One-Counter Automata for Parsing and Language Approximation

Alexander Sakharov(✉)

Synstretch, Framingham, MA, USA
mail@sakharov.net

Abstract. A grammar characterization of partially blind one-counter languages is presented. One-counter automata are used to build parse trees of the respective grammars. One-counter automata are also used to find the most probable derivations for the stochastic versions of these grammars. Both these tasks are executed in quadratic time in the size of the input. Regular expressions are extended with one-counter language capabilities. Context-free languages are approximated with one-counter languages.

1 Introduction

The time complexity of parsing context-free (CF) languages is cubic in the size of the input in a general case, which includes the majority of ambiguous languages. See the CYK and Earley algorithms [1]. This time complexity is prohibitive for some applications. Sub-cubic parsing algorithms do exist but they are not practical [2].

CF grammars are often ambiguous. Natural language grammars are almost always ambiguous [1]. Ambiguous grammars are widely used in bioinformatics [3]. Selecting better derivations is an additional parsing challenge for these languages. Stochastic (aka probabilistic) grammars are an instrument for dealing with ambiguity. They associate probabilities with productions. Stochastic parsing amounts to finding the most probable derivations for given input strings [1]. Due to grammar splitting methods, stochastic parsing can be highly accurate, including natural language parsing [4].

Finding the most probable derivation is solved by dynamic programming algorithms such as the CYK algorithm for stochastic CF grammars [1]. The stochastic CYK algorithm also has a cubic time complexity in the size of the input. Enhanced algorithms for parsing stochastic CF languages execute much faster than the CYK algorithm in practice, but still exhibit a cubic asymptotic time complexity [5].

As opposed to CF languages, regular languages can be parsed in linear time by finite automata (FA). The task of finding the most probable derivation can be solved in linear time of the size of the input by the Viterbi algorithm [1]. The Viterbi algorithm finds the most probable sequence of automaton transitions (path) accepting the input for stochastic FAs. It can be applied to the

C. Câmpeanu (Ed.): CIAA 2018, LNCS 10977, pp. 299–311, 2018.
https://doi.org/10.1007/978-3-319-94812-6_25

automata recognizing right-linear, i.e. regular, grammars. The most probable path is also the most probable derivation in the respective right-linear grammar. Its time complexity makes the Viterbi algorithm the tool of choice for numerous applications, since many of them require a real-time result. Unfortunately, regular languages constitute a very limited subclass of CF languages. Having faster parsing algorithms for supersets of regular languages including algorithms for finding the most probable derivations is crucial.

One-counter (OC) automata are FAs extended with a non-negative counter [6]. The languages recognized by OC automata are a proper subset of CF languages and a proper superset of regular languages. OC automata have plenty of applications [7,8], and so do stochastic OC automata [9,10]. Stochastic OC automata are equivalent to discrete-time quasi-birth death processes [11].

Partially blind one-counter (PBOC) automata are OC automata without counter tests [12]. They are also known as restricted one-counter automata [13] and one-counter nets [14]. PBOC automata are equivalent to B-automata [15]. The languages recognized by PBOC automata are a proper superset of regular languages and a proper subset of OC languages. The properties of OC and PBOC languages have been extensively studied [6,13,14,16].

OC automata, including stochastic ones, have not been used much for parsing. Grammar characterization of OC and PBOC languages has been an open issue for decades [6,8]. We specify a class of grammars characterizing PBOC languages. We describe how to build matching PBOC automata from these grammars and vice versa. The time complexity of parsing the characterizing languages, including stochastic parsing, is quadratic in the size of the input. We augment regular expressions (RE) with OC capabilities. These extended REs give another characterization of PBOC languages.

The approximation of CF languages has recently caught much attention. CF languages are usually approximated with regular languages [17]. Fast parsing is one of the reasons of this approximation. Some advanced approximation methods have been developed [18].

We introduce a technique for approximating CF languages by OC automata. OC automata are more adequate for handling parenthesis constructs than FAs. This technique enables the generation of approximate parse trees. The languages accepted by the approximating automata include the source CF languages. We present a sufficient decidable condition for a CF grammar to be recognizable by a OC automaton. We describe a method for approximating automaton transition probabilities from grammar production probabilities.

2 One-Counter Automata

Definition 1. *A (nondeterministic) OC automaton is a tuple* (S, R, s, F, T) *where* S *is a finite set of states,* R *is a set of input symbols (alphabet),* $s \in S$ *is the start state,* $F \subset S$ *is a set of final states,* T *is a set of transitions. The transitions have the form:* $s, t, c \rightarrow r, n$ *where* s *and* r *are states,* t *is an input symbol,* $c \in \{0, +\}$ *is a counter test,* $n \in \{+1, -1, 0\}$ *is a counter operation.*

Transitions with $c = 0$ apply when the counter is zero, and transitions with $c = +$ apply when the counter is positive. The transition adds n to the counter value. Transitions with $c = 0$ and $n = -1$ are disallowed. The value of the counter is zero at the start. An input is accepted if the automaton is in a final state and the counter is zero. Transitions are classified as incrementing ($n = +1$), decrementing ($n = -1$), and internal ($n = 0$).

Some definitions of OC automata allow ϵ-transitions, i.e. transitions without input symbols. Alternatively, OC automata can be defined as pushdown automata with a single stack symbol and a special bottom symbol [11].

Definition 2. *A transition sequence is called balanced if the counter value at the beginning equals its value at the end, and the counter value at other states is not less than that.*

In the case of stochastic OC automata, probability p is associated with every transition [11]. It is assumed that transition probabilities satisfy the following conditions for any non-final state s:

$$\sum_{t,r,n} p(s,t,0 \to r,n) = 1 \qquad\qquad \sum_{t,r,n} p(s,t,+ \to r,n) = 1$$

For any final state, these sums are less than one by the final state probability. Some definitions of stochastic OC automata include start state probabilities.

The probability of an automaton path is defined as the product of the probabilities of its transitions. The Viterbi algorithm from [19] outputs the most probable path accepting the input of a stochastic OC automaton. Its time complexity is quadratic in the size of the input. This algorithm can be used to output acceptance paths for non-stochastic OC automata if we assign $p(s,t,c \to r,n) = 1$ instead of probabilities for all transitions.

Trees can be generated from acceptance paths of OC automata [19]. These trees play the role of parse trees. Input symbols are leaf nodes of these trees. The source states of incrementing and internal transitions label non-leaf nodes. The source states of decrementing transitions could be leaf nodes, or these states could have one child. Both interpretations are based on the intuitive assumption that any incrementing transition opens a construct, and a matching decrementing transition closes it.

In order to build a tree, we iterate over transitions in an acceptance path and maintain the stack of states. For any transition $A, u, c \to B, 0$, u and B are the children of A. For any transition $A, u, c \to B, +1$, u and B are the first two children of A. A is pushed onto the stack. For any transition $C, v, + \to D, -1$, the top state is popped from the stack. Under the first interpretation, C has no child nodes, v and D are added as children of the node popped from the stack. Under the second interpretation, v becomes the sole child of C, and D is added as a child of the node popped from the stack.

Definition 3. *A OC automaton is called PBOC if the following holds for any s, t, r, h: the automaton has transition $s, t, 0 \to r, h$ iff it has transition $s, t, + \to r, h$ (cf. [12]).*

Usually, PBOC automata are defined by transitions without the counter test, and the automata halt when the counter becomes negative. Following this tradition, we drop the counter test from the notation of PBOC automaton transitions. PBOC automata can also be defined as pushdown automata with a single stack symbol and without bottom symbol transitions [13].

3 Counting Regular Expressions

REs are regarded as a simpler notation than CF grammars. REs are also more widely used [20]. The notion of a counter can be incorporated into REs without affecting their simplicity but enhancing the expressiveness. OC automata can be used to parse these extended REs, which will be called counting regular expressions (CRE), and to generate trees. Let $\mathcal{L}$ denote the language defined by a RE, grammar, or automaton.

CREs are defined as REs in which some terminals may be annotated with plus or minus, for instance, a^+ or a^- for terminal a. A string matches a CRE if the following three conditions are met. First, the string matches this CRE with the annotations ignored. Second, the number of input symbols matching the plus terminals should be greater or equal to the number of input symbols matching the minus terminals for any partial input, i.e. for any substring $n_1...n_k$ of input $n_1...n_m$. Third, these two numbers should be equal for the entire input.

For example, the following CRE specifies additive expressions (terminals are underlined, terminal $\underline{id}$ represents identifiers):

$$(\underline{(}^{+})^{*}\ \underline{id}\ (\underline{)}^{-})^{*}\ ((\underline{+} \mid \underline{-})\ (\underline{(}^{+})^{*}\ \underline{id}\ (\underline{)}^{-})^{*})^{*}$$

Theorem 1. *The sets of languages defined by CREs and PBOC automata are identical.*

Proof. We can view any CRE X as a RE in which a, a^+, a^- are treated as distinct terminals even though they match the same input. Let us convert this RE to its recognizing nondeterministic FA without ϵ-transitions X' [21]. Now we can transform X' transitions into transitions of PBOC automaton X'' as follows:

$$\begin{aligned} s, x \to t \quad &\Rightarrow \quad s, x \to t, 0 \\ s, x^{+} \to t \quad &\Rightarrow \quad s, x \to t, +1 \\ s, x^{-} \to t \quad &\Rightarrow \quad s, x \to t, -1 \end{aligned}$$

The start and final states remain unchanged. Any string $s \in \mathcal{L}(X)$ is accepted by X'. Using the above rules, we can transform any X' acceptance path into a X'' path. It is also an acceptance path for X'' because it is balanced, and the counter is zero at the end. Now suppose string u is accepted by X''. It is also accepted by X'. Since the input symbols matching annotated terminals of X satisfy the CRE conditions, $u \in \mathcal{L}(X)$.

These transformations can be reverted to transform any PBOC automaton to a FA with annotated terminals. Any FA can be transformed to an equivalent RE. Annotations are carried from the FA to this RE which is treated as a CRE. We can use the same arguments as before to show that this CRE and the source PBOC automaton define the same language. □

Using the Viterbi algorithm for OC automata and the first interpretation of acceptance paths, we can build parse trees for the strings matching CRE.

4 Compartmentalized Grammars

Without loss of generality, we can assume that CF grammar productions are in the quadratic Greibach normal form (QGNF), i.e. every production is $A \to bB_1...B_k$ where $0 \leq k \leq 2$ [6]. As usual, $=>^*$ denotes grammar derivation. Let $\mathcal{R}(A)$ be the set consisting of nonterminal A and all such nonterminals B that $A =>^* \alpha B$ where α is a string of terminals and/or nonterminals. A simple iterative procedure can calculate $\mathcal{R}(A)$ for all nonterminals A of any CF grammar. Let $\mathcal{I}$ be the set of nonterminals D such that there is production $A \to bB_1B_2$, and $D \in \mathcal{R}(B_1)$.

Consider a OC automaton whose states are grammar nonterminals. Additionally, there is one and only final state Z that does not map to any nonterminal. Start nonterminal S is the start state. Transitions are constructed as follows:

For every production $A \to bB_1$:
$A, b, 0 \to B_1, 0 \quad$ if $A \in \mathcal{R}(S)$
$A, b, + \to B_1, 0 \quad$ if $A \in \mathcal{I}$
For every production $A \to bB_1B_2$:
$A, b, 0 \to B_1, +1 \quad$ if $A \in \mathcal{R}(S)$
$A, b, + \to B_1, +1 \quad$ if $A \in \mathcal{I}$
For every production $A \to bB_1B_2$ and production $D \to d$ such that $D \in \mathcal{R}(B_1)$:
$D, d, + \to B_2, -1$
For every production $D \to d$ such that $D \in \mathcal{R}(S)$:
$D, d, 0 \to Z, 0$

Theorem 2. *If OC automaton Ω is built by the above rules from CF grammar Γ in QGNF, then $\mathcal{L}(\Gamma) \subseteq \mathcal{L}(\Omega)$.*

Proof. Consider the parse tree of an input string from $\mathcal{L}(\Gamma)$. Let us traverse this parse tree in pre-order. Nonterminals and terminals alternate in the traversal sequence. Every triple A, b, C in the sequence (where b is a terminal) corresponds to a transition of the automaton generated from the grammar. Triples originating from productions $A \to tB$ map to transitions $A, t, n \to B, 0$ where $n = 0$ or $n = +$ depending on the counter value. Productions $A \to uBC$ are the source of transitions $A, u, n \to B, +1$. Along with productions $D \to c$ where $D \in \mathcal{R}(B)$, they are also the source of transitions $D, c, + \to C, -1$. In both cases, $A \in \mathcal{R}(S)$ if $n = 0$, and $A \in \mathcal{I}$ if $n > 0$.

The destination state of every transition in the traversal sequence equals to the source state of the next one. Incrementing and decrementing transitions are paired according to parse tree nodes for productions $A \to uBC$. Therefore, the counter value is always non-negative and equals zero at the end. The counter is always positive at the source states of decrementing transitions. If A, b is the last pair in the traversal sequence, then $A \in \mathcal{R}(S)$, and this pair maps to transition $A, b, 0 \to Z, 0$. Therefore, the input is accepted by Ω. □

We also can construct a PBOC automaton from productions of any grammar in QGNF. Let both $A, u, 0 \to B, h$ and $A, u, + \to B, h$ be generated regardless of whether $A \in \mathcal{R}(S)$, $A \in \mathcal{I}$, or not. Clearly, Theorem 2 holds after this change.

Definition 4. *Grammar Γ in QGNF is called compartmentalized quadratic Greibach (CQG) if for any two productions $A \to bBC$ and $E \to fFG$, $\mathcal{R}(B) \cap \mathcal{R}(F) = \emptyset$. If for any such productions, either $\mathcal{R}(B) \cap \mathcal{R}(F) = \emptyset$ or the grammar includes productions $A \to bBG$ and $E \to fFC$ as well, then Γ is called semi-compartmentalized (SCQG).*

For example, the following grammar is CQG:

$S \to id$ $\quad S \to id\ S$ $\quad S \to (SR$ $\quad R \to)$ $\quad R \to)S$

Here is the PBOC automaton constructed from this grammar:

$S, id \to S, 0$ $\quad R,) \to S, 0$ $\quad S, (\to S, +1$

$S, id \to R, -1$ $\quad R,) \to R, -1$ $\quad S, id \to Z, 0$ $\quad R,) \to Z, 0$

Theorem 3. *If OC automaton Ω is built by the above rules from SCQG grammar Γ, then $\mathcal{L}(\Gamma) = \mathcal{L}(\Omega)$.*

Proof. Let $\overleftarrow{t}$ and $\overrightarrow{t}$ denote the source and destination state of transition t, respectively. We prove by induction on the number of incrementing transitions that if $t_1...t_n$ is a balanced sequence of transitions of Ω for input string $s_1...s_n$ and $\overrightarrow{t_n} \neq Z$, then $\overleftarrow{t_1} \Rightarrow^* s_1...s_n\overrightarrow{t_n}$ is a valid derivation in Γ, and $\overrightarrow{t_n} \in \mathcal{R}(\overleftarrow{t_1})$.

Base: Clearly, this proposition holds for balanced sequences without incrementing transitions.

Induction step: Suppose the proposition holds for sequences with not more than m incrementing transitions. Consider the first incrementing transition t_i in sequence $t_1...t_n$ with $m + 1$ incrementing transitions. Let t_j be its balancing decrementing transition.

Suppose t_i corresponds to production $A \to s_iB_1B_2$, and t_j corresponds to production pair $E \to s_j$, $C \to dD_1D_2$ such that $E \in \mathcal{R}(D_1)$. By the induction assumption, transition sequence $t_{i+1}...t_{j-1}$ maps to derivation $B_1 \Rightarrow^* s_{i+1}...s_{j-1}E$, and $E \in \mathcal{R}(B_1)$. Since $E \in \mathcal{R}(D_1)$, Γ contains production $A \to s_iB_1D_2$ as well. Transition t_i is identical to the transition generated from production $A \to s_iB_1D_2$, and t_j is identical to the transition generated from production pair $E \to s_j$, $A \to s_iB_1D_2$, and hence, $A \Rightarrow^* s_i...s_jD_2$.

By the induction assumption, $D_2 \Rightarrow^* s_{j+1}...s_n\overrightarrow{t_n}$ is a valid derivation, and $\overrightarrow{t_n} \in \mathcal{R}(D_2)$. Transitions $s_1...s_{i-1}$ are all internal, and thus, $\overleftarrow{t_1} \Rightarrow^* s_1...s_{i-1}A$, $A \in \mathcal{R}(\overleftarrow{t_1})$. Combining the three derivations, we get $\overleftarrow{t_1} \Rightarrow^* s_1...s_n\overrightarrow{t_n}$, and $\overrightarrow{t_n} \in \mathcal{R}(\overleftarrow{t_1})$.

If $t_1...t_k$ is an acceptance path, then $t_1...t_{k-1}$ is balanced, $S \Rightarrow^* s_1...s_{k-1}\overrightarrow{t_{k-1}}$ is a valid derivation in Γ, and $\overrightarrow{t_{k-1}} \in \mathcal{R}(S)$. Γ has production $\overrightarrow{t_{k-1}} \to s_k$ because $\overrightarrow{t_k} = Z$. Hence, $S \Rightarrow^* s_1...s_k$. □

Theorem 3 also holds for the PBOC automata built from SCQG grammars. Using the second interpretation of OC automaton acceptance paths, we can build trees from acceptance paths for the OC automata built from CF grammars in QGNF. These trees contain relevant syntactic information even if they do not exactly match productions of the source grammar. As the proof of Theorem 3 shows, the trees constructed from acceptance paths can be converted to the parse trees of the source SCQG grammars.

5 Stochastic Parsing

Any proper CQG grammar has no more than one distinct production $A \to bBC$ for any triple A, b, B. For any production $D \to c$ from a CQG grammar, there is no more than one such production $A \to bBC$ that $D \in \mathcal{R}(B)$.

Transition probabilities of the OC automata built from stochastic CQG grammars are expressed via grammar production probabilities:

$$p(A, b, 0 \to B, 0) = p(A, b, + \to B, 0) = p(A \to bB)$$
$$p(A, b, 0 \to B, +1) = p(A, b, + \to B, +1) = p(A \to bBC)$$
$$p(A, b, + \to B, -1) = p(A \to b) \qquad p(A, b, 0 \to Z, 0) = p(A \to b)$$

In stochastic grammars, the sum of probabilities of the productions $A \to ...$ equals one for every nonterminal A [1]. For every state A except Z, the sum of $p(A, b, + \to B, n)$ for all b, B, n equals one because the sum of the respective production probabilities equals one. The same is true about the sum of $p(A, b, 0 \to B, n)$ for all b, B, n.

If a grammar undergoes a transformation, then the probabilities of new productions usually cannot be expressed via the probabilities of original productions. The following theorem guarantees that the Viterbi algorithm from [19] can be used to find the most probable derivations for stochastic CQG grammars.

Theorem 4. *Suppose stochastic OC automaton Ω is built from stochastic CQG grammar Γ, and their probabilities satisfy the above equations. A derivation in Γ is the most probable iff the respective Ω acceptance path is the most probable.*

Proof. The probability of a grammar derivation is calculated as the product of the probabilities of the productions in the respective parse tree. The probability of an acceptance path is the product of the probabilities of its transitions. There is one-to-one mapping between nonterminal nodes of Γ parse trees and transitions of Ω acceptance paths. The probabilities of the Γ productions associated with parse tree nodes and the probabilities of the respective Ω transitions equal each other. The most probable derivations corresponds to the most probable acceptance paths for CQG grammars and their counterpart stochastic OC automata because the probabilities of the derivations and acceptance paths are calculated as the products of the same values. □

6 Grammar Characterization

We can transform any PBOC automaton into a CF grammar. Let us assign a unique positive number (identifier) to every distinct transition pair

$(A, b \to B, +1,\ D, c \to C, -1)$. We define nonterminal $[B, r]$ for every state B and every transition pair identifier r. Also, we define nonterminal $[B, 0]$ for every state B where 0 is a dummy identifier. If S is the start state, then $[S, 0]$ is the start nonterminal. The following rules define grammar productions. ϵ denotes the empty string.

1. $[A, r] \to b\ [B, r]$ for every transition $A, b \to B, 0$ and every identifier r
2. $[A, r] \to b\ [B, u]\ c\ [C, r]$ for every identifier $u = (A, b \to B, +1,\ D, c \to C, -1)$ and every identifier r
3. $[D, u] \to \epsilon$ for every identifier $u = (A, b \to B, +1,\ D, c \to C, -1)$
4. $[A, 0] \to \epsilon$ for every final state A

Lemma 1. *If grammar Γ is built from PBOC automaton Ω by the above rules, then $\mathcal{L}(\Gamma) = \mathcal{L}(\Omega)$.*

Proof. 1. $\mathcal{L}(\Gamma) \subseteq \mathcal{L}(\Omega)$

Consider an arbitrary parse tree for Γ and traverse it in pre-order. Every triple $[A, r], b, [B, r]$ in the traversal string corresponding to production $[A, r] \to b\ [B, r]$ has source transition $A, b \to B, 0$. Let $[A, r], b, [B, u]$ and $[E, v], c, [C, r]$ be the first and the last triple corresponding to production $[A, r] \to b\ [B, u]\ c\ [C, r]$ in the parse tree. This production is induced by transition pair $u = (A, b \to B, +1,\ D, c \to C, -1)$. $u = v$ because $[E, v] \in \mathcal{R}([B, u])$, and all elements of $\mathcal{R}([B, u])$ share the same identifier. Hence, $E = D$.

Every triple maps to an Ω transition. The destination state of every transition in the traversal sequence equals to the source state of the next one. Incrementing and decrementing transitions are paired according to parse tree nodes for productions $[A, r] \to b\ [B, u]\ c\ [C, r]$. Therefore, the counter value is always non-negative, and it equals zero at the end. The counter value is always positive at nodes $[D, u]$ corresponding to the source states of decrementing transitions. If $[C, s]$ is the last node in the traversal sequence, then $s = 0$, and hence, C is a final state. Therefore, the sequence of triples from the traversal maps to an Ω acceptance path.

2. $\mathcal{L}(\Omega) \subseteq \mathcal{L}(\Gamma)$

We can prove by induction on the number of incrementing transitions that if $t_1 ... t_n$ is a balanced sequence of transitions of Ω for input string $s_1 ... s_n$, and r is an identifier, then $[\overleftarrow{t_1}, r] \Rightarrow^* s_1 ... s_n [\overrightarrow{t_n}, r]$ is a valid derivation in Γ. The fact that the input string of any Ω acceptance path is derivable in Γ is a corollary of that because the last state is final, and we can pick $r = 0$.

The proof is similar to the proof of Theorem 3. The difference is the following. If t_i is the first incrementing transition, and t_j is its balancing decrementing transition, then production $[\overleftarrow{t_j}, u] \to \epsilon$ is generated for transition pair $u = (t_i, t_j)$. Productions $[\overleftarrow{t_i}, v] \to s_i\ [\overleftarrow{t_{i+1}}, u]\ s_j\ [\overrightarrow{t_j}, v]$ are generated for $u = (t_i, t_j)$ and for every v. Combining these productions with derivation $[\overleftarrow{t_{i+1}}, u] \Rightarrow^* s_{i+1} ... s_{j-1}$ $[\overrightarrow{t_{j-1}}, u]$, we get $[\overleftarrow{t_i}, v] \Rightarrow^* s_i ... s_j\ [\overrightarrow{t_j}, v]$. □

Theorem 5. *The sets of languages defined by CQG grammars, SCQG grammars, and PBOC automata are identical.*

Proof. The Theorems 2 and 3 reformulated for PBOC automata show that an equivalent PBOC automaton, i.e. defining the same language, can be constructed for any SCQG grammar. Lemma 1 shows that an equivalent CF grammar can be constructed for any PBOC automaton. Consider any two productions $[A, r] \rightarrow b\ [B, u]\ c\ [C, r]$ and $[E, s] \rightarrow f\ [F, v]\ g\ [G, s]$ of the grammar built from a PBOC automaton. If $u = v$, then these productions are identical. If the two productions are distinct, then $\mathcal{R}([B, u]) \cap \mathcal{R}([F, v]) = \emptyset$ because all elements of $\mathcal{R}([X, z])$ share the same identifier z.

The grammars built from PBOC automata can be easily transformed into equivalent grammars in QGNF. First, every production $A \rightarrow bBcC$ is transformed into pair $A \rightarrow bBC'$, $C' \rightarrow cC$ where C' is a new nonterminal. Second, productions $A \rightarrow b$ and $A \rightarrow bB$ are added in lieu of productions $D \rightarrow \epsilon$. These additional productions are the result of removing D from the right-hand sides of productions. The resulting grammar defines the same language, it is CQG, and thus, SCQG as well. □

Any CQG grammar can be converted to an equivalent PBOC automaton and vice versa. The same is true about CREs and PBOC automata. Hence, any CRE can be converted to an equivalent CQG grammar and vice versa.

7 Approximation of Context-Free Languages

Theorem 3 gives an indication that OC automata better approximate grammars with fewer productions $A \rightarrow bB_1B_2$. We call them long productions. For instance, grammars having one long production are recognizable by OC automata.

Definition 5. *Nonterminal A from a CF grammar in QGNF is called regular if no $E \in \mathcal{R}(A)$ has long productions or if all nonterminals from the right-hand side of every A production are regular.*

Regular nonterminals can be effectively identified. If the start nonterminal is regular, then the grammar is regular. The following grammar transformation may boost the accuracy of the approximation by OC automata.

If B_1 is a regular nonterminal, then productions $A \rightarrow bB_1B_2$ can be eliminated at the expense of newly introduced nonterminals and productions of the form $C \rightarrow dD$. Let us start with such B_1 that no $D \in \mathcal{R}(B_1)$ has long productions. First, we replicate all these D along with all their productions and replace all replicas $E \rightarrow c$ with productions $E \rightarrow cB_2$. Second, we replace $A \rightarrow bB_1B_2$ with $A \rightarrow bB'$ where B' is the replica of B_1. By applying the above transformation iteratively, we eliminate all long productions with regular nonterminals B_1. This transformation does not introduce new long productions and does not change the language defined by the grammar.

Any proper CF grammar can be effectively converted to QGNF [6]. After that, the aforementioned transformation can be applied. Finally, we check if the transformed grammar is SCQG. Therefore, we have a sufficient condition for CF languages to be recognizable by PBOC automata. This condition can be effectively verified for an arbitrary CF grammar, and the recognizing automaton can be built when the condition is met.

Now we outline an approach to approximating transition probabilities of stochastic OC automata generated from stochastic CF grammars in QGNF. The transition probabilities can be learned from grammar production probabilities. Describing implementation details and estimating the accuracy of this approximation is beyond the scope of this paper.

A training set can be easily created. Strings belonging to the language are generated by constructing random derivations from the start nonterminal. For this purpose, productions are randomly applied by taking into account their probabilities. We calculate the probabilities of the generated strings from their parse trees. We generate acceptance paths of the approximating OC automaton from the parse trees as it is done in the proof of Theorem 2.

Let us build a model for the approximation of the probabilities of transitions. w_i will denote the probability of transition i. If there is only one transition t for source state a and counter test c, then $w_t = 1$. Otherwise, let us pick one transition among all transitions for given a and c. Note that a is not final. Let X be the set of these picked transitions, Y be its complement, and $C(i)$ denote the set of complementary transitions for transition $i \in X$. The probabilities of transitions from Y are model parameters.

The model equation is:

$$z = \prod_{i \in Y} w_i^{m_i} \prod_{i \in X} (1 - \sum_{j \in C(i)} w_j)^{m_i}$$

where m_i is the number of occurrences of transition i in the acceptance path, z is the probability of the corresponding grammar derivation. This model also has the following box constraints: $0 \leq w_i \leq 1$ for $i \in Y$.

We can symbolically calculate $\frac{\partial z}{\partial w_i}$ for $i \in Y$. Therefore, the model parameters w_i can be learned by gradient descent methods, e.g. stochastic gradient descent with square loss or another loss function [22]. Projections onto sets defined by box constraints are trivial. It is well-known that stochastic gradient descent gives robust results for a variety of optimization problems like this [22].

8 Related Work

The author is unaware of any previous work investigating the relationship between OC automata and grammars except for XML grammars. PBOC automata are used to validate XML documents against certain recursive DTDs in [8].

Recursive REs aim to incorporate CF features into REs [20]. They extend and complicate the notation of REs. Recursive REs disallow backtracking within recursive calls [20], i.e. recursive REs are not declarative unlike REs and CF grammars. Our extension, i.e. CREs, is less ambitious, but it adds the power of

OC languages while preserving the simplicity of REs and retaining their declarative nature.

RE parsers usually produce parse trees on the basis of the structure of a given RE [23]. CRE parsing employs annotated terminals as opening and closing markers for parse tree nodes.

Regular approximation of CF languages usually leads to the loss of syntactic information. A grammar defining arithmetic expressions is used as an illustrating example in [17]. Its approximating FA has only two states, and thus, its acceptance paths do not carry much syntactic information.

It is claimed in [17] that the approximating FAs can be used for parsing, but the parse trees reconstructed from acceptance paths of the approximating automaton differ from the parse trees of the source language. Another method for reconstructing parse trees from acceptance paths of the approximating automata was proposed in [24]. The problem with this method is that the reconstruction requires cubic time, which defeats the purpose of language approximation.

A method of learning transitions probabilities of stochastic FAs was proposed in [25]. This method works only for unambiguous FAs, while stochastic automata and grammars are expected to be ambiguous. In general, unambiguous automata do not need stochastic methods.

9 Conclusion

The characterization of PBOC languages by CREs and CQG grammars is an indication of relevance of PBOC automata to parsing. However, the approximation of CF languages could potentially be more accurate if it involves all OC automata. CQG languages is a new addition to the sparse collection of subclasses of ambiguous or stochastic CF languages that can be parsed in quadratic time or faster.

The approximation of CF languages by OC automata has the potential to be more realistic than the approximation by regular languages. The latter is inhibited by the limitations of regular languages. Our approximation applies to stochastic CF grammars as well. Language approximation makes more sense for stochastic languages because stochastic parsing is inherently approximate.

References

1. Jurafsky, D., Martin, J.H.: Speech and Language Processing, 2nd edn. Prentice-Hall Inc., Upper Saddle River (2009)
2. Lee, L.: Fast context-free grammar parsing requires fast boolean matrix multiplication. J. ACM **49**(1), 1–15 (2002)
3. Dowell, R.D., Eddy, S.R.: Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. BMC Bioinformatics **5**(1), 71 (2004)
4. Petrov, S., Barrett, L., Thibaux, R., Klein, D.: Learning accurate, compact, and interpretable tree annotation. In: Proceedings of the 21st International Conference on Computational Linguistics, pp. 433–440 (2006)

5. Klein, D., Manning, C.D.: A* parsing: fast exact Viterbi parse selection. In: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1, pp. 40–47 (2003)
6. Autebert, J., Berstel, J., Boasson, L.: Context-free languages and push-down automata. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Springer, Heidelberg (1997). https://doi.org/10.1007/978-3-642-59136-5_3
7. Bouajjani, A., Bozga, M., Habermehl, P., Iosif, R., Moro, P., Vojnar, T.: Programs with lists are counter automata. Formal Methods Syst. Des. **38**(2), 158–192 (2011)
8. Chitic, C., Rosu, D.: On validation of XML streams using finite state machines. In: Proceedings of the 7th International Workshop on the Web and Databases, pp. 85–90 (2004)
9. Brázdil, T., Brozek, V., Etessami, K., Kucera, A., Wojtczak, D.: One-counter Markov decision processes. In: Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 863–874 (2010)
10. Brázdil, T., Brozek, V., Etessami, K.: One-counter stochastic games. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, pp. 108–119 (2010)
11. Etessami, K., Wojtczak, D., Yannakakis, M.: Quasi-birth-death processes, tree-like QBDs, probabilistic 1-counter automata, and pushdown systems. In: 5th International Conference on the Quantitative Evaluation of Systems, pp. 243–253 (2008)
12. Greibach, S.A.: Remarks on blind and partially blind one-way multicounter machines. Theoret. Comput. Sci. **7**, 311–324 (1978)
13. Berstel, J.: Transductions and Context-Free Languages. Leitfäden der angewandten Mathematik und Mechanik. Teubner (1979)
14. Czerwinski, W., Lasota, S.: Regular separability of one counter automata. In: 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, pp. 1–12 (2017)
15. Render, E., Kambites, M.: Polycyclic and bicyclic valence automata. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 464–475. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88282-4_42
16. Brandenburg, F.J.: On the intersection of stacks and queues. Theoret. Comput. Sci. **58**(1), 69–80 (1988)
17. Mohri, M., Nederhof, M.J.: Regular approximation of context-free grammars through transformation. In: Junqua, J.C., van Noord, G. (eds.) Robustness in Language and Speech Technology. Text, Speech and Language Technology, pp. 153–163. Springer, Dordrecht (2001). https://doi.org/10.1007/978-94-015-9719-7_6
18. Eğecioğlu, Ö.: Strongly regular grammars and regular approximation of context-free languages. In: Developments in Language Theory: 13th International Conference, pp. 207–220 (2009)
19. Sakharov, A., Sakharov, T.: The Viterbi algorithm for subsets of stochastic context-free languages. Inf. Process. Lett. **135**, 68–72 (2018)
20. Friedl, J.E.F.: Mastering Regular Expressions. O'Reilly & Associates Inc., Sebastopol (2002)
21. Hromkovič, J., Seibert, S., Wilke, T.: Translating regular expressions into small ϵ-free nondeterministic finite automata. J. Comput. Syst. Sci. **62**(4), 565–588 (2001)
22. Cotter, A., Gupta, M.R., Pfeifer, J.: A light touch for heavily constrained SGD. In: Proceedings of the 29th Conference on Learning Theory, pp. 729–771 (2016)
23. Grathwohl, N.B.B., Henglein, F., Rasmussen, U.T.: Optimally streaming greedy regular expression parsing. In: International Conference on Theoretical Aspects of Computing, pp. 224–240 (2014)

24. Nederhof, M.J.: Context-free parsing through regular approximation. In: Proceedings of the International Workshop on Finite State Methods in Natural Language Processing, pp. 13–24 (1998)
25. Nederhof, M.J.: A general technique to train language models on language models. Comput. Linguist. **31**(2), 173–186 (2005)