigma to cognitive architectures more broadly, these results suggest a path toward evaluating more generally whether SPNs by themselves, or some suitable generalization of them, might provide the long-sought solution to limiting the cognitive cycle to tractable, and even

bounded, inference while remaining sufficiently expressive to support generic cognition. Such an approach would be akin to, but hopefully more successful due to being more expressive than, earlier efforts in Soar to restrict the expressiveness of rules in order to guarantee tractable match. Beyond cognitive architectures, these results yield a novel approach to combining the efficiency of SPNs with the generality of graphical models. Although other approaches to such a combination have recently been explored, such as Expression Graphs (Demski, 2015) and Sum Product Graphical Models (Desana & Schnorr, 2017), none of these alternatives also extends to include rules and neural networks.

This work showed how SPNs can be leveraged for PCFG processing in a purely reactive fashion. A deliberative sentence processing capability that leverages a dynamic and online form of SPNs for sentence processing is more suitable in a cognitive architecture setting. This provides an intriguing opportunity for future work. Another important opportunity for future work involves exploring the learning of SPNs, both their structure and parameters. Applications of SPNs towards efficient acoustic modeling, language modeling, etc. may also be explored.

## Acknowledgements

This effort has been sponsored by the U.S. Army. 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. We would also like to thank Abram Demski for helpful discussions on SPNs.

## References

Anderson, J. R., Bothell, D. Byrne, M. D., Douglass, S., Lebiere, C., & Qi, Y. (2004). An integrated theory of the mind. *Psychological Review, 111*, 1036−1060.

Darwiche, A. (2009). *Modeling and reasoning with Bayesian networks.* New York: Cambridge University Press.

Demski, A. (2015). Expression graphs. *Proceedings of the Eighth Conference on Artificial General Intelligence* (pp. 241−250). Berlin, Germany: Springer.

Desana, M., & Schnorr, C. (2017). *Sum-product graphical models.* Retrieved from arXiv: `https://arxiv.org/abs/1708.06438`.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence, 19*, 17−37.

Friesen, A., & Domingos, P. (2016). The sum-product theorem: A foundation for learning tractable models. *Proceedings of the Thirty-Third International Conference on Machine Learning* (pp. 1909−1918). New York: PMLR.

Gens, R., & Domingos, P. (2013). Learning the structure of sum-product networks. *Proceedings of the Thirtieth International Conference on Machine Learning* (pp. 873−880). Atlanta, GA: PMLR.

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

Joshi, H., Rosenbloom, P. S., & Ustun, V. (2014). Isolated word recognition in the Sigma cognitive architecture. *Biologically Inspired Cognitive Architectures*, *10*, 1−9.

Joshi, H., Rosenbloom, P. S., & Ustun, V. (2016). Continuous phone recognition in the Sigma cognitive architecture. *Biologically Inspired Cognitive Architectures*, *18*, 23−32.

Jurafsky, D., & Martin, J. H. (2008). *Speech and language processing*. Upper Saddle River, NJ: Prentice Hall.

Koller, D., & Friedman, N. (2009). *Probabilistic graphical models: Principles and techniques*. 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. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Lebiere, C., & Rosenbloom, P. S. (2017). A standard model of the mind: Toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics. *AI Magazine*, *38*, 13−26.

Laird, J. E., & Newell, A. (1983). A universal weak method: Summary of results. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp. 771−773). Karlsruhe, Germany: Morgan Kaufmann.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1−64.

Langley, P., Laird, J. E., & Rogers, S. (2009). Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, *10*, 141−160.

Naradowsky, J., Vieira, T., & Smith, D. (2012). Grammarless parsing for joint inference. *Proceedings of the Twenty-Fourth International Conference on Computational Linguistics* (pp. 1995−2010). Mumbai, India: Association for Computational Linguistics.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Francisco, CA: Morgan Kaufman.

Poon, H., & Domingos, P. (2011). Sum-product networks: A new deep architecture. *Proceedings of the IEEE International Conference on Computer Vision Workshops* (pp. 689−690). Barcelona, Spain: IEEE.

Pynadath, D., & Wellman, M. P. (1998). Generalized queries on probabilistic context-free grammars. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*, 65−77.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*, 257−286.

Rosenbloom, P. S. (2010). Combining procedural and declarative knowledge in a graphical architecture. *Proceedings of the Tenth International Conference on Cognitive Modeling* (pp. 205−210). Philadelphia, PA: Drexel University.

Rosenbloom, P., Demski, A., & Ustun, V. (2016a). The Sigma cognitive architecture and system: Towards functionally elegant grand unification. *Journal of Artificial General Intelligence*, *7*, 1−103.

Rosenbloom, P. S., Demski, A., & Ustun, V. (2016b). Rethinking Sigma's graphical architecture: An extension to neural networks. *Proceedings of the Ninth Conference on Artificial General Intelligence* (pp. 84−94). New York.

Smith, D., & Eisner, J. (2008). Dependency parsing by belief propagation. *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing* (pp. 145−156). Honolulu, HI: Association for Computational Linguistics.

Sun, R. (2016). *Anatomy of the mind: Exploring psychological mechanisms and processes with the Clarion cognitive architecture*. New York: Oxford University Press.

Tambe, M., Newell, A., & Rosenbloom, P. S. (1990). The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning, 5*, 299−348.