Stochastic And-Or Grammars: A Unified Framework and Logic Perspective (Supplementary Material)

Kewei Tu
School of Information Science and Technology
ShanghaiTech University, Shanghai, China
tukw@shanghaitech.edu.cn

1 Related Models and Special Cases

1.1 Stochastic Context-Free Grammars

Definition 1. A stochastic context-free grammar (SCFG) is a 4-tuple $\langle \Sigma, N, S, R \rangle$:

- Σ is a set of terminal symbols
- N is a set of nonterminal symbols
- S is a special nonterminal called the start symbol
- R is a set of production rules, each of the form $A \rightarrow \alpha$ $[p]$ where $A \in N$, $\alpha \in (\Sigma \bigcup N)^*$, and p is the conditional probability $P(\alpha|A)$.

Any SCFG can be converted into *And-Or normal form* as described in [1]. The conversion results in a linear increase in the grammar size.

Definition 2. An SCFG is in And-Or normal form iff. its nonterminal symbols are divided into two disjoint subsets: And-symbols and Or-symbols, such that:

- each And-symbol appears on the left-hand side of exactly one production rule, and the right-hand side of the rule contains a sequence of two or more terminal or nonterminal symbols;
- each Or-symbol appears on the left-hand side of one or more rules, each of which has a single terminal or nonterminal symbol on the right-hand side.

Proposition 1. *Any SCFG can be converted into And-Or normal form with linear increase in size.*

Proof. We construct a SCFG in And-Or normal form as follows. For each production rule $A \to \alpha$ $[p]$ with two or more symbols in α, create an And-symbol B and replace the production rule with two new rules: $A \to B$ $[p]$ and $B \to \alpha$ [1.0]. Regard all the nonterminals in the original SCFG as Or-symbols. □

Proposition 2. *Any SCFG can be represented by a stochastic context-free AOG with linear increase in size.*

Proof. We first convert the SCFG into And-Or normal form. We then construct an equivalent stochastic context-free AOG $\langle \Sigma, N, S, \theta, R \rangle$:

- Σ is the set of terminal symbols in the SCFG.
- N is the set of nonterminal symbols in the SCFG, with a correspondence from And-symbols to And-nodes and from Or-symbols to Or-nodes.
- S is the start symbol of the SCFG.
- θ maps a substring represented by a terminal or nonterminal symbol to its start/end positions in the complete sentence.
- R is constructed from the set of production rules in the And-Or normal form SCFG; each rule headed by an And-symbol becomes an And-rule, with its parameter relation specifies that the substrings represented by the child nodes must be adjacent (by checking their start/end positions) and its parameter function outputs the start/end positions of the concatenated string represented by the parent And-node (i.e., the start position of the leftmost substring and the end position of the rightmost substring); each rule headed by an Or-symbol becomes an Or-rule with the same conditional probability.

It is easy to verify that the size of the stochastic context-free AOG is linear in the size of the original SCFG. □

1.2 Linear Context-Free Rewriting Systems

Linear context-free rewriting systems (LCFRS) [2] are a class of mildly context-sensitive grammars, which subsume as special cases a few other grammar formalisms [3, 4].

Definition 3. A linear context-free rewriting system is a 4-tuple $\langle \Sigma, N, S, R \rangle$:

- Σ is a set of terminal symbols
- N is a set of nonterminal symbols
- S is a special nonterminal called the start symbol

- R is a set of production rules, each of the form $p : A[g(\beta_1, \ldots, \beta_r)] \rightarrow B_1[\beta_1], \ldots, B_r[\beta_r]$ such that p is the conditional probability of the rule given A, $A, B_1, \ldots, B_r \in N$, $\beta_i \in V^{\phi(B_i)}$ (for $i = 1, \ldots, r$) where $\phi : N \rightarrow \mathbb{N}$ specifies the *fan-out* of a nonterminal symbol and V is a set of variables, and $g : V^{\phi(B_1)} \times \ldots \times V^{\phi(B_r)} \rightarrow ((V \bigcup \Sigma)^+)^{\phi(A)}$ is a *composition function* that is *linear* and *regular*, i.e., in the equation

  $$g(\beta_1, \ldots, \beta_r) = \langle t_1, \ldots, t_{\phi(A)} \rangle$$

  each variable in V appears at most once on each side of the equation and the two sides of the equation contain exactly the same set of variables.

We can define And-Or normal form of LCFRS in a similar way as for SCFG.

Definition 4. An LCFRS is in And-Or normal form iff. its nonterminal symbols are divided into two disjoint subsets: And-symbols and Or-symbols, such that:

- each And-symbol appears on the left-hand side of exactly one production rule, and the number of nonterminal symbols on right-hand side of the rule plus the number of terminals inserted by the composition function is larger than or equal to two;
- each Or-symbol appears on the left-hand side of one or more rules, in each of which the number of nonterminal symbols on right-hand side plus the number of terminals inserted by the composition function is one.

Proposition 3. *Any LCFRS can be converted into And-Or normal form with linear increase in size.*

Proof. The conversion can be done in the same way as for SCFG. □

Proposition 4. *Any LCFRS can be represented by a stochastic context-free AOG with linear increase in size.*

Proof. We first convert the LCFRS into And-Or normal form. We then construct an equivalent stochastic context-free AOG $\langle \Sigma, N, S, \theta, R \rangle$:

- Σ is the set of terminal symbols in the LCFRS.
- N is the set of nonterminal symbols in the LCFRS, with a correspondence from And-symbols to And-nodes and from Or-symbols to Or-nodes.
- S is the start symbol of the LCFRS.
- θ maps a list of substrings represented by a terminal or nonterminal symbol to a list of start/end positions of these substrings in the complete sentence.
- R is constructed from the set of production rules in the And-Or normal form LCFRS:

- Each rule headed by an And-symbol becomes an And-rule, whose right-hand side includes all the right-hand side nonterminal symbols of the original rule as well as all the terminal symbols added by the composition function. Note that each of the substrings represented by the And-symbol is formed by the composition function by concatenating terminals and/or substrings represented by the nonterminal symbols on the right-hand side of the rule. The parameter relation enforces that these component substrings are adjacent (by checking their start/end positions), and the parameter function outputs the start/end positions of the concatenated strings.
- Each rule headed by an Or-symbol becomes an Or-rule with the same conditional probability, whose right-hand side contains the single right-hand side nonterminal symbol of the original rule or the single terminal symbol from the composition function.

It is easy to verify that the size of the stochastic context-free AOG is linear in the size of the original LCFRS. □

1.3 Constraint-based Grammar Formalisms

Constraint-based grammar formalisms [5] associate feature structures to nonterminals and use them to specify constraints in the grammar rules.

Definition 5. A *feature structure* is a set of attribute-value pairs. The value of an attribute is either an atomic symbol or another feature structure. A *feature path* in a feature structure is a list of attributes that leads to a particular value.

Below is an example feature structure, and ⟨Agreement Number⟩ is a feature path leading to the atomic symbol value *singular*.

$$\begin{bmatrix} \text{Category} & \textit{NP} \\ \text{Agreement} & \begin{bmatrix} \text{Number} & \textit{singular} \\ \text{Person} & \textit{third} \end{bmatrix} \end{bmatrix}$$

Definition 6. A constraint-based grammar formalism is a 4-tuple $\langle \Sigma, N, S, R \rangle$:

- Σ is a set of terminal symbols
- N is a set of nonterminal symbols
- S is a special nonterminal called the start symbol
- R is a set of production rules, each of the form $p : A \to \alpha \ \{C\}$ where p is the conditional probability $P(\alpha|A)$, $A \in N$, $\alpha \in (\Sigma \bigcup N)^*$, and C is a set of *feature constraints*; each nonterminal symbol in the rule is associated with a feature structure; each feature constraint takes the form of either "$\langle X$ feature-path$\rangle$ = atomic-value" or "$\langle X$ feature-path$\rangle$ = $\langle Y$ feature-path$\rangle$", where X, Y are nonterminal symbols in the rule.

Proposition 5. *Any constraint-based grammar formalism can be represented with linear increase in size by a generalization of stochastic context-free AOG that allows an And-rule to have only one symbol on the right-hand side.*

Proof. We construct an equivalent stochastic context-free AOG $\langle \Sigma, N, S, \theta, R \rangle$ in which we allow an And-rule to have only one symbol on the right-hand side:

- Σ is the set of terminal symbols in the constraint-based grammar formalism.
- For N, all the nonterminal symbols of the constraint-based grammar formalism become Or-nodes, and for each production rule we create an And-node.
- S is the start symbol of the constraint-based grammar formalism.
- θ maps a word represented by a terminal symbol to the start/end positions of the word in the complete sentence and maps a substring represented by a nonterminal symbol to a feature structure in addition to the start/end positions of the substring.
- R is constructed as follows. For each rule $p : A \to \alpha \; \{C\}$ in the constraint-based grammar formalism, create one Or-rule $p : A \to B$ and one And-rule $B \to \alpha$ where B is a new And-node. Suppose C' is a copy of C with all the appearance of A changed to B. Then the parameter relation of the And-rule is the conjunction of the constraints in C' that does not involve B plus the constraint that the substrings represented by the child nodes must be adjacent (by checking their start/end positions); the parameter function outputs the start/end positions of the concatenated string as well as a new feature structure constructed according to the constraints in C' that involve B.

It is easy to verify that the size of the stochastic context-free AOG is linear in the size of the original constraint-based grammar formalism. □

1.4 Sum-Product Networks

Sum-product networks (SPN) [6] are a new type of deep probabilistic models that can be more compact than traditional graphical models.

Definition 7. A sum-product network over random variables $x_1, x_2, \ldots, x_d$ is a rooted directed acyclic graph. Each leaf node is an indicator x_i or $\bar{x}_i$. Each non-leaf node is either a sum node or a product node. A sum node computes a weighted sum of its child nodes. A product node computes the product of its child nodes. The value of an SPN is the value of its root node. The scope of a node is the set of variables appearing in its descendant leaf nodes. For an SPN to correctly compute the probability of all evidence, the children of any sum node must have identical scopes and the children of any product node cannot contain conflicting descendant leaf nodes (i.e., x_i in one child and $\bar{x}_i$ in another).

Definition 8. A decomposable SPN is an SPN in which the children of any product node have disjoint scopes.

It has been shown that any SPN can be converted into a decomposable SPN with polynomial increase in size [7].

Proposition 6. *Any decomposable SPN can be represented by a stochastic context-free AOG with linear increase in size.*

Proof. We construct an equivalent stochastic context-free AOG $\langle \Sigma, N, S, \theta, R \rangle$:

- Σ is the set of leaf nodes (indicators) in the SPN.
- N is the set of non-leaf nodes in the SPN, with a correspondence from product nodes to And-nodes and from sum nodes to Or-nodes.
- S is the root node of the SPN.
- θ maps any node instance to null (i.e., we set all the instance parameters to null).
- R is constructed as follows: for each product node in the SPN, create an And-rule with the product node as the left-hand side and the set of child nodes as the right-hand side, let the parameter relation be always true, and let the parameter function always return null; for each child node of each sum node in the SPN, create an Or-rule with the sum node as the left-hand side, the child node as the right-hand side, and the normalized weight of the child node as the conditional probability.

As shown in [7], normalization of the child node weights of the sum nodes do not change the distribution modeled by the SPN. Therefore, for any assignment to the random variables, the marginal probability computed by the constructed stochastic context-free AOG and the probability computed by the original SPN are always equivalent. It is easy to verify that the size of the stochastic context-free AOG is linear in the size of the original SPN. □

Note that although SPNs are also general-purpose probabilistic models that can be used in modeling many types of data, stochastic AOGs go beyond SPNs in a few important aspects. Specifically, stochastic AOGs can simultaneously model data samples of different sizes, explicitly model relations, reuse grammar rules over different scopes, and allow recursive rules. These differences make stochastic AOGs better suited for certain domains and applications, e.g., to model recursion in language and translation invariance in computer vision.

2 Computational Complexity of Inference

We prove that the parsing problem of stochastic AOGs (i.e., given a data sample consisting of only terminal nodes, finding its most likely parse) is NP-hard.

Theorem 1. *The parsing problem of stochastic AOGs is NP-hard.*

Proof. Below we reduce 3SAT to the parsing problem.

For a 3SAT CNF formula with n variables and k clauses, we construct a stochastic AOG of polynomial size in n and k. The node parameters in this AOG always take the value of null (i.e., no parameter), and accordingly in any And-rule of the AOG the parameter relation always returns true and the parameter function always returns null. For each variable x_i, create one Or-node A_i, two And-node X_i and $\overline{X_i}$, and two Or-rules $A_i \to X_i | \overline{X_i}$ with equal probabilities. Create an And-rule $S \to \{A_1, A_2, \ldots, A_n\}$ where S is the start symbol. For each clause c_j, create an Or-node B_j, a terminal node C_j and two Or-rules $B_j \to C_j | \epsilon$ with equal probabilities. Here ϵ represents the empty set. For each literal l (which can be either x_i or $\overline{x_i}$ for some i), suppose L is the corresponding And-node (i.e., X_i or $\overline{X_i}$), if l appears in one or more clauses $c_{h_1}, c_{h_2}, \ldots, c_{h_m}$, then create an And-rule $L \to \{B_{h_1}, B_{h_2}, \ldots, B_{h_m}\}$; otherwise create an And-rule $L \to \epsilon$. Note that the constructed AOG does not conform to the standard definition of AOG in that it contains the empty set symbol ϵ and that some And-rules may have only one child node. However, the constructed AOG can be converted to the standard form with at most polynomial increase in grammar size. See [8] for a list of CFG conversion approaches, which can be extended for AOGs. For simplicity in proof, we will still use the non-standard form of the constructed AOG below.

We then construct a data sample which simply contains all the terminal nodes with no duplication: $\{C_1, C_2, \ldots, C_k\}$.

We first prove that if the 3SAT formula is satisfiable, then the most likely parse of the data sample can be found (i.e., there exists at least one valid parse). Given a truth assignment that satisfies the 3SAT formula, we can construct a valid parse tree. First of all, the parse tree shall contain the start symbol and hence the production $S \to \{A_1, A_2, \ldots, A_n\}$. For each variable x_i, if it is true in the assignment, then the parse tree shall contain production $A_i \to X_i$; if it is false, then the parse tree shall contain production $A_i \to \overline{X_i}$. For each clause c_j, select one of its literals that are true and suppose L is the corresponding And-node; then the parse tree shall contain productions $L \to \{\ldots, B_j, \ldots\} \to \{\ldots, C_j, \ldots\}$, where the first production is based on the And-rule headed by L and the second production is based on Or-rule $B_j \to C_j$. In this way, all the terminal nodes in the data sample are covered by the parse tree. Finally, for any B_k node (for some k) in the parse tree that does not produce C_k, add production $B_k \to \epsilon$ to the parse tree. The parse tree construction is now complete.

Next, we prove that if the most likely parse of the data sample can be found, then the 3SAT formula is satisfiable. For each variable x_i, the parse tree must contain either production $A_i \to X_i$ or production $A_i \to \overline{X_i}$ but not both. In the former case, we set x_i to true; in the latter case, we set it to false. We can show that this truth assignment satisfies the 3SAT formula. For each clause c_j in the formula, suppose in the parse tree the corresponding terminal node C_j is a descendant of And-node L (which can be X_i or $\overline{X_i}$ for some i). Let l be the literal corresponding to And-node L. According to the construction of the

AOG, clause c_j must contain l. Based on our truth assignment specified above, l must be true and hence c_j is true. Therefore, the 3SAT formula is satisfied. □

Another inference problem of stochastic AOGs is to compute the marginal probability of a data sample. The proof above can be easily adapted to show that this problem is NP-hard as well (with the same AOG construction, one can show that the 3SAT formula is satisfiable iff. the marginal probability is nonzero).

3 Conversion to Generalized Chomsky Normal Form

In our inference algorithm, we assume the input AOG is in a generalized version of Chomsky normal form, i.e., (1) each And-node has exactly two child nodes which must be Or-nodes, (2) the child nodes of Or-nodes must not be Or-nodes, and (3) the start symbol S is an Or-node.

By extending previous approaches for context-free grammars [8], we can convert any AOG into this generalized Chomsky normal form with the following steps. Both the time complexity of the conversion and the size of the new AOG is polynomial in the size of the original AOG.

1. (**START**) If the start symbol is an And-node, create a new Or-node as the start symbol that produces the original start symbol.
2. (**BIN**) For any And-rule that contains more than two nodes on the right-hand side, replace the And-rule with a set of binary And-rules, i.e., convert $A \to \{x_1, x_2, \ldots, x_n\}$ $(n > 2)$ to $A_1 \to \{x_1, x_2\}, A_2 \to \{A_1, x_3\}, \ldots, A \to \{A_{n-2}, x_n\}$, where A_i are new And-nodes. We will discuss how to convert parameter relation and function later.
3. (**UNIT**) For any Or-rule with an Or-node on the right-hand side, $O_1 \to O_2$, remove the Or-rule and for each Or-rule $O_2 \to x$ create a new Or-rule $O_1 \to x$ (unless it already exists in the grammar).
4. (**ALT**) If an And-rule contains an And-node or terminal node on the right-hand side, replace the node with a new Or-node that produces the node.

In the **BIN** step, we have to binarize the parameter relation t and function f along with the production rule, such that:

$$\begin{aligned} f(\theta_{x_1}, \theta_{x_2}, \ldots, \theta_{x_n}) &= f_A(\theta_{A_{n-2}}, \theta_{x_n}) \\ \theta_{A_{n-2}} &= f_{A_{n-2}}(\theta_{A_{n-3}}, \theta_{x_{n-1}}) \\ &\vdots \\ \theta_{A_2} &= f_{A_2}(\theta_{A_1}, \theta_{x_3}) \\ \theta_{A_1} &= f_{A_1}(\theta_{x_1}, \theta_{x_2}) \end{aligned}$$

and

$$t(\theta_{x_1}, \theta_{x_2}, \ldots, \theta_{x_n}) \Leftrightarrow t_A(\theta_{A_{n-2}}, \theta_{x_n}) \wedge t_{A_{n-2}}(\theta_{A_{n-3}}, \theta_{x_{n-1}}) \\ \cdots \wedge t_{A_2}(\theta_{A_1}, \theta_{x_3}) \wedge t_{A_1}(\theta_{x_1}, \theta_{x_2})$$

In some cases (e.g., the example AOG of line drawings in the main text), the parameter relation and function can be naturally factorized into this form. In general, however, we have to cache multiple parameters of the right-hand side nodes of the And-rule in the intermediate parameters $\theta_{A_1}, \theta_{A_2}, \ldots, \theta_{A_{n-2}}$:

$$\begin{aligned} \theta_{A_1} &= f_{A_1}(\theta_{x_1}, \theta_{x_2}) := \langle \theta_{x_1}, \theta_{x_2} \rangle \\ \theta_{A_2} &= f_{A_2}(\theta_{A_1}, \theta_{x_3}) := \langle \theta_{x_1}, \theta_{x_2}, \theta_{x_3} \rangle \\ &\vdots \\ \theta_{A_{n-2}} &= f_{A_{n-2}}(\theta_{A_{n-3}}, \theta_{x_{n-1}}) := \langle \theta_{x_1}, \theta_{x_2}, \ldots, \theta_{x_{n-1}} \rangle \end{aligned}$$

then we define

$$f_A(\theta_{A_{n-2}}, \theta_{x_n}) := f(\theta_{x_1}, \theta_{x_2}, \ldots, \theta_{x_n})$$

and

$$\begin{aligned} & t_{A_1}(\theta_{x_1}, \theta_{x_2}) = t_{A_2}(\theta_{A_1}, \theta_{x_3}) = \cdots = t_{A_{n-2}}(\theta_{A_{n-3}}, \theta_{x_{n-1}}) := \top \\ & t_A(\theta_{A_{n-2}}, \theta_{x_n}) := t(\theta_{x_1}, \theta_{x_2}, \ldots, \theta_{x_n}) \end{aligned}$$

Note that the sizes of the intermediate parameters can be polynomial in n. This actually violates the requirement that the parameter size shall be upper bounded by a constant. Nevertheless, when running our inference algorithm on the resulting Chomsky normal form AOG, the inference time complexity is only slightly affected, with the last factor $(|X| + |G|)$ changed to a function polynomial in $|X|$ and $|G|$, and hence the condition for tractable inference remains unchanged.

References

[1] Kewei Tu and Vasant Honavar. Unsupervised learning of probabilistic context-free grammar using iterative biclustering. In *ICGI*, 2008.

[2] David Jeremy Weir. Characterizing mildly context-sensitive grammar formalisms. *Ph.D. diss., University of Pennsylvania*, 1988.

[3] Carl Pollard. Generalized context-free grammars, head grammars and natural language. *Ph.D. diss., Stanford University*, 1984.

[4] Mark Johnson. Parsing with discontinuous constituents. In *ACL*, 1985.

[5] Stuart M Shieber. *Constraint-based grammar formalisms: parsing and type inference for natural and computer languages.* MIT Press, 1992.

[6] Hoifung Poon and Pedro Domingos. Sum-product networks : A new deep architecture. In *UAI*, 2011.

[7] Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. On theoretical properties of sum-product networks. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, pages 744–752, 2015.

[8] Martin Lange and Hans Leiß. To CNF or not to CNF? an efficient yet presentable version of the CYK algorithm. *Informatica Didactica*, 8:2008–2010, 2009.