Querying Parse Trees of Stochastic Context-Free Grammars

Sara Cohen
The Selim and Rachel Benin School of Engineering and Computer Science
The Hebrew University
Jerusalem 91094, Israel
sara@cs.huji.ac.il

Benny Kimelfeld
IBM Research – Almaden
San Jose, CA 95120, USA
kimelfeld@us.ibm.com

ABSTRACT

Stochastic context-free grammars (SCFGs) have long been recognized as useful for a large variety of tasks including natural language processing, morphological parsing, speech recognition, information extraction, Web-page wrapping and even analysis of RNA. A string and an SCFG jointly represent a probabilistic interpretation of the meaning of the string, in the form of a (possibly infinite) probability space of parse trees. The problem of evaluating a query over this probability space is considered under the conventional semantics of querying a probabilistic database. For general SCFGs, extremely simple queries may have results that include irrational probabilities. But, for a large subclass of SCFGs (that includes all the standard studied subclasses of SCFGs) and the language of tree-pattern queries with projection (and child/descendant edges), it is shown that query results have rational probabilities with a polynomial-size bit representation and, more importantly, an efficient query-evaluation algorithm is presented.

Categories and Subject Descriptors

H.2 [**Information Systems**]: Database Management; F.4.2 [**Mathematical Logic and Formal Languages**]: Grammars and Other Rewriting Systems—*parsing*; I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*language parsing and understanding, text analysis*; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*linguistic processing*

General Terms

Algorithms, Theory

Keywords

Stochastic Context Free Grammars, Querying, Probabilistic Databases

ICDT 2010, March 22–25, 2010, Lausanne, Switzerland.

1. INTRODUCTION

Uncertainty is often an inherent property of data. To give just a few examples, a medical diagnosis may be based on statistical interpretation of examination results, RFID tags generate signals that noisily represent (possibly ambiguous) events, natural language processing of a string may generate many different possible parses, and information extraction may derive several semantic interpretations of given data. It is interesting to note that all the above-mentioned scenarios involve two components: (1) observed, certain data *and* (2) an uncertain, or probabilistic, process in which the certain data is interpreted. Thus, the result of a medical examination, or of a sequence of sensor transmissions, yields precise observed facts, but their interpretation may be only probabilistic. Similarly, in natural language processing and information extraction, there is an observed string, and a probabilistic process that yields the many possible meanings (i.e., grammatical or semantic parses) of the string.

Recently, there has been significant progress on processing uncertain and probabilistic data. Probabilistic relational databases were studied extensively (e.g., [12, 13, 28, 29]) as were probabilistic XML trees (e.g. [9, 18, 23, 36, 42]). The referenced papers present different approaches to modeling and querying probabilistic relations or XML. In this paper, we adapt the concept of querying a probabilistic database to the task of querying the meaning of a string; this is done by applying the query to the probability space of parse trees (which we view as a probabilistic database).

More specifically, we study the problem of querying the space of parse trees of a given string, as represented by a stochastic context free grammar (SCFG). Basically, our queries are tree patterns with child and descendant edges, Boolean conditions attached to the nodes, and a sequence of projected nodes. An answer is a sequence of strings (for the projected nodes), along with its associated probability value. Thus, we provide a method to answer rich hierarchical queries over the space of parse trees, while deriving the probabilities of the results. This ability is extremely useful in the context of information extraction [10, 16, 17, 45] and natural language processing [30], as well as additional areas in which SCFGs have proven successful (e.g., speech recognition [21, 35], recognizing complex multi-tasked activities [32, 33], and modeling RNA and tRNA [14, 27, 41]).

To demonstrate, consider the scenario in which users enter search queries in an e-commerce site—more specifically, in a site that sells books. The string entered by the user is observable, and hence, is known. The user's intent is, of

course, not provided explicitly. (Note that this is a small-scale example of the problem of information extraction.) Rather, the user intent is modeled as a parse tree that is randomly produced by an SCFG. Figure 1 shows a string virginia woolf orlando biography and an SCFG describing user intent. Together, the string and SCFG imply a space of parse trees appearing at the top of the same figure; for instance, t_1 means that the user is interested in the book "Orlando Biography" by Virginia Woolf. The right side of the figure depicts a query (over the strings logged as search queries), returning authors requested in the search queries, provided that the user asked for a book of the genre biography, or the with keyword biography, or with a title containing the word biography. Each answer (e.g., "virginia woolf" or "orlando") will be associated with its probability.

A string and an SCFG define a *probabilistic parse* which can be viewed as a probabilistic database (namely, a probability space over parse trees). This representation, though, significantly differs from that of previous work on probabilistic databases. For one, the common representation of a probabilistic database (e.g., as in the work referenced earlier) is based on annotation of data items (e.g., tuples or XML elements) with probabilities, along with some additional specifications like mutual exclusion and choice nodes. The representation of a probabilistic parse, on the other hand, follows the two-component view of uncertain data (previously discussed) where the interpretation of the observation is given by a probabilistic branching process (namely, the SCFG). To illustrate this significant difference, a possible world of a standard probabilistic database is a substructure of the annotated database, which is far from being the case here. As another illustration, an SCFG and a string can easily encode an infinite space of parses (with an unbounded size), which cannot be done by annotating an ordinary database. Another pivotal difference is the following. While query evaluation in the above probabilistic-database models is well understood, the process of actually translating real-life uncertainty into those models is yet unclear. In contrast, the task of learning an SCFG has been extensively studied [26, 31, 38, 39].

We note that there has been little work on querying parse trees of SCFGs in the past. Instead, the focus is often on finding the Viterbi (i.e., most likely) parse [25, 44] (or top-k parses [19, 37]), which can then be queried as a deterministic object. A notable exception is [40] which constructs a Bayesian network from an SCFG, and then can query this network. However, the query language of [40] is significantly more restricted than our tree queries, they do not provide complexity guarantees, and they only consider very restricted types of SCFGs; in particular, they do not allow rules with an empty right-hand side, which are shown here to constitute a major source of difficulty in query evaluation.

Devising an efficient algorithm for query evaluation is challenging. First, the space of parse trees can be infinite (and each parse tree in the space can be arbitrarily large) and hence, we must avoid explicit enumeration of the probability space. Second, in the general case, probabilities associated with query answers may require an exponential number of bits to represent, and may even be irrational. If either of these is the case then, obviously, efficient query evaluation is unattainable. To overcome this second hurdle, we (slightly) restrict SCFGs to a special form called *weakly linear SCFGs*. The main contribution of this paper is an efficient algorithm for evaluating a tree query (in the form described above) over a probabilistic parse given by a string and a weakly linear SCFG. The algorithm involves several steps and techniques, including grammar normalization, query decomposition, dynamic programming, computation of a least fixed point, and linear algebra. Efficiency of our algorithm is in terms of data complexity (namely, the query is fixed), and the running time is actually exponential in the size of the query. We show that this is essential, as the problem of query evaluation has an intractable query-and-data complexity, even under strong restrictions.

The paper is organized as follows. In Section 2, we give some preliminary definitions and describe the basic notions of a parse tree and a probabilistic parse. The concept of querying a probabilistic parse is formalized in Section 3. SCFGs, as our specific realization of probabilistic parses, are defined in Section 4. In Section 5, we give results on the complexity of query evaluation (for probabilistic parses given by a string and an SCFG, and for the language of tree queries). Our query-evaluation algorithm (for weakly linear SCFGs) is presented in Section 6. We conclude in Section 7.

2. PRELIMINARIES

In this section, we present some preliminary definitions and notation that we use throughout the paper. Specifically, we define trees and hedges, as well as probability spaces of parse trees.

2.1 Trees and Hedges

Our trees are ordered, unranked, and with labeled nodes. Such trees are often used in the literature to model XML data [23, 34]. Here we will use such trees to represent both parses of text (a concept similar to XML) and queries.

Let Σ be an alphabet. A Σ-*tree* is inductively defined as follows.

- If $\sigma \in \Sigma$ then $\sigma()$ is a Σ-tree.
- If $\sigma \in \Sigma$ and $t_1, \ldots, t_n$ are Σ-trees, then $\sigma(t_1 \cdots t_n)$ is a Σ-tree.

For a one-node tree $\sigma()$, we often omit the parentheses and write just σ.

A Σ-*hedge* is a sequence $h = t_1 \cdots t_n$ of Σ-trees. Note that every Σ-tree is a Σ-hedge. Also, if h is a Σ-hedge and $\sigma \in \Sigma$, then $\sigma(h)$ is a Σ-tree.

Each *occurrence* of a label σ in a Σ-hedge h defines a unique *node* v. We denote by $\mathcal{V}(h)$ the set of nodes of the hedge h. The label associated with the node v is denoted by $\lambda(v)$. A *leaf* of a Σ-hedge is a node without children. The *root* of a Σ-tree t, denoted $\mathrm{root}(t)$, is the node without a parent. For a node v of t, we denote by t^v_Δ the subtree of t that is induced by v and all of its descendants. Note that, as a special case, $t = t^r_\Delta$ where $r = \mathrm{root}(t)$.

When Σ is clear from the context, we may write just *tree* and *hedge* instead of Σ-*tree* and Σ-*hedge*, respectively.

2.2 Parse Trees

We now consider a special type of trees, called *parse trees*. Throughout the paper, we fix countably infinite sets $\mathbf{\Theta}$ of *terminal symbols* (or just *terminals*) and $\mathbf{N}$ of *nonterminal symbols* (or just *nonterminals*), such that $\mathbf{N} \cap \mathbf{\Theta} = \emptyset$. We use S, X, Y and Z (possibly, with a subscript and/or a superscript) to denote nonterminals, a, b and c to denote

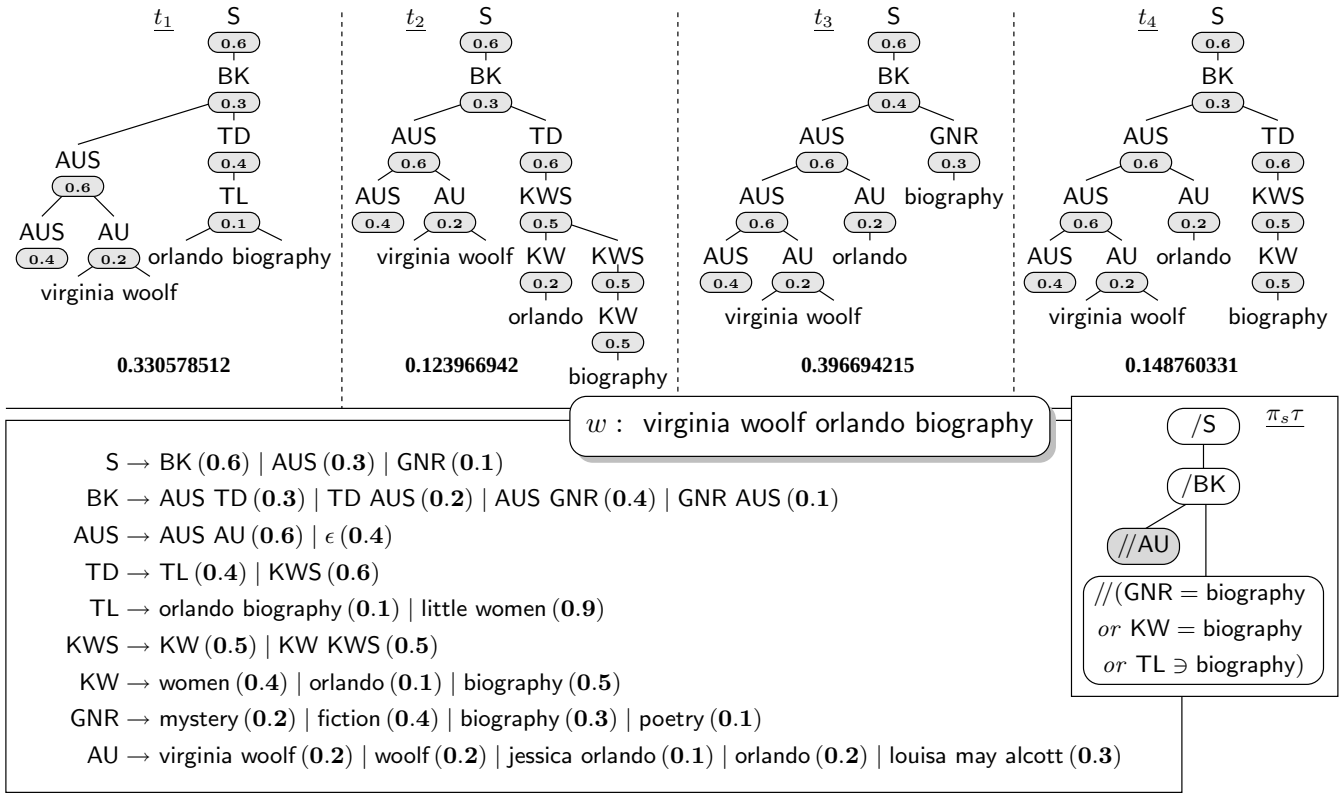

Figure 1: An SCFG Γ^p_{S}, a string $w \in \mathcal{L}(\Gamma_{\mathsf{S}})$, parses of w, and a query $\pi_s\tau$

terminals, and A to denote general (terminal and nonterminal) symbols. A *string* is a member w of $\mathbf{\Theta}^*$, that is, a finite (possibly empty) sequence $a_1 \cdots a_k$ where $a_i \in \mathbf{\Theta}$ for all $i = 1, \ldots, k$. Typically, strings will represent *observable data*, such as a textual document, a sentence uttered, or, as in our running example, a search query that the user has entered.

A *parse tree*, or just *parse* for short, is a $(\mathbf{\Theta} \cup \mathbf{N})$-tree t such that terminals appear only on leaves (that is, every interior node is labeled with a nonterminal, and every leaf is labeled with either a terminal or a nonterminal). Note that a terminal a and a nonterminal X are special cases of parses. Also, if $t = X(t_1 \cdots t_n)$ is a parse, then each t_i (where $1 \leq i \leq n$) is a parse. We denote by $\mathbf{PT}$ the set of all parses. Note that $\mathbf{PT}$ is countable (since our trees are finite).

Let t be a parse. The *yield* of t, denoted by $str(t)$, is the string that is obtained by concatenating all the terminals on the leaves of t, from left to right. Formally, $str(t)$ is recursively defined as follows. If $t = a$ and a is a terminal, then $str(t) = a$; if X is a nonterminal and $t = X(t_1, \ldots, t_n)$, then $str(t) = str(t_1) \cdots str(t_n)$. If $str(t) = w$, then we say that t is a parse *of* w. We denote by $\mathbf{PT}_w$ the (countably infinite) set that comprises all the parses of the string w.

Intuitively, a parse tree t provides a *meaning* or *semantics* to the observable string $w = str(t)$. For example, if w is a textual document, t may provide meta-data describing w (e.g., t can be an XML tree). If w is a sentence uttered, then t can provide the natural-language parsing of w. Finally, in our running example, described below, w is a search query, and t provides the user intent behind the query.

Example 2.1. We demonstrate the key concepts in this paper with a small running example. We consider the scenario in which users enter search queries in an e-commerce site—more specifically, in a site that sells books. The string entered by the user is observable, and hence, is known. However, the user's intent is, of course, not provided. Understanding the user's intent is a key to providing the user with relevant search results, and to identifying user trends. Analyzing user intent for search queries can be thought of as a small-scale example of *information extraction*, where the goal is to obtain contextual information from a string by properly parsing the string, and by identifying the roles of various of the string's components.

Figure 1 shows (among other things that we will discuss later on) a string $w =$ virginia woolf orlando biography (in the center) and four parses $t_1, \ldots, t_4 \in \mathbf{PT}_w$ (in the top). Terminals are represented by lowercase words (e.g., virginia) and nonterminals are represented by uppercase words (e.g., BK). The numbers appearing in these trees should be ignored, for now.

Intuitively, different parses of w represent different meanings. The parse t_1 means that the user is interested in the book "Orlando Biography" by Virginia Woolf[1] and t_3 means that the user is interested in a biography written by both Virginia Woolf and Orlando. □

Example 2.1 demonstrates that a string w can have many plausible parses. In the upcoming section we formalize this idea by considering probability spaces of parses.

[1]For simplifying the example, we slightly changed the real name of the original book which is "Orlando: A Biography."

2.3 Probabilistic Parse Trees

All the probability spaces that are considered in this paper are countable (i.e., either finite or countably infinite). Formally, a *probability space* is a pair (Ω, p), where Ω is a countable set and p is a function from Ω to $[0,1]$ that satisfies $\sum_{o\in\Omega} p(o) = 1$. We say that (Ω, p) is a probability space *over* Ω. Each element of Ω is a *sample* (or a *random instance*) of the probability space and a subset of Ω is an *event* of the probability space. The *support* of a probability space $\tilde{\mathcal{P}} = (\Omega, p)$, denoted $supp(\tilde{\mathcal{P}})$, is the set comprising all the samples of $\tilde{\mathcal{P}}$ with a nonzero probability, that is, $\{o \in \Omega \mid p(o) > 0\}$. We say that the probability space (Ω, p) is *finite* if $supp(\tilde{\mathcal{P}})$ is finite; otherwise, it is *infinite*.

A *probabilistic parse* of a string w is a probability space $\tilde{\mathcal{P}}_w$ over $\mathbf{PT}_w$. We use $\mathcal{P}_w$ (i.e., without the tilde) to denote the random variable that represents a sample chosen from $\tilde{\mathcal{P}}_w$. For example, if $\psi(\cdot)$ is some property of parses (e.g., "contains the nonterminal X"), then $\Pr(\psi(\mathcal{P}_w))$ is the probability that $\psi(t)$ holds for a random instance t of $\tilde{\mathcal{P}}_w$.

Example 2.2. Consider the four parse trees in Figure 1. Below each parse t_i, there is a positive number (in bold font) that we tentatively denote by $p(t_i)$. These numbers sum up to 1. We extend p to all possible parses of w by setting $p(t) = 0$ for all $t \notin \{t_1, t_2, t_3, t_4\}$. Hence, $\tilde{\mathcal{P}}_w = (\mathbf{PT}_w, p)$ is a (finite) probabilistic parse of the string w appearing in the center of Figure 1. □

Recall that different parses of a string w assign different meanings to w. Consequently, a probabilistic parse of w defines a probability space over the meanings of w. As the meaning is not observable or user provided, it is natural to think of a probability space of parses as being derived by a probabilistic process. In this paper, we consider parses described by a stochastic context-free grammar (defined in Section 4), and our results are specific to this model. However, the basic concepts provided in this and the upcoming sections, which formalize the notion of querying a probability space of parses, can be applied to other models of probabilistic parses besides stochastic context-free grammars.

3. QUERYING PROBABILISTIC PARSES

In this section, we present the concept of querying a probabilistic parse of a string. We start by introducing queries over parses, and then provide the semantics for querying probabilistic parses.

3.1 Querying Parses

We now formalize the notion of *querying* a parse. Our queries are basically tree patterns with projection and specialized node conditions. The formal syntax follows.

A *node constraint* $c(L, W)$ is a Boolean predicate over a label (symbol) L and a string W. As we later explain, when such a constraint is applied to a node u of a parse t, the variable L stands for the label of u and W stands for the substring underneath u, namely, $str(t^u_\Delta)$. We fix an infinite class $\mathbf{C}$ of node constraints. We do not pose any restriction on the formalism used for specifying a constraint $c(L, W)$ of $\mathbf{C}$, except that our later complexity analysis will make the assumption that it can be efficiently determined whether $c(A, w)$ holds, for given symbol A and string w. We may omit the parameters L and W when they are not needed, and write just c.

A tree pattern is a tree of node constraints, connected by child and descendant relationships. Formally, let $/\mathbf{C}$ and $//\mathbf{C}$ be the sets of all expressions of the form $/c$ and $//c$, where $c \in \mathbf{C}$. A *tree pattern* τ, or just a *pattern* for short, is a $(/\mathbf{C} \cup //\mathbf{C})$-tree.

Let t be a parse, and let τ be a pattern. A *match* of τ in t is defined in the standard way (e.g., [5, 24]), namely, it is a mapping from the nodes of τ to those of t, such that the node constraints are satisfied and the axis (i.e., / and //) relationships are preserved. Formally, it is recursively defined as follows. A set φ is a match of τ in t, where $t = A(t_1 \cdots t_n)$, if the following hold.

1. If τ is the pattern $/c$, then $c(A, str(t))$ is true and $\varphi = \{\text{root}(\tau) \mapsto \text{root}(t)\}$.
2. If τ is $/c(\tau_1 \cdots \tau_k)$, then φ is the union of matches φ_0, φ_1, ..., φ_k, where φ_0 is a match of $/c$ in t, and for all $1 \le i \le k$, the match φ_i is of τ_i in some t_j (for some $1 \le j \le n$).
3. If $\tau = //c(\tau_1 \cdots \tau_k)$, then φ is a match of $/c(\tau_1 \cdots \tau_k)$ in t^u_Δ for some node u of t.

Item 2 implies that a match of a pattern τ in a parse t is *unordered*; in other words, the order among siblings in a pattern τ is meaningless (in this paper). We use $\mathcal{M}(\tau, t)$ to denote the set of all matches of τ in t, and we denote by $\tau \mapsto t$ the fact that at least one such match exists (i.e., $\mathcal{M}(\tau, t) \neq \emptyset$). Observe that a node v of τ occurs on the left side of exactly one element $v \mapsto u$ of a match. Thus, we naturally view a match $\varphi \in \mathcal{M}(\tau, t)$ as a mapping from the nodes of τ to those of t (i.e., $\varphi(v) = u$ means that $v \mapsto u$ is a member of φ).

A *tree-pattern query* (or just *query* for short) is obtained from a pattern τ by applying *projection*. Formally, a *projection sequence* for a pattern τ is a list $\mathbf{v} = (v_1, \ldots, v_k)$ of nodes of τ. A *query* has the form $\pi_{\mathbf{v}}\tau$, where τ is a pattern and $\mathbf{v}$ is a projection sequence for τ. Given a parse tree t, an *answer* for $\pi_{\mathbf{v}}\tau$, where $\mathbf{v} = (v_1, \ldots, v_k)$, is a sequence $\mathbf{a} = (w_1, \ldots, w_k)$ of substrings of w, such that there exists a match φ of τ in t that satisfies $w_i = str(t^{\varphi(v_i)}_\Delta)$ for all $i = 1, \ldots, k$. We use $\pi_{\mathbf{v}}\tau(t)$ to denote the set of all the answers for $\pi_{\mathbf{v}}\tau$. In other words,

$$\pi_{\mathbf{v}}\tau(t) \stackrel{\text{def}}{=} \{(str(t^{\varphi(v_1)}_\Delta), \ldots, str(t^{\varphi(v_k)}_\Delta)) \mid \varphi \in \mathcal{M}(\tau, t)\}.$$

Remark 3.1. Traditionally, the result of applying a pattern to a tree is a series of nodes, and not of substrings. In our setting, returning internal nodes as answers is rather meaningless, as the internal nodes differ in each parse. Therefore, our choice of returning substrings is natural. We note that our setting can easily accommodate other definitions of query answers, such as indices into the string instead of substrings. □

Example 3.2. The box on the right side of Figure 1 shows a query $\pi_{\mathbf{v}}\tau$. The pattern τ contains three nodes that are depicted by small rounded rectangles, and one node (in the bottom) that forms a large rectangle. Each of the first three nodes has the node constraint "$L = A$," where A is the symbol written inside the rectangle. For example, the constraint of the grey node, which we denote by v_{AU}, is "$L = \mathsf{AU}$," namely, the label should be AU; this node is the output node, that is $\mathbf{v} = (v_{\mathsf{AU}})$. The constraint $c(L, W)$ of the large-rectangle node should be interpreted as follows: $L = \mathsf{GNR}$

and W = biography, or L = KW and W = biography, or L = TL and W contains the terminal biography.

Intuitively, when evaluated over a parse of a user search query, the pattern $\pi_{\mathbf{v}}\tau$ returns the authors requested in the search query, provided that the user asked for a book of the genre biography, or with the keyword biography, or with a title containing the word biography. There are two matches φ_1 and φ_2 of τ in the parse t_3 (also in Figure 1). Both matches map the S and BK to the (single) S and BK nodes of t_3, respectively. Also, the two matches map the right child of BK to the GNR node of t_3. Finally, φ_1 maps the AU node to the AU node above virginia woolf and φ_2 maps it to the AU node above orlando. As a result, we conclude that $\pi_{\mathbf{v}}\tau(t_3) = \{$virginia woolf, orlando$\}$; that is, both virginia woolf and orlando are answers. We can similarly compute $\pi_{\mathbf{v}}\tau(t_i)$ for each of the three other parses t_i in the figure. The reader can verify that the following hold.

- $\pi_{\mathbf{v}}\tau(t_1) = \pi_{\mathbf{v}}\tau(t_2) = \{$virginia woolf$\}$
- $\pi_{\mathbf{v}}\tau(t_3) = \pi_{\mathbf{v}}\tau(t_4) = \{$virginia woolf, orlando$\}$

□

A special case of a query is a *Boolean* query, where $\mathbf{v}$ is the empty sequence. In this case, the answer is the singleton $\{()\}$ if $\tau \mapsto t$, or the empty set otherwise.

3.2 Querying Probabilistic Parses

We define the semantics of querying the probabilistic parse $\tilde{\mathcal{P}}_w$ of a string w according to the conventional notion of querying probabilistic data [9, 12, 13, 23, 24, 36, 42]. Let $\pi_{\mathbf{v}}\tau$ be a query. When evaluating $\pi_{\mathbf{v}}\tau$ over $\tilde{\mathcal{P}}_w$, the set of answers $\pi_{\mathbf{v}}\tau(\tilde{\mathcal{P}}_w)$ comprises all the pairs $(\mathbf{a}, \mathit{conf}_{\mathbf{a}})$, such that $\mathbf{a} \in \pi_{\mathbf{v}}\tau(t)$ for some random parse $t \in \mathit{supp}(\tilde{\mathcal{P}}_w)$, and $\mathit{conf}_{\mathbf{a}}$ is the *confidence* of $\mathbf{a}$, namely, the probability of obtaining $\mathbf{a}$ when querying a random parse. In other words,

$$\pi_{\mathbf{v}}\tau(\tilde{\mathcal{P}}_w) \stackrel{\text{def}}{=} \{(\mathbf{a}, \mathit{conf}_{\mathbf{a}}) \mid \mathit{conf}_{\mathbf{a}} = \Pr(\mathbf{a} \in \pi_{\mathbf{v}}\tau(\mathcal{P}_w)) \wedge \mathit{conf}_{\mathbf{a}} > 0\}\,.$$

Observe that although $\tilde{\mathcal{P}}_w$ is infinite, $\pi_{\mathbf{v}}\tau(\tilde{\mathcal{P}}_w)$ is finite, since w is finite (hence, it has a finite number of substrings). Also observe that if $\pi_{\mathbf{v}}\tau$ is Boolean (i.e., $\mathbf{v}$ is empty), then query evaluation amounts to determining $\Pr(\tau \mapsto \mathcal{P}_w)$.

Example 3.3. Consider again Figure 1. Recall the probabilistic parse $\tilde{\mathcal{P}}_w$ from Example 2.2, containing $t_1, \ldots, t_4$ with the probabilities shown below these parses (and the rest of the parses of w have probability 0). Now consider the query $\pi_{\mathbf{v}}\tau$ (in the same figure). As described in Example 3.2, the possible answers are virginia woolf and orlando. The answer virginia woolf is obtained in all of the four parses; hence, its confidence is 1. The answer orlando is obtained only in t_3 and t_4; hence, the probability of orlando is $0.396694215 + 0.148760331 = 0.545454546$. In conclusion, we have

$$\pi_{\mathbf{v}}\tau(\tilde{\mathcal{P}}_w) = \{(\text{virginia woolf}, 1), (\text{orlando}, 0.545454546)\}\,.$$

□

4. STOCHASTIC CONTEXT-FREE GRAMMARS

In the previous sections, we described the notions of probabilistic parses and querying thereof. In this section, we provide the formalism used to represent a probabilistic parse, namely, a *stochastic context-free grammar*. To make the paper self-contained, we first briefly review the notion of a context-free grammar.

4.1 CFG

A *production rule* (or just *rule*) has the form $X \to \gamma$, where X is a nonterminal and $\gamma \in (\mathbf{\Theta} \cup \mathbf{N})^*$. In particular, γ may be empty and then it is also denoted by ϵ. For a set Γ of production rules, we denote by $\mathbf{\Theta}(\Gamma)$ and $\mathbf{N}(\Gamma)$ the sets of all terminals and nonterminals, respectively, that appear in Γ (in either the left-hand side or the right-hand side of some rule). A *context-free grammar* (CFG) consists of a finite set Γ of rules, such that every nonterminal $X \in \mathbf{N}(\Gamma)$ appears on the left-hand side of some rule in Γ (that is, for all $X \in \mathbf{N}(\Gamma)$ there exists γ such that $X \to \gamma \in \Gamma$). In addition, a CFG has a designated symbol $S \in \mathbf{N}(\Gamma)$ called the *start symbol*. The CFG defined by Γ and S is denoted by Γ_S.

A parse t is *generated by* a CFG Γ_S if the following two conditions hold. First, the root of t is labeled with S, that is, t is of the form $S(t_1 \cdots t_n)$. Second, for all nodes u of t, if $\lambda(u)$ is a nonterminal and $u_1, \ldots, u_k$ is the sequence of children of u (from left to right), then Γ contains the rule $\lambda(u) \to \lambda(u_1) \cdots \lambda(u_k)$. Note that the second condition implies that a leaf u of t is either labeled with a terminal or is such that Γ contains the rule $\lambda(u) \to \epsilon$. The set of all the parses that are generated by Γ_S is denoted by $\mathbf{PT}(\Gamma_S)$. Similarly, for a string w, the subset of $\mathbf{PT}(\Gamma_S)$ that comprises the parses of w (i.e., the set $\mathbf{PT}(\Gamma_S) \cap \mathbf{PT}_w$) is denoted by $\mathbf{PT}_w(\Gamma_S)$. The *language* of Γ_S, denoted $\mathcal{L}(\Gamma_S)$, is the set of all the strings w that have a parse generated by Γ_S; that is,

$$\mathcal{L}(\Gamma_S) \stackrel{\text{def}}{=} \{w \in \mathbf{\Theta}^* \mid \mathbf{PT}_w(\Gamma_S) \neq \emptyset\}\,.$$

Example 4.1. The bottom part of Figure 1 shows a CFG Γ_{S} (that is, Γ comprises all the specified rules, and the start symbol is S), when ignoring the numbers appearing in parentheses. We use the convention that $X \to \gamma_1 \mid \cdots \mid \gamma_m$ is a shorthand notation for the m rules $X \to \gamma_i$ for $i = 1, \ldots, m$. Each of the parses $t_1, \ldots, t_4$ (in the top of the figure) is generated by Γ_{S}. The reader can verify that for the string w in the figure, the set $\mathbf{PT}_w(\Gamma_{\mathsf{S}})$ is exactly $\{t_1, t_2, t_3, t_4\}$. □

4.2 SCFG

A CFG describes a process of producing parse trees and strings thereof. This process is nondeterministic since a specific nonterminal can appear on the left side of several rules. A *stochastic* (also called *probabilistic*) *context-free grammar* (SCFG) is similar to a CFG, except that the rules are augmented with probabilities; that is, the nondeterministic process is replaced with a probabilistic one. Formally, an SCFG comprises a CFG Γ_S and a function $p : \Gamma \to (0, 1]$ that assigns probabilities to rules, such that for all nonterminals $X \in \mathbf{N}(\Gamma)$,

$$\sum_{(X \to \gamma) \in \Gamma} p(X \to \gamma) = 1\,.$$

That is, for all nonterminals X of Γ, the probabilities of the rules with X on the left-hand side sum up to 1. Note that we do not allow zero probabilities. The SCFG that is given by Γ_S and p is denoted by Γ_S^p.

Example 4.2. Figure 1 shows an SCFG Γ_{S}^p. The CFG Γ_{S} is described in Example 4.1. The probability $p(X \to \gamma)$

of each rule $X \to \gamma$ is written in parentheses to the right of γ. As an example, $p(\mathsf{AUS} \to \mathsf{AUS\ AU}) = 0.6$ and $p(\mathsf{AUS} \to \epsilon) = 0.4$. Note that the sum of these two numbers is 1, which is as required since these two are the probabilities of all the rules having AUS on the left-hand side. □

Consider an SCFG Γ_S^p. Recall that p is a function over Γ. We extend p to $\mathbf{PT}(\Gamma_S)$ as follows. The number $p(t) \in (0, 1]$, for a parse t generated by Γ_S, is the probability that all the applications of production rules that generated t indeed occur, when viewing different applications as probabilistically independent. Formally, $p(t)$ is recursively defined as follows. If $t = a$ where a is a terminal, then $p(t) = 1$; otherwise, if $t = X(t_1 \cdots t_n)$ where X is a nonterminal, then $p(t) = p(X \to \gamma) \times p(t_1) \times \cdots \times p(t_n)$ where γ is the concatenation of the root labels of $t_1, \ldots, t_n$ (i.e., $\gamma = \lambda(\mathrm{root}(t_1)) \cdots \lambda(\mathrm{root}(t_n))$); as a special case, if $n = 0$ (i.e., $t = X$), then $p(t) = p(X \to \epsilon)$.

Example 4.3. We continue with the running example. Recall that $\mathbf{PT}_w(\Gamma_\mathsf{S}) = \{t_1, t_2, t_3, t_4\}$. Below each node u of a parse t_i, the figure contains a rounded rectangle with the probability $p(X \to \gamma)$, where X and γ correspond to the labels of u and its children, respectively. As an example, consider the BK node u of the parse t_1. Then u is the only child of the root (labeled with S), and the probability 0.6 below the root is $p(\mathsf{S} \to \mathsf{BK})$. The probability below u is 0.4, which is $p(\mathsf{BK} \to \mathsf{AUS\ TD})$. For $i = 1, \ldots, 4$, the number $p(t_i)$ (which is not shown in the figure) is obtained by simply multiplying the probabilities in the rounded rectangles of t_i. For example, $p(t_1) = 0.6^2 \times 0.4^2 \times 0.3 \times 0.2 \times 0.1 = 0.0003456$. □

One may think that (the extended) p implies a probability function over $\mathbf{PT}(\Gamma_S)$ (that is, the pair $(\mathbf{PT}(\Gamma_S), p)$ is a probability space). However, this is not the case in general, since the sum $\sum_{t \in \mathbf{PT}(\Gamma_S)} p(t)$ can be smaller than one. (It is easy to show that this sum is never larger than one.) For instance, this sum is smaller than one if $\mathbf{PT}(\Gamma_S)$ (and hence $\mathcal{L}(\Gamma_S)$) is empty. A more interesting example is shown next.

Example 4.4. Consider the SCFG Γ_S^p that is given by the rules

$$S \to SS\ (q) \mid a\ (1-q)\,,$$

where q is a number satisfying $1 > q > 1/2$. It is shown in [6] that $\sum_{t \in \mathbf{PT}(\Gamma_S)} p(t) = \frac{1}{q} - 1$, and hence is smaller than 1. The "lost" probability is taken by infinite parses, which by definition are excluded from $\mathbf{PT}(\Gamma_S)$. □

Conditions guaranteeing that $(\mathbf{PT}(\Gamma_S), p)$ is a probability space have been studied in [4, 48], and in [15] an efficient algorithm is given for deciding whether this is the case for a given SCFG. However, this property is *not* required for the tasks we consider here, since we are only interested in the subspace that comprises the parses of a given string w. The formal definition of this subspace follows.

Let Γ_S^p be an SCFG and let $w \in \mathcal{L}(\Gamma_S)$ be a string. We again abuse the notation and denote by $p(w)$ the probability that the random process defined by Γ_S^p terminates and produces w; that is, $p(w) = \sum_{t \in \mathbf{PT}_w(\Gamma_S)} p(t)$. Note that $p(w) > 0$ since $w \in \mathcal{L}(\Gamma_S)$. The probability space $\tilde{\mathcal{P}}_w^S$ is the pair $(\mathbf{PT}_w, q)$ where for all parses t of w,

$$q(t) = \begin{cases} \frac{p(t)}{p(w)} & \text{if } t \in \mathbf{PT}_w(\Gamma_S); \\ 0 & \text{otherwise.} \end{cases}$$

Thus, $\tilde{\mathcal{P}}_w^S$ can be viewed as the conditional probability subspace of the probabilistic branching process Γ_S^p, where the condition is (termination and) yield of w. Clearly, $\tilde{\mathcal{P}}_w^S$ is a probabilistic parse. Observe that $\tilde{\mathcal{P}}_w^S$ may be infinite. For instance, this is the case for the SCFG Γ_S^p that is given by $S \to S\ (0.5) \mid a\ (0.5)$ and the string $w = a$.

Example 4.5. Consider again the parses t_1, t_2, t_3 and t_4 for the string w in our running example. Recall from Example 4.1 that $\mathbf{PT}_w(\Gamma_\mathsf{S}) = \{t_1, t_2, t_3, t_4\}$, and recall from Example 4.3 that each $p(t_i)$ is obtained by multiplying the probabilities in the rounded rectangles of t_i. The reader can verify that $p(w)$, which is the sum of the $p(t_i)$ for $i \in \{1, 2, 3, 4\}$, is 0.00104544. Thus, each $q(t_i)$, or equivalently $\Pr(\mathcal{P}_w^\mathsf{S} = t_i)$, is equal to $p(t_i)/0.00104544$. For example, $q(t_1) = 0.0003456/0.00104544 = 0.330578512$. For each i, the probability $q(t_i)$ is shown in Figure 1 below the parse t_i. □

In the remainder of this paper, we focus on the problem of querying a probabilistic parse that is represented by means of an SCFG. More formally, given a string w, an SCFG Γ_S^p, and a query $\pi_\mathbf{v}\tau$, the task is to evaluate $\pi_\mathbf{v}\tau$ over the underlying probabilistic parse $\tilde{\mathcal{P}}_w^S$, that is, to obtain the result $\pi_\mathbf{v}\tau(\tilde{\mathcal{P}}_w^S)$. However, this task is not a well defined computational problem (e.g., in the Turing-machine model) since the probabilities specified in $\pi_\mathbf{v}\tau(\tilde{\mathcal{P}}_w^S)$ do not necessarily admit a finite representation (even if each $p(X \to \gamma)$ is given as a rational number). Therefore, we impose some restrictions on Γ_S^p. This and other computational aspects of this problem are discussed in the next section.

5. COMPLEXITY RESULTS

In the remainder of this paper, we study the computational task of evaluating a query over a probabilistic parse of a string, given the string and an SCFG. In this section, we first discuss how probabilities are represented, as well as cases in which the probabilities in the results cannot be efficiently represented. We then present our main complexity results.

5.1 Representing Probabilities

We first need to specify how the numbers (probabilities) are represented in both the input and the output. We use the convention (e.g., [9, 15]) that numbers are rational and each is represented by a pair of integers: the numerator and the denominator. Moreover, each of the two integers is represented in the standard binary encoding (thus, the number of bits required to represent an integer d is logarithmic in the actual value of d).

Recall that our goal is to compute $\pi_\mathbf{v}\tau(\tilde{\mathcal{P}}_w^S)$, that is, the pairs $(\mathbf{a}, \mathit{conf}_\mathbf{a})$, such that $\mathbf{a} \in \pi_\mathbf{v}\tau(t)$ for some parse $t \in \mathbf{PT}_w(\Gamma_S)$, and $\mathit{conf}_\mathbf{a}$ is the probability of $\mathbf{a}$. We desire the probabilities in the output to have the same representation as those in the input. Unfortunately, the following example shows that in general, the sought probabilities in the output may be irrational. Thus, the problem of query evaluation is not well defined since the output does not necessarily admit a finite representation (in, e.g., the Turing-machine model).

EXAMPLE 5.1. Let Γ_S^p be defined as follows:

$$S \to X\ (1/2) \mid Y\ (1/2)$$
$$X \to a\ (1)$$
$$Y \to aZ\ (1)$$
$$Z \to \epsilon\ (1/2) \mid b\ (2/6) \mid ZZZZZ\ (1/6)$$

Let w be the (one-symbol) string a, and let τ be the pattern $/X$ (in XPath notation, i.e., $/l$ is a shorthand notation for $/c$ where $c(L, V)$ is "$L = l$"). We view τ as a Boolean query, where the result of query evaluation is essentially the probability $\Pr(\tau \mapsto \mathcal{P}_w^S)$. There is exactly one parse $t \in \mathbf{PT}_w(\Gamma_S)$, namely $S(X(a))$, that satisfies $\tau \mapsto t$; let t be that parse. Then, $\Pr(\tau \mapsto \mathcal{P}_w^S)$ is equal to $p(t)/p(w)$. Now, $p(w)$ is equal to $p(t) + 0.5q$, where q is the probability that Z produces the empty string. Since $p(t) = 0.5$, we get that $\Pr(\tau \mapsto \mathcal{P}_w^S) = 1/(1+q)$. So, if $\Pr(\tau \mapsto \mathcal{P}_w^S)$ is rational, then q is rational as well. But q satisfies the equality $q = \frac{1}{2} + \frac{1}{6}q^5$, which means that q is a root of the polynomial $x^5 - 6x + 3$, and it is known (e.g., [15, 43]) that this polynomial has no rational roots (or even ones that are radical over $\mathbb{Q}$). □

The above example is essentially an adaptation of the example given in [15] for the irrationality of the termination probability of a *recursive Markov chain*.

REMARK 5.2. Due to Example 5.1, the query evaluation problem does not admit a computational solution. One way to circumvent this issue is to allow the probabilities in the output to be approximate (e.g., *k-bit precise*). Whether this enables an *efficient* evaluation remains an open problem. However, it can easily be shown that such an approximation would yield an approximation for the probability of termination of an SCFG (i.e., the value $\sum_{t \in \mathbf{PT}(\Gamma_S)} p(t)$), and whether the latter approximation can be efficiently obtained is a long-standing open problem (see, e.g., [15]). In this paper, we take a different approach to the problem illustrated by Example 5.1, as discussed below. □

We restrict the language of SCFGs considered, so as to avoid irrational probabilities in the output. Care must be taken when devising such a restriction, to ensure that the probabilities in the output can be represented efficiently (i.e., the numerators and the denominators require a polynomial number of bits). A simple, (but rather drastic) restriction is to consider only *non-recursive* SCFGs. Formally, Γ_S^p is *non-recursive* if there are no cycles in the directed graph that has the node set $\mathbf{N}(\Gamma)$ and an edge (X, Y) whenever a rule of the form $X \to \gamma_1 Y \gamma_2$ is in Γ.

If the SCFG Γ_S^p is non-recursive, then $\mathbf{PT}_w(\Gamma_S)$ is finite. This implies that for a query $\pi_{\mathbf{v}}\tau$, the result $\pi_{\mathbf{v}}\tau(\tilde{\mathcal{P}}_w^S)$ contains only rational probabilities. Nevertheless, the following example shows that even if non-recursiveness is assumed, the desired probabilities may require exponentially many bits to represent (moreover, that holds even if the query is Boolean).

EXAMPLE 5.3. Let w be the string a, and let τ be the pattern $/Y$. For $n \geq 1$, consider the following SCFG Γ_S^p.

$$S \to X\ (1/2) \mid Y\ (1/2)$$
$$X \to a\ (1)$$
$$Y \to aZ_n\ (1)$$
$$Z_i \to Z_{i-1}Z_{i-1}\ (1) \quad i = n, n-1, \ldots, 1$$
$$Z_0 \to \epsilon\ (1/2) \mid b\ (1/2)$$

Let p_X and p_Y be the probabilities that X and Y produce w, respectively. Then $p(w)$ is equal to $0.5p_X + 0.5p_Y$, and $\Pr(\tau \mapsto \mathcal{P}_w^S)$ is equal to $0.5p_Y/(0.5p_X + 0.5p_Y) = p_Y/(p_X + p_Y)$. Now, $p_X = 1$, and p_Y is the probability that Z_n produces the empty string. The probability that Z_0 produces the empty string is 0.5. For Z_1, it is 0.5^2, for Z_2 it is 0.5^4, and in general, the probability Z_i produces the empty string is 2^{-2^i}. Hence, $p_Y = 2^{-2^n}$. We conclude that $\Pr(\tau \mapsto \mathcal{P}_w^S) = 1/(1 + 2^{2^n})$ and, therefore, the number of bits required for representing the probability $\Pr(\tau \mapsto \mathcal{P}_w^S)$ (in particular, its denominator) is $\Omega(2^n)$. □

Thus, even though the language of SCFGs has been greatly restricted, efficient representation of the output is still not possible. In the next section, we briefly review other restrictions of SCFGs (equivalently, of CFGs) that are common in the literature, and then describe the restriction assumed in this paper, called *weak linearity*. Importantly, weak linearity requires weaker assumptions than other common restrictions, and allows recursion.

5.2 Weakly Linear SCFGs

Consider an SCFG Γ_S^p. We say that Γ_S^p is *(1) null free* if Γ contains no rule of the form $X \to \epsilon$, *(2)* in *Chomsky normal form* if Γ contains only rules of the form $X \to YZ$, $X \to a$ or $S \to \epsilon$ where X, Y and Z are nonterminals, a is a terminal, and neither Y nor Z is S and *(3) linear* if the right-hand side γ of every rule $X \to \gamma$ of Γ has at most one nonterminal. These classes of SCFGs have been considered extensively in the past.

Let Γ_S^p be an SCFG. A symbol A of Γ_S^p is said to be *nullable* if A is a nonterminal and the empty string ϵ can be produced from A, that is, $\epsilon \in \mathcal{L}(\Gamma_A)$. *Weak linearity* of an SCFG is defined as follows.

DEFINITION 5.4 (WEAK LINEARITY). *An SCFG Γ_S^p is* weakly linear *if for all rules $X \to \gamma$ of Γ, if γ contains two or more symbols, then at least one of them is not nullable.*

It is easy to see that null-free SCFGs, linear SCGFs and SCFGs in Chomsky normal form are all special cases of weakly linear SCFGs (this is also the case for other popular normal forms, such as *Greibach normal form*). The following example shows that weakly linear SCFGs strictly generalize these three classes. Note that weakly linear SCFGs can be recursive, and the classes of weakly linear SCFGs and non-recursive SCFGs are incomparable.

EXAMPLE 5.5. Recall the SCFG Γ_S^p of Figure 1. The reader can verify that Γ_S is weakly linear. This follows immediately from the fact that the only nullable symbol is AUS, and this symbol appears at most once on the right-hand side of each rule. Note that Γ_S is not null free, not in Chomsky normal form, and not linear (since, e.g., it contains the rule BK → AUS TD). Also, observe that Γ_S is recursive (since it contains the rules AUS → AUS AU and KW → KW KWS). □

Observe that for a weakly linear SCFG Γ_S^p and a string w, the set $\mathbf{PT}_w(\Gamma_S)$ (and the probability space $\tilde{\mathcal{P}}_w^S$) can still be infinite. Intuitively, weakly linear SCFGs avoid arbitrarily wide parses, but allow arbitrarily deep parses. As an example, let Γ_S^p be the weakly linear SCFG defined by

$$S \rightarrow S\ X\ (0.5) \mid a\ X\ (0.5)$$
$$X \rightarrow \epsilon\ (0.5) \mid b\ (0.5)$$

and let w be the string a. Then $\mathbf{PT}_w(\Gamma_S)$ contains infinitely many parses, such as $S(aX)$, $S(S(aX)X)$, $S(S(S(aX)X)X)$, and so on.

REMARK 5.6. In principle, an SCFG can be transformed into one in Chomsky normal form in a manner that preserves the probability of every string [1]. However, in this work we *cannot* assume that the given SCFG is normalized, for several reasons. First, such a translation is intractable as it may require specification of probabilities that cannot be represented efficiently (or even finitely). Even more fundamentally, such a translation changes the parses themselves and, consequently, it is likely to change the result (namely, the answers and their probabilities) for a query. □

5.3 Complexity

Our main focus is on query evaluation under the standard assumption of *data complexity* [47]. Formally, under data complexity, the query $\pi_{\mathbf{v}}\tau$ is held fixed and the input consists of an SCFG Γ_S^p and a string w. The goal is to compute $\pi_{\mathbf{v}}\tau(\tilde{\mathcal{P}}_w^S)$.

The central result of the paper is that for weakly linear SCFGs, query evaluation is tractable. Formally, we prove the following.

THEOREM 5.7. *Let $\pi_{\mathbf{v}}\tau$ be a fixed query. The evaluation of $\pi_{\mathbf{v}}\tau(\tilde{\mathcal{P}}_w^S)$, given a weakly linear SCFG Γ_S^p and a string $w \in \mathcal{L}(\Gamma_S)$, is in polynomial time; in particular, $\pi_{\mathbf{v}}\tau(\tilde{\mathcal{P}}_w^S)$ can be represented using a polynomial number of bits.*

Theorem 5.7 is comprised of two separate results. First, weak linearity is sufficient to overcome the problems shown in Examples 5.1 and 5.3, thereby ensuring that all probabilities can be represented using a polynomial number of bits. Second, there is an efficient algorithm that computes these probabilities.

Theorem 5.7 gives an upper bound on the data complexity of the evaluation problem, under the assumption of weak linearity. We now show that this result cannot be improved to *combined* (query-and-data) complexity, even under strong restrictions on the input. Observe that the output can be exponential in the size of the query (specifically, in the length of the projection sequence $\mathbf{v}$), hence, under combined complexity, polynomial time is not enough for just writing the output. For such problems, the conventional yardstick of efficiency is *polynomial total time* [20], namely, the running time is polynomial in the size of both the input and the output. However, the following theorem rules out polynomial total time, by showing that query evaluation is intractable even if the query is Boolean. Moreover, this intractability holds even under strong restrictions on the SCFG and on the pattern.

THEOREM 5.8. *Computing $\Pr(\tau \mapsto \mathcal{P}_w^S)$, given a pattern τ, an SCFG Γ_S^p and a string $w \in \mathcal{L}(\Gamma_S)$ such that Γ_S is non-recursive and null-free, is $\mathrm{FP}^{\#P}$-complete; determining whether this probability is nonzero is NP-complete. Moreover, these hold even when adding one of the following assumptions.*

1. *Each label of τ has the form $/c$ (in particular, τ has no descendant edges) where c is $L = l$.*
2. *Γ_S is in Chomsky normal form.*
3. *Γ_S is linear.*

Recall that $\mathrm{FP}^{\#P}$ is the class of functions that are efficiently computable using an oracle to some function in #P. A function f is $\mathrm{FP}^{\#P}$-hard if there is a polynomial-time *Turing reduction* from every function in $\mathrm{FP}^{\#P}$ to f. Note that this is an intractable complexity class, since by using an oracle to a complete problem for this class one can solve every problem in the polynomial hierarchy [46].

6. EVALUATION ALGORITHM

In this section, we present an efficient algorithm for evaluating a query over a probabilistic parse represented by a weakly linear SCFG. Formally, we fix a query $\pi_{\mathbf{v}}\tau$, and the input for the problem consists of a weakly linear SCFG Γ_S^p and a string $w \in \mathcal{L}(\Gamma_S)$. The goal is to compute the set $\pi_{\mathbf{v}}\tau(\tilde{\mathcal{P}}_w^S)$, that is, the set of all pairs $(\mathbf{a}, \mathit{conf}_{\mathbf{a}})$, such that $\mathbf{a} \in \pi_{\mathbf{v}}\tau(t)$ for some $t \in \mathbf{PT}_w(\Gamma_S)$ and $\mathit{conf}_{\mathbf{a}} = \Pr(\mathbf{a} \in \pi_{\mathbf{v}}\tau(\mathcal{P}_w^S))$.

Our strategy will be to first introduce a probability space $\tilde{\mathcal{P}}^S$ of parses rooted at the symbol S, regardless of any string w (Section 6.1). Then, we reduce the problem at hand to that of computing the probability of an event over $\tilde{\mathcal{P}}^S$ (Section 6.2). Basically, this involves reducing the evaluation of general queries to that of determining satisfaction of Boolean queries. Next, we show how to normalize an SCFG, so that it has specific properties that make the formulation of the algorithm easier (Section 6.3). Finally, we present our algorithm for the reduced problem (Sections 6.4–6.7) and analyze the running time (Section 6.8).

6.1 The Probability Space $\tilde{\mathcal{P}}^S$

Let Γ_S^p be an SCFG. As shown in Example 4.4, the pair $(\mathbf{PT}(\Gamma_S), p)$ is not necessarily a probability space (since the random branching process defined by Γ_S^p does not necessarily terminate with probability 1). However, in this section we do wish to view Γ_S^p as a probability space over parses (regardless of any specific string). So, we add an artificial symbol $\perp_S$ that consumes all the "missing" probability. Formally, we assume that $\perp_S$ is a nonterminal that does not belong to $\mathbf{N}(\Gamma)$. We define $p(\perp_S) = 1 - \sum_{t \in \mathbf{PT}(\Gamma_S)} p(t)$. Then, the pair $(\mathbf{PT}(\Gamma_S) \cup \{\perp_S\}, p)$ is a probability space of parses rooted at S (and, also, the symbol $\perp_S$), and is denoted $\tilde{\mathcal{P}}^S$.

Consider an SCFG Γ_S^p and let $X \in \mathbf{N}(\Gamma_S)$ be a nonterminal. Note that Γ_X^p is an SCFG (i.e., the one obtained from Γ_S^p by using X, instead of S, as the start symbol). Thus, the function p is now also defined over $\mathbf{PT}(\Gamma_X)$. The probability space $\tilde{\mathcal{P}}^X$ is the pair $(\mathbf{PT}(\Gamma_X) \cup \{\perp_X\}, p)$, namely, the probability space consisting of all parses that are rooted at the symbol X (and, also, the symbol $\perp_X$). For convenience, we also define the probability space $\tilde{\mathcal{P}}^a$ for a terminal $a \in \mathbf{\Theta}(\Gamma)$, and then it is the fixed symbol a (which is a special case of a

parse). For $A \in \mathbf{N}(\Gamma) \cup \mathbf{\Theta}(\Gamma)$, we use $\mathcal{P}^A$ (i.e., without the tilde) to denote the random variable that represents a sample chosen from $\tilde{\mathcal{P}}^A$ (i.e., $\tilde{\mathcal{P}}^A$ and $\mathcal{P}^A$ relate to each other as $\tilde{\mathcal{P}}^S_w$ and $\mathcal{P}^S_w$ do).

In the remainder of this section, we assume a fixed query $\pi_{\mathbf{v}}\tau$ and a specific input w and Γ^p_S for the problem. (Thus, Γ^p_S is weakly linear and $w \in \mathcal{L}(\Gamma_S)$.)

6.2 Reduction to an Event Probability

We first reduce the problem to that of evaluating the probability of an event over $\tilde{\mathcal{P}}^S$. The reduction is rather standard (e.g., it is similar in spirit to that used in [9, 12, 24]). The idea is to produce a set $\mathcal{A}$ such that $\mathcal{A}$ contains all the possible answers $\mathbf{a}$ without their confidences (and possibly additional elements), and the size of $\mathcal{A}$ is bounded by a polynomial. For each element $\mathbf{a} \in \mathcal{A}$, we generate a pattern $\tau_{\mathbf{a}}$ such that $\Pr(\mathbf{a} \in \pi_{\mathbf{v}}\tau(\mathcal{P}^S_w))$ is equal to $\Pr(\tau_{\mathbf{a}} \mapsto \mathcal{P}^S_w)$, that is, the result of a Boolean query. Then, if $q = \Pr(\tau_{\mathbf{a}} \mapsto \mathcal{P}^S_w)$ is nonzero, we output the pair $(\mathbf{a}, q)$. Next, we give the details of how $\mathcal{A}$ and $\tau_{\mathbf{a}}$ are constructed.

Let $\mathbf{v} = (v_1, \ldots, v_k)$. The set $\mathcal{A}$ comprises all the sequences $(w_1, \ldots, w_k)$, where each w_i $(1 \le i \le k)$ is a substring of w. Various heuristics can be used for filtering out sequences $(w_1, \ldots, w_k)$ that have a zero probability (and, thus, do not affect the output). But even without those, $|\mathcal{A}|$ is bounded by n^{2k}, which is polynomial due to the assumption that $\pi_{\mathbf{v}}\tau$ (and in particular k) is fixed.

For a given $\mathbf{a} \in \mathcal{A}$, the pattern τ_a is constructed as follows. Recall that $\mathbf{v} = (v_1, \ldots, v_k)$. Let $\mathbf{a} = (w_1, \ldots, w_k)$ and suppose that each node v_i is labeled by the node constraint $c_i(L, W)$. Then $\tau_{\mathbf{a}}$ is obtained from τ by replacing each $c_i(L, W)$ with the conjunction $c_i(L, W) \wedge W = w_i$. It can be easily verified that, indeed,

$$\Pr(\mathbf{a} \in \pi_{\mathbf{v}}\tau(\mathcal{P}^S_w)) = \Pr(\tau_{\mathbf{a}} \mapsto \mathcal{P}^S_w)\,.$$

Using our reduction it follows that to solve the problem at hand, it is sufficient to be able to evaluate a Boolean query. Therefore, in the sequel we assume that $\mathbf{v}$ is empty and, hence, the goal is to compute the probability $\Pr(\tau \mapsto \mathcal{P}^S_w)$. The following holds.

$$\begin{aligned}\Pr\left(\tau \mapsto \mathcal{P}^S_w\right) &= \Pr\left(\tau \mapsto \mathcal{P}^S \mid str(\mathcal{P}^S) = w\right) \\ &= \frac{\Pr\left(\tau \mapsto \mathcal{P}^S \wedge str(\mathcal{P}^S) = w\right)}{\Pr\left(str(\mathcal{P}^S) = w\right)} \qquad (1)\end{aligned}$$

Consequently, it is enough to compute the numerator and the denominator of (1). We will show how to compute the numerator. The denominator can be viewed as a special case of the numerator with the pattern τ that matches every parse (i.e., *). Thus, we are left with the problem of computing

$$\Pr\left(\tau \mapsto \mathcal{P}^S \wedge str(\mathcal{P}^S) = w\right)\,. \qquad (2)$$

6.3 2-normalization

Our next step is that of *normalizing* Γ^p_S, where the goal is to transform it into a grammar with some specific desired properties. Formally, we transform Γ^p_S into an *equivalent* SCFG $\Upsilon^{p'}_{S'}$, such that $\Upsilon^{p'}_{S'}$ has the property that every sequence γ on the right-hand side of a rule $X \to \gamma$ has two or fewer symbols (i.e., γ comprises one or two symbols, or it is the empty string ϵ). We say that $\Upsilon^{p'}_{S'}$ is *2-normalized*, and the transformation presented below is called *2-normalization*. This type of transformation is a standard operation over CFGs (e.g., it is done when transforming a CFG into the Chomsky normal form). The problem with 2-normalization (or any transformation that changes the rules) is that it changes the parses and, therefore, can affect the final result of query evaluation. In particular, the probability that there is a match of τ in a random parse can change. Thus, we introduce *virtual* nonterminals that can be used to reverse the effect of normalization.

Formally, we assume that the global set $\mathbf{N}$ of nonterminals contains an infinite subset $\mathbf{VN}$ of *virtual* nonterminals. These nonterminals are not used in Γ^p_S (that is, $\mathbf{N}(\Gamma) \subseteq \mathbf{N} \setminus \mathbf{VN}$), and are only introduced by us (in the way we show below) for the technical reason of query evaluation. A *concrete* nonterminal is one that is not virtual. We use V and U to denote virtual nonterminals. Virtual nonterminals are used for 2-normalization, as follows.

We start with $\Upsilon^{p'}_{S'} = \Gamma^p_S$. We repeat the following until $\Upsilon^{p'}_{S'}$ is 2-normalized, while in each step we guarantee that $\Upsilon^{p'}_{S'}$ is weakly linear. We choose a rule $X \to \gamma$ such that γ contains three or more symbols. We introduce a fresh virtual nonterminal V. Note that at least one of the symbols of γ is not nullable (due to weak linearity). So, γ is either of the form $A\delta$ or δA, where δ contains two or more symbols, not all are nullable. In the first case, we replace $X \to \gamma$ with $X \to AV$, and in the second we replace it with $X \to VA$. We also add the rule $V \to \delta$. The probability $p'(X \to AV)$ (or $p'(X \to VA)$) is $p(X \to \gamma)$, and $p'(V \to \delta) = 1$. Finally, $S' = S$.

Example 6.1. As a simple example, consider the following SCFG Γ^p_S.

$$\begin{aligned} S &\to\ S\ X\ X\ (0.5) \mid a\ X\ (0.5) \\ X &\to\ \epsilon\ (0.5) \mid b\ (0.5) \end{aligned}$$

In the 2-normalized version of this SCFG, the rule $S \to S\ X\ X\ (0.5)$ would be replaced with two rules $S \to V\ X\ (0.5)$ and $V \to S\ X\ (1)$. We use V to replace SX, instead of XX, since the latter replacement would create an SCFG which is not weakly linear. □

Next, we present the reverse operation $\overset{\mathsf{vn}}{\downarrow}$ that removes virtual nonterminals from a parse. Let $\Upsilon^{p'}_{S'}$ be an SCFG that possibly uses virtual nonterminals. Let $X \in \mathbf{N}(\Upsilon)$ be a (concrete or virtual) nonterminal and let $t \in \mathbf{PT}(\Upsilon_X)$ be a parse. The operation $\overset{\mathsf{vn}}{\downarrow} t$ repeatedly removes virtual nonterminals from t until none is left; when a virtual nonterminal V is removed, its children (if exist) become the children of the parent of V, unless V is the root and then its children become roots. This means that if X if virtual, then $\overset{\mathsf{vn}}{\downarrow} t$ is not necessarily a tree, but rather a hedge (comprising zero or more trees). Formally, $\overset{\mathsf{vn}}{\downarrow} t$ is defined as follows. If $t = A(t_1 \cdots t_n)$ where $A \notin \mathbf{VN}$, then $\overset{\mathsf{vn}}{\downarrow} t$ is the tree $A(\overset{\mathsf{vn}}{\downarrow} t_1 \cdots \overset{\mathsf{vn}}{\downarrow} t_n)$. If $t = V(t_1 \cdots t_n)$ where $V \in \mathbf{VN}$, then $\overset{\mathsf{vn}}{\downarrow} t$ is the hedge $\overset{\mathsf{vn}}{\downarrow} t_1 \cdots \overset{\mathsf{vn}}{\downarrow} t_n$.

The following proposition, which is rather straightforward, shows that 2-normalization is indeed reversible by $\overset{\mathsf{vn}}{\downarrow}$. Moreover, weak linearity is preserved by 2-normalization. In the proposition, $\mathcal{P}^S$ and $\mathcal{P}^{S'}$ correspond to Γ^p_S and $\Upsilon^{p'}_{S'}$, respectively. Also, recall that Γ^p_S is weakly linear.

PROPOSITION 6.2. *Suppose that 2-normalizing Γ_S^p results in $\Upsilon_{S'}^{p'}$. Then $\Upsilon_{S'}^{p'}$ is 2-normalized and weakly linear. Moreover,* $\Pr(\mathcal{P}^S = t) = \Pr(\overset{\mathsf{vn}}{\downarrow}\mathcal{P}^{S'} = t)$ *for all* $t \in \mathbf{PT}$*; hence,* $\Pr(\tau \mapsto \mathcal{P}^S \wedge str(\mathcal{P}^S) = w) = \Pr(\tau \mapsto \overset{\mathsf{vn}}{\downarrow}\mathcal{P}^{S'} \wedge str(\mathcal{P}^{S'}) = w)$.

Thus, in the sequel, we assume that Γ_S^p is already 2-normalized (in addition to being weakly linear). In particular, Γ_S^p may include virtual nonterminals. Hence, due to Proposition 6.2, instead of computing Equation (2), we need to compute the following probability.

$$\Pr\left(\tau \mapsto \overset{\mathsf{vn}}{\downarrow}\mathcal{P}^S \wedge str(\mathcal{P}^S) = w\right) \tag{3}$$

REMARK 6.3. Virtual nonterminals are of an independent interest, since they allow the user to introduce nonterminals that are not semantically meaningful, and should be ignored during query evaluation. Our results immediately generalize to accommodate the existence of such nonterminals in the original Γ. For example, in the SCFG Γ_S^p of Figure 1, if AUS was defined as a virtual terminal, then from the standpoint of query processing, it would be as if parses have all AU nodes directly below BK (which is the fashion in which such data would naturally be modeled in XML). □

6.4 Computed Probabilities

Our approach is to compute, by dynamic programming, probabilities for a set of events. In the end, Equation (3) will be easily obtained from the computed probabilities. In this section, we describe this set of events.

We assume that the length of w is n. We denote by $w_{[s,e)}$, where $e \geq s$, the substring of w that starts at the sth symbol and ends with the $(e-1)$st symbol. In particular, $w_{[s,s)}$ is the empty string ϵ.

Consider the pattern τ. We use $Sub(\tau)$ to denote the set of sub-patterns of τ, that is, $Sub(\tau) = \{\tau_\Delta^v \mid v \in \mathcal{V}(\tau)\}$. For example, Figure 2 shows a pattern τ and its proper sub-patterns τ_a, τ_b, τ_c, $\tau_{a'}$, $\tau_{b'}$, and τ_d; thus, $Sub(\tau)$ is the set $\{\tau, \tau_a, \tau_b, \tau_c, \tau_{a'}, \tau_{b'}, \tau_d\}$. Given a subset $\mathcal{Q} \subseteq Sub(\tau)$, we will use $\overline{\mathcal{Q}}$ to denote the set $Sub(\tau) \setminus \mathcal{Q}$.

We straightforwardly extend the notion of a match of a pattern τ in a parse t to a match of τ in a hedge $h = t_1, \ldots, t_k$ of parses, as follows. A match of τ in h is a match of τ in t_i for some $1 \leq i \leq k$. Thus, $\tau \mapsto h$ means that there exists some i $(1 \leq i \leq k)$ such that $\tau \mapsto t_i$.

Consider a symbol A, a parse t, a set $\mathcal{Q} \subseteq Sub(\tau)$, and two integers s and e such that $e \geq s$. We write $t \Vdash_{s,e} \mathcal{Q}$ if all of the following three conditions hold: *(1)* $str(t)$ is $w_{[s,e)}$, *(2)* for all $\tau' \in \mathcal{Q}$, there is a match of τ' in $\overset{\mathsf{vn}}{\downarrow}t$, *and (3)* for all $\tau' \in \overline{\mathcal{Q}}$, there is *no* match of τ' in $\overset{\mathsf{vn}}{\downarrow}t$. The algorithm computes $\Pr(\mathcal{P}^A \Vdash_{i,j} \mathcal{Q})$ for all symbols $A \in \mathbf{N}(\Gamma) \cup \mathbf{\Theta}(\Gamma)$, subsets $\mathcal{Q}$ of $Sub(\tau)$, and integers s and e such that $1 \leq s \leq e \leq n+1$.

Recall that our goal is to compute the probability of (3). If every $\Pr(\mathcal{P}^A \Vdash_{s,e} \mathcal{Q})$ is computed, this can be done by to the following equality.

$$\Pr\left(\tau \mapsto \overset{\mathsf{vn}}{\downarrow}\mathcal{P}^S \wedge str(\mathcal{P}^S) = w\right) = \sum_{\{\mathcal{Q} \subseteq Sub(\tau) \mid \tau \in \mathcal{Q}\}} \Pr\left(\mathcal{P}^S \Vdash_{1,n+1} \mathcal{Q}\right)$$

So, it remains to compute the probabilities $\Pr(\mathcal{P}^A \Vdash_{s,e} \mathcal{Q})$. We show how this is done in the following sections.

6.5 Dynamic Program

Our approach employs the idea of the *inside algorithm*, which is an immediate adaptation of the CYK algorithm [7, 22, 49] (that determines acceptance of a string by a CFG) to the computation of the probability of a string in an SCFG. The inside algorithm is applicable to SCFGs in Chomsky normal form, and applies dynamic programming. However, the inside algorithm is heavily based on the assumption that the grammar is null free. Thus, our approach significantly differs in many key aspects from the inside algorithm.

In more detail, we compute the probabilities $\Pr(\mathcal{P}^A \Vdash_{s,e} \mathcal{Q})$ by dynamic programming, such that if $e' - s' < e - s$ (i.e., $w_{[s',e')}$ is shorter than $w_{[s,e)}$), then each $\Pr(\mathcal{P}^{A'} \Vdash_{s',e'} \mathcal{Q}')$ is computed before any $\Pr(\mathcal{P}^A \Vdash_{s,e} \mathcal{Q})$.

In the following sections, we fix s and e (where $1 \leq s < e \leq n+1$). We show how to compute $\Pr(\mathcal{P}^A \Vdash_{s,e} \mathcal{Q})$ for all A and $\mathcal{Q}$, assuming that every $\Pr(\mathcal{P}^{A'} \Vdash_{s',e'} \mathcal{Q}')$ has been computed for all e' and s' such that $e' - s' < e - s$. Note that this includes the first step where $e = s$ (and no probabilities are pre-computed). Essentially, applying this step of the dynamic program (i.e., computing the probabilities $\Pr(\mathcal{P}^A \Vdash_{s,e} \mathcal{Q})$ from the previously computed probabilities) entails establishment of linear relationships among the probabilities $\Pr(\mathcal{P}^A \Vdash_{s,e} \mathcal{Q})$ for the different A and $\mathcal{Q}$ (Section 6.6), and translation of these relationships into a non-singular system of linear equations (Section 6.7).

6.6 Top-Down Linear Relationships

We use $q_{A,\mathcal{Q}}^{s',e'}$ as a shorthand notation for $\Pr(\mathcal{P}^A \Vdash_{s',e'} \mathcal{Q})$. Our goal is to compute $q_{A,\mathcal{Q}}^{s,e}$ for all A and $\mathcal{Q}$. (Recall that we fixed s and e.) If A is a terminal symbol a, then the tree $\mathcal{P}^A$ comprises only the symbol a (and, in particular, it is deterministic), and we can directly test whether $\mathcal{P}^A \Vdash_{s,e} \mathcal{Q}$ (in which case, $q_{A,\mathcal{Q}}^{s,e} = 1$) or not (and then $q_{A,\mathcal{Q}}^{s,e} = 0$).

So, we need to compute $q_{X,\mathcal{Q}}^{s,e}$ for all nonterminals $X \in \mathbf{N}(\Gamma)$ and subsets $\mathcal{Q}$ of $Sub(\tau)$. Our approach is to reduce the computation of the $q_{X,\mathcal{Q}}^{s,e}$ to that of solving a system of linear equations. For that, we use the pre-computed probabilities. More particularly, we will show how to formulate $q_{X,\mathcal{Q}}^{s,e}$ as a linear combination other $q_{X',\mathcal{Q}'}^{s',e'}$, such that X' belongs to γ for some rule $(X \to \gamma) \in \Gamma$. Recall that we have pre-computed probabilities $q_{X',\mathcal{Q}'}^{s',e'}$ for which $e' - s' < e - s$, thus such probabilities are "known."

Let X be a nonterminal of Γ. Then X appears on the left side of rules of R and, moreover, p defines a probability space over the rules of the form $X \to \gamma$. Recall that, in a parse, the relationship between a node labeled X and its children corresponds to one of the rules $X \to \gamma$; that is, the children of the root form γ when their labels are concatenated. From the law of total probability, we have

$$q_{X,\mathcal{Q}}^{s,e} = \sum_{(X \to \gamma) \in R} p(X \to \gamma) \times r_{X \to \gamma}\,, \tag{4}$$

where $r_{X \to \gamma}$ is the probability that $\mathcal{P}^X \Vdash_{s,e} \mathcal{Q}$ holds given that the children of the root of $\mathcal{P}^X$ correspond to γ. Recall that γ comprises either zero, one or two symbols. Next, we show how to process each $r_{X \to \gamma}$ in (4).

6.6.1 Empty Rule

The first case is where $\gamma = \epsilon$, (i.e., the empty string). In this case, $\mathcal{P}^X$ is a deterministic hedge: if X is virtual,

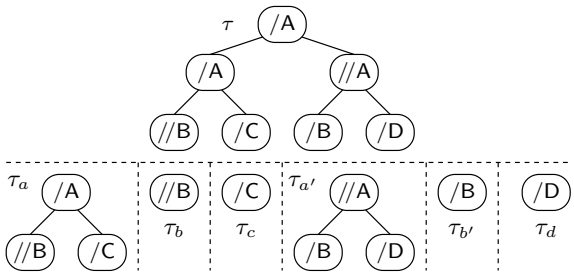

Figure 2: A pattern τ and its proper sub-patterns; note that $Sub(\tau) = \{\tau, \tau_a, \tau_b, \tau_c, \tau_{a'}, \tau_{b'}, \tau_d\}$

then $\mathcal{P}^X$ is the empty hedge; otherwise, it is the tree X. Therefore, $r_{X\to\gamma}$ is either 0 or 1, and we simply test which case it is.

6.6.2 Virtual Nonterminal

Now, we consider the case where X is a virtual nonterminal. Suppose first that $\gamma = A$ (i.e., γ comprises the single symbol A). It follows from the definition of a virtual nonterminal that in this case we have $r_{X\to\gamma} = q^{s,e}_{A,\mathcal{Q}}$. Thus, if A is a terminal, then $q^{s,e}_{A,\mathcal{Q}}$ is either 0 or 1, and it is determined above. Otherwise, if A is a nonterminal, then $r_{X\to\gamma} = q^{s,e}_{A,\mathcal{Q}}$.

Next, we suppose that $\gamma = A_1A_2$. The root of $\mathcal{P}^X$ has two children: the left parse is $t_1 \in \mathbf{PT}(\Gamma_{A_1})$ and the right is $t_2 \in \mathbf{PT}(\Gamma_{A_2})$. For $\mathcal{P}^X \Vdash_{s,e} \mathcal{Q}$ to be satisfied, the conjunction of the following three conditions is necessary and sufficient.

- There exists some k such that $str(t_1) = w_{[s,k)}$ and $str(t_2) = w_{[k,e)}$.
- For all $\tau' \in \mathcal{Q}$, there is a match of τ' in either t_1 or t_2.
- For all $\tau' \in \overline{\mathcal{Q}}$, there is a match of τ' in neither t_1 nor t_2.

Thus, $r_{X\to\gamma}$ is equal to

$$\Pr\left(\bigvee_{k=s}^{e} \bigvee_{\mathcal{Q}_1\cup\mathcal{Q}_2=\mathcal{Q}} t_1 \Vdash_{s,k} \mathcal{Q}_1 \wedge t_2 \Vdash_{k,e} \mathcal{Q}_2\right).$$

Now, different choices of k, $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are disjoint events. Furthermore, the parses t_1 and t_2 are probabilistically independent, and they are distributed by the probability spaces $\tilde{\mathcal{P}}^{A_1}$ and $\tilde{\mathcal{P}}^{A_2}$, respectively. Therefore, we conclude the following

$$r_{X\to\gamma} = \sum_{k=s}^{e} \sum_{\mathcal{Q}_1\cup\mathcal{Q}_2=\mathcal{Q}} q^{s,k}_{A_1,\mathcal{Q}_1} \times q^{k,e}_{A_2,\mathcal{Q}_2} \tag{5}$$

Observe that almost all the probabilities in the right-hand side of Equation (5) are pre-computed. In fact, the only unknowns in the right-hand side of Equation (5) are $q^{s,e}_{A_1,\mathcal{Q}_1}$ (when $k = e$) and $q^{s,e}_{A_2,\mathcal{Q}_2}$ (when $k = s$).[2] If $e = s$ (i.e., $w_{[s,e)} = \epsilon$), then $r_{X\to\gamma} = 0$, since either A_1 or A_2 is not nullable.

[2]At least one of these values is multiplied by 0, since at most one of A_1 and A_2 is nullable, and thus, either $q^{s,s}_{A_1,\mathcal{Q}_1} = 0$ or $q^{e,e}_{A_2,\mathcal{Q}_2} = 0$.

6.6.3 Concrete Nonterminal

Suppose now that X is a concrete nonterminal. There are two cases in which $\mathcal{P}^X \Vdash_{s,e} \mathcal{Q}$ is trivially false. First, there is some $\tau' \in \mathcal{Q}$ of the form $/c(\tau_1, \ldots, \tau_k)$, and $c(X, w_{[s,e)}) =$ **false**. Second, there is some single-node $\tau' \in \overline{\mathcal{Q}}$ of the form $/c$ or $//c$, and $c(X, w_{[s,e)}) =$ **true**. Thus, in both cases, we have $r_{X\to\gamma} = 0$.

We now suppose that the above conditions do not hold, and we introduce some notation. Let $\mathcal{Q}'$ be a subset of $Sub(\tau)$. We write $\mathcal{Q} \downarrow^X \mathcal{Q}'$ if all of the following conditions hold:

1. If $\tau' = /c(\tau_1 \cdots \tau_k) \in \mathcal{Q}$, then $\tau_l \in \mathcal{Q}'$ for all $1 \le l \le k$.
2. If $\tau' = //c(\tau_{q_1} \cdots \tau_{q_k}) \in \mathcal{Q}$, then either *(a)* $\tau' \in \mathcal{Q}'$, *or (b)* $c(X, w_{[s,e)}) =$ **true**, and $\tau_l \in \mathcal{Q}'$ for all $1 \le l \le k$.
3. If $\tau' = /c(\tau_1 \cdots \tau_k) \in \overline{\mathcal{Q}}$ and $c(X, w_{[s,e)}) =$ **true**, then there is some $1 \le l \le k$ such that $\tau_l \notin \mathcal{Q}'$.
4. If $\tau' = //c(\tau_1 \cdots \tau_k) \in \overline{\mathcal{Q}}$, then $\tau' \notin \mathcal{Q}'$; if in addition, $c(X, w_{[s,e)}) =$ **true**, then there is some $1 \le l \le k$ such that $\tau_l \notin \mathcal{Q}'$.

Example 6.4. Consider again the patterns of Figure 2. Let $\mathcal{Q} = \{\tau_a, \tau_b\}$, and let $X = \mathsf{A}$. Here, the node constraints are all of the form $L = l$ and, in particular, they do not involve $w_{[s,e)}$. Let $\mathcal{Q}'$ be such that $\mathcal{Q} \downarrow^X \mathcal{Q}'$. Note that there are different sets $\mathcal{Q}'$ such that $\mathcal{Q} \downarrow^X \mathcal{Q}'$. However, the four conditions above imply that $\mathcal{Q}'$ must (or must not) contain certain patterns (or combinations thereof). Condition 1 is applicable only for $\tau' = \tau_a$, and it requires both τ_b and τ_c to be in $\mathcal{Q}'$. Condition 2 is applicable only for $\tau' = \tau_b$, and it requires $\mathcal{Q}'$ to include τ_b (since X differs from B). Condition 3 is applicable only for $\tau' = \tau$, and it requires $\mathcal{Q}'$ to exclude either τ_a or $\tau_{a'}$ (or both). Finally, Condition 4 is applicable only for $\tau' = \tau'_a$, and it requires $\mathcal{Q}'$ to exclude $\tau_{a'}$ and either $\tau_{b'}$ or τ_d (or both). One can verify that $\mathcal{Q}'$ can be, for example, $\{\tau_b, \tau_c, \tau_{b'}\}$ or $\{\tau, \tau_b, \tau_c, \tau_d\}$. □

We now observe the following. Suppose that the root of $\mathcal{P}^X$ has a single child v labeled X'. Then $\mathcal{P}^X \Vdash_{s,e} \mathcal{Q}$ holds if and only if $t' \Vdash_{s,e} \mathcal{Q}'$ holds for the subtree t' of $\mathcal{P}^X$ that is rooted at the child v and for some $\mathcal{Q}'$ such that $\mathcal{Q} \downarrow^X \mathcal{Q}'$. Therefore, for the case where $\gamma = A$ we get that

$$r_{X\to\gamma} = \sum_{\mathcal{Q}\downarrow^X\mathcal{Q}'} q^{s,e}_{A,\mathcal{Q}'}. \tag{6}$$

Now suppose that $\gamma = A_1A_2$. Then, the root of $\mathcal{P}^X$ has two children labeled with A_1 and A_2, and these children are the roots of two subtrees t_1 and t_2, respectively. In this case, $\mathcal{P}^X \Vdash_{s,e} \mathcal{Q}$ holds if and only if both $t_1 \Vdash_{s,k} \mathcal{Q}_1$ and $t_2 \Vdash_{k,e} \mathcal{Q}_2$ hold for some k such that $s \le k \le e$, and some $\mathcal{Q}_1$ and $\mathcal{Q}_2$ that satisfy $\mathcal{Q} \downarrow^X (\mathcal{Q}_1 \cup \mathcal{Q}_2)$. Then we get:

$$r_{X\to\gamma} = \sum_{k=s}^{e} \sum_{\mathcal{Q}\downarrow^X(\mathcal{Q}_1\cup\mathcal{Q}_2)} q^{s,k}_{A_1,\mathcal{Q}_1} \times q^{k,e}_{A_2,\mathcal{Q}_2} \tag{7}$$

As in the case where X is virtual and $\gamma = A_1A_2$, the only unknowns in (7) are $q^{s,e}_{A_1,\mathcal{Q}_1}$ and $q^{s,e}_{A_2,\mathcal{Q}_2}$.

6.7 Finding the Probabilities

In the previous section, we established a set of linear equations over the values $q^{s,e}_{X,\mathcal{Q}}$. The reader can easily verify that the equations defining $q^{s,e}_{X,\mathcal{Q}}$ (Equations (4)–(7)), are indeed linear. This is correct, since each product $q^{s,k}_{A_1,\mathcal{Q}_1} \times q^{k,e}_{A_2,\mathcal{Q}_2}$ has at least one component whose value is known (since either $k-s < e-s$, $e-k < e-s$, or $e = k = s$ and then one of $q^{s,e}_{A_1,\mathcal{Q}_1}$ and $q^{s,e}_{A_2,\mathcal{Q}_2}$ is zero since Γ^p_S is weakly linear).

In order to derive the unknown values $q^{s,e}_{X,\mathcal{Q}}$, we must solve the linear equations from the previous section. To this end, we represent the unknown $q^{s,e}_{X,\mathcal{Q}}$ by the variable $x_{X,\mathcal{Q}}$. The vector $\mathbf{x}$ comprises the variables $x_{X,\mathcal{Q}}$ for all $X \in \mathbf{N}(\Gamma)$ and $\mathcal{Q} \subseteq Sub(\tau)$ (we assume some translation from a pair $(A, \mathcal{Q})$ to an integer index). Now, by replacing each unknown $q^{s,e}_{X,\mathcal{Q}}$ with $x_{X,\mathcal{Q}}$, we derive a system of linear equations over $\mathbf{x}$.

Let $m = |\mathbf{N}(\Gamma)| \times 2^{|Sub(\tau)|}$. We view $\mathbf{x}$ as an m-vector (that is, a column vector of dimension m). For all nonterminals $X \in \mathbf{N}(\Gamma)$ and sets $\mathcal{Q} \subseteq Sub(\tau)$, we have an equation of the form

$$x_{X,\mathcal{Q}} = (\mathbf{m}_{X,\mathcal{Q}})^{\mathsf{T}}\mathbf{x} + d_{X,\mathcal{Q}}$$

where $\mathbf{m}_{X,\mathcal{Q}}$ is an m-vector and $d_{X,\mathcal{Q}}$ is a (nonnegative) number. Let $\mathbf{M}'$ be the $m \times m$ matrix that comprises all the row m-vectors $(\mathbf{m}_{X,\mathcal{Q}})^{\mathsf{T}}$, and let $\mathbf{d}$ be the m-vector that comprises all the $d_{X,\mathcal{Q}}$. Finally, let $\mathbf{q}^{s,e}$ be the m-vector that comprises all the $q^{s,e}_{X,\mathcal{Q}}$.

Our goal is to find the vector $\mathbf{q}^{s,e}$. From the construction of $\mathbf{M}'$ it follows that $\mathbf{q}^{s,e}$ is a solution for $\mathbf{x} = \mathbf{M}'\mathbf{x} + \mathbf{d}$; that is, $\mathbf{q}^{s,e} = \mathbf{M}'\mathbf{q}^{s,e} + \mathbf{d}$. The problem is that there can be (infinitely) many (nonnegative) solutions $\mathbf{q}'$, where $\mathbf{q}^{s,e}$ is just one of them. So next, we will show how to transform $\mathbf{M}'$ into a new $m \times m$ matrix $\mathbf{M}$, so that $\mathbf{q}^{s,e}$ is the single solution for $\mathbf{x} = \mathbf{M}\mathbf{x} + \mathbf{d}$.

We start with detecting the zero elements of $\mathbf{q}^{x,e}$ (that is, X and $\mathcal{Q}$ such that $\Pr(\mathcal{P}^X \Vdash_{s,e} \mathcal{Q}) = 0$). We can efficiently do so in linear time in $|\Gamma|$ and $2^{|SubQ|}$ by a computation that is similar to that of the *closure* of database attributes under *functional dependencies* [2]. Now, we obtain the matrix $\mathbf{M}$ from $\mathbf{M}'$ by zeroing out every column and every row of $\mathbf{M}'$ that corresponds to a zero element $q^{s,e}_{X,\mathcal{Q}}$. The following lemma shows that $\mathbf{M}$ is, indeed, the matrix we need. Note that $\mathbf{I}_m$ is the $m \times m$ identity matrix. The proof, which uses recent results of [15], is based on showing that $\mathbf{q}^{s,e}$ is a *least fixed point* of the operator $\mathbf{M}'\mathbf{x} + \mathbf{d} : \mathbb{R}^m \to \mathbb{R}^m$.

LEMMA 6.5. *$\mathbf{M} - \mathbf{I}_m$ is non-singular. Moreover, $\mathbf{q}^{s,e}$ is the unique solution for $(\mathbf{M} - \mathbf{I}_m)\mathbf{x} = -\mathbf{d}$.*

As a consequence of Lemma 6.5, we can now complete the computation of $\mathbf{q}^{s,e}$ (and, hence, the whole algorithm) by means of inverting an $m \times m$ matrix (which can be done in $O(m^{2.376})$ time [11]). Next section, we discuss the efficiency of the algorithm.

6.8 Efficiency

It is fairly obvious from the description of the algorithm (in the previous sections) that its running time is polynomial if one simplistically assumes that each arithmetic operation has a fixed cost (e.g., as in the *rational Blum-Shub-Smale* computational model [3]). However, we need to show that the algorithm is efficient in a realistic model, that is, under *bit complexity*. To this end, it is sufficient to prove that the intermediate numbers do not explode in representation, that is, the number of bits used for representing each of the intermediate probabilities is upper bounded by a polynomial. This is shown in the next lemma. The proof of this lemma is by analyzing every generated linear system $(\mathbf{M} - \mathbf{I}_m)\mathbf{x} = -\mathbf{d}$, mainly based on applications of Cramer's rule.

LEMMA 6.6. *The number of bits required for representing each of the computed numbers $q^{s,e}_{A,\mathcal{Q}}$ is polynomial in w and Γ^p_S, and exponential in τ.*

This completes the proof of Theorem 5.7.

7. CONCLUSIONS

We formalized the concept of query evaluation over a probabilistic parse of a string, and focused on representation of probabilistic parses by means of SCFGs. We discussed the computational limitations of query evaluation, and we presented an efficient algorithm for evaluating tree-pattern queries (with projection) in the fairly general class of weakly linear SCFGs.

Although beyond the scope of this paper, our algorithm can be extended to richer queries that include negation and disjunction (in addition to conjunction), in the spirit of the queries of [8]. Moreover, it is possible to show that for acyclic SCFGs, query evaluation can be done within k-bit precision (i.e., the probabilities are approximated by a 2^{-k} additive factor) while the running time is polynomial in the input and in k (despite the fact that the exact result of a query would require an exponential number of bits to represent). An important direction for future work is the evaluation of queries under models of (stochastic) parsing that are different from SCFGs.

Acknowledgments

We thank the reviewers of this paper in ICDT 2010 for helpful comments and suggestions. We also thank Kenneth L. Clarkson and Dan Suciu for fruitful discussions.

The research of Sara Cohen is partially supported by the Israel Science Foundation (Grant 143/09) and the Israeli Ministry of Science and Technology (Grant 3-6472).

8. REFERENCES

[1] S. P. Abney, D. A. McAllester, and F. Pereira. Relating probabilistic grammars and automata. In *ACL*, 1999.

[2] C. Beeri and P. A. Bernstein. Computational problems related to the design of normal form relational schemas. *ACM Trans. Database Syst.*, 4(1):30–59, 1979.

[3] L. Blum, M.Shub, , and S. Smale. On a theory of computation and complexity over the real numbers: NP completeness, recursive functions, and universal machines. *Bull. A.M.S.*, 21:1–46, 1989.

[4] T. Booth and R. Thompson. Applying probability measures to abstract languages. *IEEE Trans. Computers*, C-22(5):442–450, 1973.

[5] N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: optimal XML pattern matching. In *SIGMOD*, pages 310–321. ACM, 2002.

[6] Z. Chi. Statistical properties of probabilistic context-free grammars. *Computational Linguistics*, 25(1):131–160, 1999.

[7] J. Cocke and J. I. Schwartz. Programming languages and their compilers. Technical report, Courant Institute of Mathematical Sciences, New York University, 1970.

[8] S. Cohen, Y. Kanza, Y. A. Kogan, Y. Sagiv, W. Nutt, and A. Serebrenik. Equix - a search and query language for xml. *JASIST*, 53(6):454–466, 2002.

[9] S. Cohen, B. Kimelfeld, and Y. Sagiv. Incorporating constraints in probabilistic XML. In *PODS*, pages 109–118, 2008.

[10] M. Collins and S. Miller. Semantic tagging using a probabilistic context free grammar. In *Processings of the Sixth Workshop on Very Large Corpora*, pages 38–48, 1997.

[11] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

[12] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[13] N. N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302, 2007.

[14] R. Dowell and S. Eddy. Evaluation of several lightweight stochastic context-free grammars for rna secondary structure prediction. *BMC Bioinformatics*, 5(71), 2004.

[15] K. Etessami and M. Yannakakis. Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. *J. ACM*, 56(1), 2009.

[16] R. Feldman, B. Rosenfeld, and M. Fresko. Teg-a hybrid approach to information extraction. *Knowl. Inf. Syst.*, 9(1):1–18, 2006.

[17] T. W. Hong and K. L. Clark. Using grammatical inference to automate information extraction from the web. In *5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*, pages 216–227, 2001.

[18] E. Hung, L. Getoor, and V. S. Subrahmanian. PXML: A probabilistic semistructured data model and algebra. In *ICDE*, pages 467–478, 2003.

[19] V. M. Jiménez and A. Marzal. Computation of the n best parse trees for weighted and stochastic context-free grammars. In *SSPR/SPR*, pages 183–192, 2000.

[20] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988.

[21] D. Jurafsky, C. Wooters, J. Segal, A. Stolcke, E. Fosler, G. Tajchman, and N. Morgan. Using a stochastic context-free grammar as a language model for speech recognition. In *IEEE International conference on acoustics speech and signal processing (ICASSP)*, 1995.

[22] T. Kasami. An efficient recognition and syntax analysis algorithm for context free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA, 1965.

[23] B. Kimelfeld, Y. Kosharovsky, and Y. Sagiv. Query efficiency in probabilistic XML models. In *SIGMOD Conference*, pages 701–714, 2008.

[24] B. Kimelfeld and Y. Sagiv. Matching twigs in probabilistic XML. In *VLDB*, pages 27–38, 2007.

[25] D. Klein and C. D. Manning. A* parsing: Fast exact viterbi parse selection. In *HLT-NAACL*, 2003.

[26] D. Klein and C. D. Manning. Accurate unlexicalized parsing. In *ACL*, pages 423–430, 2003.

[27] B. Knudsen and J. Hein. Pfold: Rna secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Research*, 31(13):3423–3428, 2003.

[28] C. Koch. Approximating predicates and expressive queries on probabilistic databases. In *PODS*, pages 99–108. ACM, 2008.

[29] J. Li and A. Deshpande. Consensus answers for queries over probabilistic databases. In *PODS*, pages 259–268. ACM, 2009.

[30] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[31] T. Matsuzaki, Y. Miyao, and J. Tsujii. Probabilistic CFG with latent annotations. In *ACL*, 2005.

[32] D. Moore and I. Essa. Recognizing multitasked activities using stochastic context-free grammar. In *In Proceedings of AAAI Conference*, 2001.

[33] D. J. Moore and I. A. Essa. Recognizing multitasked activities from video using stochastic context-free grammar. In *AAAI/IAAI*, pages 770–776, 2002.

[34] F. Neven. Automata, logic, and xml. In *CSL*, volume 2471 of *Lecture Notes in Computer Science*, pages 2–26. Springer, 2002.

[35] H. Ney. Dynamic programming parsing for context-free grammars in continuous speech recognition. *IEEE Transactions on Signal Processing*, 39(2):336–340, 1991.

[36] A. Nierman and H. V. Jagadish. ProTDB: Probabilistic data in XML. In *VLDB*, pages 646–657, 2002.

[37] A. Pauls and D. Klein. K-best a* parsing. In *ACL*, 2009.

[38] S. Petrov, L. Barrett, R. Thibaux, and D. Klein. Learning accurate, compact, and interpretable tree annotation. In *ACL*, 2006.

[39] D. Prescher. Inducing head-driven PCFGs with latent heads: Refining a tree-bank grammar for parsing. In *ECML*, pages 292–304, 2005.

[40] D. V. Pynadath and M. P. Wellman. Generalized queries on probabilistic context-free grammars. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(1):65–77, 1998.

[41] Y. Sakakibara, M. Brown, R. Hughey, I. Mian, K. Sjölander, R. Underwood, and D. Haussler. Stochastic context-free grammers for trna modeling. *Nucleic Acids Research*, 22(23):5112–5120, Nov. 1994.

[42] P. Senellart and S. Abiteboul. On the complexity of managing probabilistic XML data. In *PODS*, pages 283–292, 2007.

[43] I. Stewart. *Galois Theory*. Chapman & Hall, 2 edition, 1989.

[44] A. Stolcke. An efficient probabilistic context-free

parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201, 1995.

[45] F. M. Suchanek, G. Ifrim, and G. Weikum. Combining linguistic and statistical analysis to extract relations from web documents. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 712–717, 2006.

[46] S. Toda and M. Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 21(2), 1992.

[47] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, pages 137–146. ACM, 1982.

[48] C. S. Wetherell. Probabilistic languages: A review and some open questions. *ACM Comput. Surv.*, 12(4):361–379, 1980.

[49] D. Younger. Recognition and parsing of context-free languages in time n cubed. *Information and Control*, 10(2):189–208, 1967.