Rethinking Sigma's Graphical Architecture: An Extension to Neural Networks

Paul S. Rosenbloom[1,2], Abram Demski[1,2], Volkan Ustun[1]

[1] Institute for Creative Technologies, [2] Department of Computer Science
University of Southern California, Los Angeles, CA USA

Abstract. The status of Sigma's grounding in graphical models is challenged by the ways in which their semantics has been violated while incorporating rule-based reasoning into them. This has led to a rethinking of what goes on in its graphical architecture, with results that include a straightforward extension to feedforward neural networks (although not yet with learning).

Keywords: Cognitive architecture, graphical models, neural network.

1 Introduction

Sigma [1] is a *cognitive architecture* – a computational hypothesis about the fixed structures that together yield a mind – which has manifested a wide range of capabilities implicated in general intelligence, including forms of memory and learning, speech and language, perception and imagery, affect and attention, and reasoning and problem solving. The approach is grounded in the *graphical architecture hypothesis*, that the key at this point is to synthesize across what has been learned from over three decades worth of separate progress on cognitive architectures and *graphical models* [2]. Graphical models provide a general approach to computing efficiently with complex multivariate functions by decomposing them into products of simpler functions and mapping these products onto graphs where they can be solved, typically via message passing or sampling. They are particularly promising as the basis for a cognitive architecture because of how they yield state-of-the-art results across signals, probabilities and symbols from a uniform reasoning algorithm.

The graphical architecture hypothesis is operationalized in Sigma by a two-layer design, with the cognitive architecture implemented on top of a *graphical architecture* that is based on *factor graphs* – a very general form of graphical model – that are solved via the *summary-product* message-passing algorithm [3]. However, it has become increasingly apparent that although Sigma's graphical architecture was inspired by factor graphs, it is not strictly limited to them. What distinguishes factor graphs, and in fact all graphical models, from arbitrary networks of computations is that the former represent a global function, and compute specific properties of this function (most often *marginals* of its variables). This function thus defines a fixed semantics for the graph and the resulting computations over it. The solution algorithm may be exact or approximate, but it should always reflect these semantics.

It has been clear since the beginning that any factor graph in Sigma at best has a form of bottom-up semantics. The overall graph – which both comprises Sigma's memory and structures its reasoning – is built incrementally by compiling fragments of knowledge defined within the cognitive architecture into subgraphs within the graphical architecture. The overall function that defines the semantics is then determined bottom-up from the graph that results. However, beyond this, it turns out that the compiler can create a variety of structures that are not directly interpretable in terms of the semantics of factor graphs, or in fact that of any known graphical model.

How did this come about? To what extent does it occur? And what are its implications? This article provides initial answers to these questions, yielding a broadened perspective on the graphical architecture that focuses on its message-passing algorithm and a fixed set of node and link types rather than on the semantics of the resulting computation. Factor graphs then become one *graphical idiom* – in analogy to a programing idiom – that is defined via a constrained set of node and link types and that provides a particular semantic guarantee; but it is not the only such idiom. Rules, for example, turn out to depend on a related yet distinct idiom.

In new results, one of the key implications is that feedforward neural networks [4] – although not yet with learning – can be supported via the simple addition of a new variant of an existing node type that was originally introduced in support of negated conditions and actions in rules. Some forms of neural networks – such as supervised Boltzmann machines and radial basis functions – are directly compatible with factor graphs [5], but feedforward networks are not. Yet, with this change, they can now coexist in the same overall graph/memory with both rules and factor graphs.

2 How Did This Come About?

Sigma's development is driven by four desiderata: (1) *grand unification*, combining not only the traditional cognitive aspects of intelligence but also the key subcognitive aspects; (2) *generic cognition*, both constructing artificial intelligence and modeling natural intelligence; (3) *functional elegance*, enabling the diversity of intelligent behavior from the interactions among a small general set of mechanisms; and (4) *sufficient efficien*cy, for work at scale. As new capabilities have been added, in service of grand unification and generic cognition, the result may simply be a new factor graph – as happened for isolated word speech recognition [6] and distributed vectors (i.e., word embeddings) [7] – while at other times architectural extensions have been required. Functional elegance biases all such changes to be minimal, preferring small tweaks in existing mechanisms to addition of whole new modules.

Factor graphs are undirected, bipartite graphs composed of variables nodes (VNs) and factor nodes (FNs) (Fig. 1). There is a VN for each variable in the original function and an FN for each subfunction in its decomposition, with each FN connected to all of its variables' VNs. The summary-product algorithm computes messages at these nodes to send to their neighbors. At a VN, an outgoing message along a link is computed via products over the other links' incoming messages. At an

FN, this product also includes the factor function at the node, and then all variables not relevant to the VN on the outgoing link are summarized out, by *sum* (or *integral* for continuous variables) to yield marginals or *max* to yield the mode.

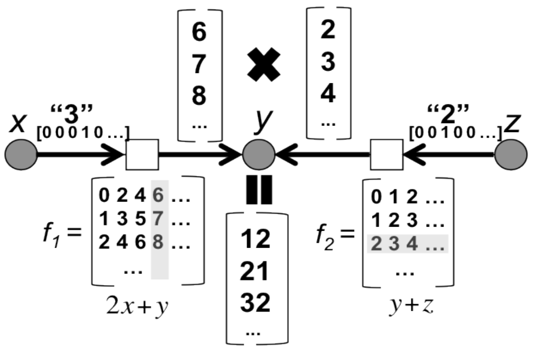


Fig. 1: Summary product computation over the factor graph for $f(x,y,z) = y^2+yz+2yx+2xz = (2x+y)(y+z) = f_1(x,y)f_2(y,z)$ of the marginal on y given evidence concerning x and z.

A number of extensions are possible to this pure model without affecting what the graph computes, and thus without violating factor graph semantics. Two leveraged extensively for sufficient efficiency in Sigma's graphical architecture are: (1) suppression of messages that will not change the ultimate results; and (2) optimization of how specific types of FNs compute outgoing messages. One example of (1) is permanently shutting down one direction of a link if the messages in that direction can never affect the results. A canonical example of (2) is specialized FN implementations for the affine transforms in mental imagery [9]; the same outgoing messages could have been computed by multiplying the incoming messages by an appropriate function, but the resulting *delta* functions would be highly inefficient.

More to the point though are changes that actually alter what is computed. This may involve: (1) eliminating messages on links that can change what is computed; and (2) including FNs that perform computations that aren't reducible, even in principle, to products and summarizations. As an example of (1), links compiled from rule conditions and actions have one direction shut off to enforce the unidirectionality of information flow in rules [10]. As an example of (2), consider negated conditions in rules, which should trigger activity only when the pattern fails to match. This is implemented by specialized FNs that have one input and one output in directed "condition" subgraphs. The input message is subtracted from a constant function of `1`, and then floored at `0` – a value of `1` stands in for *true* with `0` doing the same for *false*, but values in general can also be between these values, or even outside of this range. Negating `[0 .3 1.2 1]`, for example, yields `[1 .7 0 0]`.

Section 3 discusses a set of changes to the graphical architecture that violate factor graph semantics, and thus yield what could be considered a *generalized factor graph*, although it is probably more appropriately considered a generalization of the summary-product algorithm, as the semantics of factor graphs have not been extended in a manner that corresponds to the algorithmic changes that have been made. Such semantic extensions have been explored, but so far without success.

3 To What Extent Does It Occur?

The single biggest driver in extending Sigma's graphical architecture beyond the semantics of factor graphs has been the desire to combine rules with probabilistic graphs. Sigma's primary long-term knowledge structure within the cognitive architecture is a generalized notion of a *conditional*, which combines the forms of conditionality provided by both rules and probabilities. *Conditions* and *actions* in conditionals enable rule-like behavior, with conditions triggering further processing when matched to working memory, and actions proposing changes to working memory. However, conditionals may also contain: *condacts*, which provide a synthesis of conditions and actions to enable the bidirectional flow of information necessary for probabilistic reasoning; and *functions*, where distributions can be stored.

Condacts and functions together yield standard factor graphs. A number of the particular extensions required to support rules and their conditions and actions are what induce semantic divergence, including: (1) directed links, (2) closed-world semantics, (3) universal variables, (4) filter nodes, and (5) transform nodes.

3.1 Directed Links

As mentioned in Section 2, one direction of each link is shut off within conditions and actions to yield one-way rule behavior. This effectively makes these links directed, although this is very different from the directed links in Bayesian networks that still imply bidirectional message passing. The key question though is what has been lost here given that the summary-product algorithm specifies bidirectionality?

In some cases, the omission of back-messages does not affect the end result; it just yields a simple efficiency gain. If excluding back-messages changes the result, however, it's much less clear what is going on. A correlation that has influence in one direction but not the reverse has no place in probability theory; conclusions may be stronger in one direction than the other, but the influence is always there. This isn't necessarily a problem for Sigma, however, because factor graphs don't always carry probabilities. The cases where one-directional propagation changes the end result are often in fact those where the messages are not probabilities.

A simple example is the transitive rule in Sigma. Given a predicate $Above(x, y)$, we can define a conditional with conditions $Above(x, y)$ and $Above(y, z)$, and action $Above(x, z)$. (Henceforth $Above()$ will be abbreviated $A()$.) If the initial contents of working memory are $A(1, 2)$, $A(2, 3)$, $A(3, 4)$, Sigma will generate $A(1, 3)$, $A(2, 4)$, $A(1, 4)$. This is represented by setting A to 1 for the initial knowledge and 0 everywhere else. One-way message passing from conditions to actions does the rest.

Given how this problem is represented, it wouldn't make sense to reason in the opposite direction, as it runs the risk of concluding the premises are false: initially $A(1, 3)$ is 0, so $A(1, 2)$ and $A(2, 3)$ must also be 0. Enforcing the logical constraints in both directions simply doesn't do the right thing. However, the summary-product algorithm provides no justification for restricting message passing to get the desired result. The algorithm computes efficiently over complex global functions by applying the distributive law to push computations to the local-message level. If some messages must be eliminated to get a correct result, the question is: does this somehow improve the approximation or is a different value being computed? The answer appears to be the latter; no specifiable global function is being computed here.

3.2 Closed-World Semantics

Given that probabilities aren't being passed in the transitivity example above, what are the messages? Sigma's employment of *closed-world predicates* is key here [10]. The closed-world assumption is that whatever is not yet known is assumed false. When operating on probabilities, `1` acts as a "total ignorance" number in the summary-product algorithm: multiplying by `1` changes nothing. In closed-world computation, `0` is the total-ignorance number: it represents a lack of information. Therefore, it is natural to combine closed-world information with *or* as opposed to *product*, allowing positive messages to overwrite `0`s, and *probabilistic or* handling intermediate values. Sigma employs special-purpose *action-combination* nodes to enable such a disjunctive combination of messages from multiple actions.

We might try to account for this behavior within the summary-product algorithm by using a different *commutative semiring*, – an algebraic structure like a *ring* but with product commutative and no additive inverse – as factor graphs need not be based on *sum* and *product*, and are in fact well defined for any commutative semiring [3]. Here we would use *or* for combination instead of *product* and *max*, for example, for summarization. However, *or* and *max* fail to form a commutative semiring. `0` is the identity element of both operations, but for a semiring the identity element for the summary operation should be an annihilator for the combination operation. More critically, the distributive law also fails for the two operators – the probabilistic formula for *or* does not distribute over *max*. The distributive law is what justifies shifting from global to local computation in the summary-product algorithm [3].

Really, though, the story in Sigma is more complicated. Open-world or closed-world semantics is associated with predicates, not with variables, implying that as values get passed around the network they will be treated however that local part of the network treats things. The summary operation may be *max* or *sum*. The combination operation will be *product* in most of the network, with *or* and other operations in a few places. In fact, it would not even be possible to make the transitivity example work with closed-world variables if we could not use *product* combination in some places – to combine $A(x, y)$ with $A(y, z)$ – while applying *or* combination elsewhere (to feed the result back into working memory). Is it possible to explain such a mixed approach with the semiring idea?

Multiple semirings can interact nicely in the summary-product algorithm. *Sum-product* and *max-product* can work together to maximize a function over one set of variables while summing over others. In speech recognition, for example, Sigma uses *max-product* for Viterbi processing and *sum-product* everywhere else [6]. A natural approach is to associate summary operators with variables so that each variable is summarized out according to its kind. However, if the order of operations matters, we need to be careful; for instance, $\max_x \sum_y f(x,y)$ differs from $\sum_y \max_x f(x,y)$ in general. Sigma does account for this complication when combining *max* with *sum*.

Closed-world semantics does not seem amenable to a multi-semiring account, however, and no way of specifying a global function has so far emerged.

3.3 Universal Variables

Universal variables in Sigma represent logic variables, indexing many specific cases [10]. This contrasts with *unique/distributional variables* that can represent random variables of the kind used in statistics. If we create an open-world predicate $P(X, Y)$ and declare X to be universal and Y to be unique, this is conceptually like declaring a larger number of predicates $P_1(Y)$, $P_2(Y)$, ... for each possible value of X, allowing generalization over many cases. Unlike lifted reasoning in Alchemy [11], however, Sigma does not compute the same value in these two different cases.

Sigma uses a different summary operator with universal variables, modifying the semiring choice (as discussed in Section 3.2). In particular, *max* is used to yield a form of existential behavior that enables rule-like conditionality: a conditional yields nonzero results if there is any match to its conditions. When Sigma represents probabilistic values, *max* could be seen as a way of computing a probabilistic lower bound; the probability of an existential statement is at least as great as the probability of any instance. It could also be compared to using *max-product* mixed with *sum-product*, which maximizes over some variables while marginalizing over others.

In practice, however, neither of these interpretations adequately captures what's going on because the usefulness of *max* depends on the other modifications that allow Sigma to display rule-like behavior. There is in fact no justification for this formula from a pure factor-graph perspective, and the meaning is unclear. If a universal variable is to indicate multiple instances in a factor graph, then *product* rather than *max* should be used to combine the instances [11, 12]. But *product* would not fit the more common existential use of universal variables in Sigma that *max* supports.

3.4 Filter Nodes

Filter nodes in Sigma implement the constant tests found in rule conditions (plus a bit more [1]). A constant test would be used, for example, in a rule condition like $Above(3, x)$ to yield the values of x that are above 3. To implement this, a filter function in Sigma has the following effect: all values that do not match the desired portion are set to `0`, and the part that's wanted is left unaltered. In the example, the incoming message would be the content of $Above(x, y)$ for all x, y; the y entries for $x = 3$ would remain while those for other values would be set to `0`.

If this were a pure factor graph, such a filter node would be multiplying the global function by a factor that zeros out everything but $x = 3$; that is, this factor would act as a constraint, forcing across the entire network. This is not at all what is happening in Sigma. The undesired entries are only to be removed within the scope of the corresponding conditional. Due to how universal variables are handled, and Sigma's use of closed-world semantics, `0`s are the appropriate way to do this.

Filtering out cases is possible in a pure open-world factor graph, however it would be done with a factor that establishes a uniform distribution, rather than `0`s, over the unwanted parts of the message. This would imply that whatever happened in the sub-network on the other side of the filter node would affect only the selected part of the global function. This illustrates how Sigma's departure from factor-graph operations in some areas forces further departure in other areas. Due to the manner in which universal variables (and some other aspects) are handled, it becomes impossible to view a filter factor node as a standard factor-graph node. The local computations at such a node look like summary-product operations – multiplying the incoming message times a function that is 1 for $x = 3$ and 0 elsewhere – but the global effect on the computation, and thus on the semantics of what is being computed, can be quite different from the effect an identical node would have in a pure factor graph.

3.5 Transform Nodes

A *transform node* applies a one-way transformation over message functions in directed portions of the graph; that is, within conditions and actions. Negated conditions are handled in this manner, as are negated actions. Another example not related to rules is an *exponential* transformation that enables softmax computations, in support of reasoning about other agent's decision processes [13]. Although standard factors can represent arbitrary functions, they are defined only over domain variables in message functions, not over their ranges. These transforms all directly modify the range, with negation subtracting and flooring it, and softmax exponentiating it.

One concern beyond the semantic that is introduced by transform nodes is whether they provide a hook for incorporating arbitrary outside code into Sigma. While such a capability could be appropriate and even useful in a toolkit, it would threaten the graphical architecture's status as an architecture, comprised of fixed structure, and thus also indirectly threaten the status of the cognitive architecture. One way of dealing with this might be to reconceptualize these transformations as learnable knowledge, and then to provide a learning mechanism for them. However, the approach that has been taken is instead to commit to the set of transformations ultimately being bounded, although likely continuing to evolve for some time.

4 What Are Its Implications (Including to Neural Networks)?

At first glance, systematic violations of factor graph semantics might appear to violate the graphical architecture hypothesis. However, the hypothesis does not state that all capabilities must be producible from graphical models, only that understanding the relationship between cognitive architectures and graphical models is crucial. Consistent with this latter notion, even though the new perspective implies that the graphical architecture need not be limited strictly to what falls within the sphere of traditional graphical models, Sigma remains very much based on what has been learned from them. The former notion could be viewed as an alternative *strong graphical architecture hypothesis*, whereas the one actually used here is a weak variant. The strong hypothesis would be a deeper scientific claim, and would likely also yield a simpler and more elegant system, but work to date fails to support it.

On the negative side, Sigma's violations of factor graph semantics do add to the complexity and non-uniformity of the graphical architecture – reducing its simplicity and elegance in essential ways – while also making it more difficult to prove properties about how reasoning and learning work. Cognitive architectures do not in general have well-defined semantics, and even when such attempts are made – e.g., via a formal specification [14] – the result is neither simple nor elegant. So we could just be satisfied with this status. But we can do better, because some of the idioms – such as the one for factor graphs – do still have simple and elegant semantics.

On the positive side, small enhancements to Sigma's graphical architecture can yield major gains in functionality, with much of the core representation and reasoning being reused across idioms, rather than being constructed from scratch in separate modules, to yield a form of algorithmic rather than semantic elegance that is central to how Sigma achieves functional elegance. Semantic elegance is more compelling, but algorithmic elegance still goes far beyond typical cognitive architectures.

The latest example of this is the implementation of feedforward neural networks within Sigma. The one extension required for this is a new one-way transform (Section 3.5) that transforms the incoming message via a non-linear *logistic function*, a variant of a sigmoid. Everything else needed to implement feedforward neural networks, and to incorporate them into the larger architectural context, already exists.

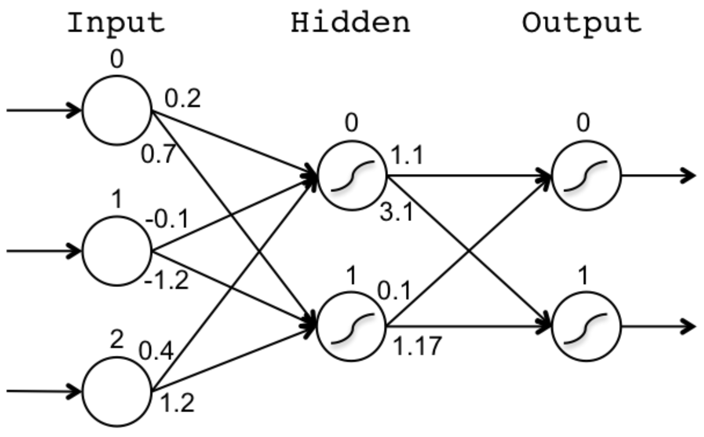


Fig. 2: Example two-layer feedforward neural network (adapted from http://www.doc.ic.ac.uk/~sgc/teaching/pre2012/v231/lecture13.html).

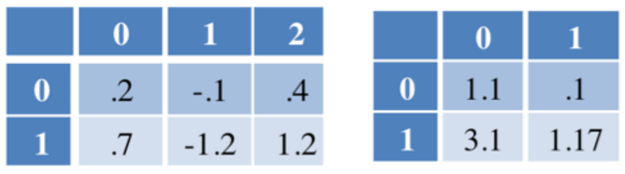
Consider the two-layer feedforward network in Fig. 2, with three inputs, two outputs, and two hidden units. Fig. 3 shows the two conditionals that implement this network in Sigma. Each specifies one layer of the network via two conditions and an action. These are essentially rules, except that the predicates are open world rather than closed world, and the conditions therefore match to either perception (the `Input` predicate) or functions in long-term memory (the `Layer1` and `Layer2` predicates). The weights are stored in the long-term memory functions (Fig. 4). The other key difference from a standard rule is that both actions are marked with **s**, denoting that a sigmoid transform should be applied to their messages. This is just how negated actions are marked, except with an **s** here rather than a **-**.

```
CONDITIONAL C-Layer1
  Conditions: (Input arg:i)
              (Layer1 lower:i upper:h)
  Actions: (Hidden s arg:h)

CONDITIONAL C-Layer2
  Conditions: (Hidden arg:h)
              (Layer2 lower:h upper:o)
  Actions: (Output s arg:o)
```

Fig. 3: Conditionals that define the network in Fig. 2.

Processing is initially driven by conditional `C-Layer1`. A perceived `Input` vector, such as `[10, 30, 20]`, is multiplied times the `Layer1` function with the `Input` variable then summarized out via sum/integral to generate a raw `Hidden` distribution of `[7, -.5]`. This is then logistically transformed to yield `[0.999089, 0.006692851]`. Conditional `C-Layer2` then picks up the processing, multiplying the transformed `Hidden` vector by the `Layer2` function and summarizing out the `Hidden` variable to yield a raw `Output` distribution of `[1.0996672, 3.1050065]`. This is then logistically transformed to yield `[0.75019777, 0.9570988]`. The web source for this network lists the transformed `Output` as `[.750, .957]`, which is the same after round off.



	0	1	2
0	.2	-.1	.4
1	.7	-1.2	1.2

(a) `Layer1` function

	0	1
0	1.1	.1
1	3.1	1.17

(b) `Layer2` function

Fig. 4: Long-term memory functions for layer weights.

Because this form of neural network is just another idiom in the graphical architecture, and thus also in the memory and reasoning of the cognitive architecture, it should be usable just as with all other memory structures; and interfaceable directly, via shared predicates, with all of its other forms of memory, whether rule, semantic, episodic, imaginal, perceptual, etc. It also should be usable directly in reasoning, whether for perception or control of operator selection during problem solving.

One critical piece that remains missing here is learning. Sigma embodies a general form of gradient-descent learning that can acquire many kinds of long-term memory functions from messages arriving at their factor nodes [15]; however, simply enabling bidirectional message passing via condacts and inverting the logistic function for backwards messages through a sigmoid node does not yield appropriate learning. We are currently exploring a generalization of Sigma's learning approach that could extend it appropriately to such neural networks. If this succeeds, it will make sense to consider whether the graphical architecture hypothesis should be extended to include what has been learned from the decades worth of progress on neural networks – a different but related graphical formalism – in addition to graphical models.

5 Conclusion

Sigma's graphical architecture, although inspired by factor graphs, diverges from their semantics in a number of ways. Historically, this has mostly involved how to combine rule-based reasoning with probabilistic reasoning, but here this is extended to feedforward neural networks as well (sans learning for now). This has led to a rethinking of the graphical architecture, to where it is based more explicitly on the summary-product algorithm for solving factor graphs, and its extensions, than on factor graphs themselves. Factor graphs then become one of several graphical idioms that can be supported, although one with a well-defined semantics. Rules and neural networks become two other idioms, each without such semantics; and additional idioms are also conceivable. Whether ultimately a single clean semantics can be developed for Sigma's graphical architecture, or whether an alternative architecture can be found that has a clean semantics, remains an important open question. Either way, the intent is still to yield a single coherent cognitive architecture.

Acknowledgments. 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 Himanshu Joshi for useful discussions on neural networks in Sigma.

References

1. Rosenbloom, P.S.: The Sigma cognitive architecture and system. AISB Quarterly, 136, 4-13 (2013)
2. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press, Cambridge (2009)
3. Kschischang, F.R., Frey, B.J., Loeliger, H.: Factor Graphs and the Sum-Product Algorithm. IEEE Transactions on Information Theory, 47, 498-519 (2001)
4. Rumelhart, D.E., McClelland J.L., the PDP Research Group: Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations. MIT Press, Cambridge (1986)
5. Jordan, M.I., Sejnowski, T. J.: Graphical Models: Foundations of Neural Computation. MIT Press, Cambridge (2001)
6. Joshi, H., Rosenbloom, P.S., Ustun, V.: Isolated word recognition in the Sigma cognitive architecture. Biologically Inspired Cognitive Architectures, *10*, 1-9 (2014)
7. Ustun, V., Rosenbloom, P.S., Sagae, K., Demski, A.: Distributed vector representations of words in the Sigma cognitive architecture. In: Proceedings of the 7th Annual Conference on Artificial General Intelligence (2014)
8. Rosenbloom, P.S.: Towards a 50 msec cognitive cycle in a graphical architecture. In: Proceedings of the 11th International Conference on Cognitive Modeling (2012)
9. Rosenbloom, P.S.: Mental imagery in a graphical cognitive architecture. In: Second International Conference on Biologically Inspired Cognitive Architectures (2011)
10. Rosenbloom, P.S.: Combining Procedural and Declarative Knowledge in a Graphical Architecture. In: 10th International Conference on Cognitive Modeling (2010)
11. Singla, P., Domingos, P.: Lifted First-Order Belief Propagation. In: Proceedings of the 23rd AAAI Conference on Artificial Intelligence (2008)
12. Kersting, K., Ahmadi, B., Natarajan, S.: Counting belief propagation. In: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (2009)
13. Pynadath, D.V., Rosenbloom, P.S., Marsella, S.C., Li, L.: Modeling two-player games in the Sigma graphical cognitive architecture. In: Proceedings of the 6th Conference on Artificial General Intelligence (2013)
14. Milnes, B.G., Pelton, G., Doorenbos, R., Hucka, M. Laird, J., Rosenbloom, P., Newell, A.: A Specification of the Soar Cognitive Architecture in Z. CMU CS Technical Report, Pittsburgh (1992)
15. Rosenbloom, P.S., Demski, A., Han, T., Ustun, V.: Learning via gradient descent in Sigma. In: Proceedings of the 12th International Conference on Cognitive Modeling (2013)