Mathematical and Computer Modelling 52 (2010) 490–500

Contents lists available at ScienceDirect

Mathematical and Computer Modelling

journal homepage: www.elsevier.com/locate/mcm

Consistency of stochastic context-free grammars

Roland Gecse [a,*], Attila Kovács [b]

[a] Ericsson Hungary Ltd., P.O. Box 107, 1300 Budapest, Hungary

[b] Eötvös Loránd University, Department of Computer Algebra, Pázmány P. sétány 1/C, 1117 Budapest, Hungary

ARTICLE INFO

Article history:
Received 23 March 2010
Accepted 23 March 2010

Keywords:
Probabilistic grammar
Consistency
Test data generation

ABSTRACT

Computational linguistics play an important role in modeling various applications. Stochastic context-free grammars (SCFGs), for instance, are widely used in compiler testing, natural language processing (NLP), speech recognition and bioinformatics. The commonality of the former projects is that all require consistent SCFGs. This article addresses the consistency problem of SCFGs. We introduce a criterion for deciding the consistency and present a method for turning an inconsistent SCFG into consistent. The feasibility of our theory is demonstrated on random test data generation for some programming languages and formal notations.

1. Introduction

A stochastic (or probabilistic) context-free grammar (SCFG) is a context-free grammar wherein each production is augmented with a probability. Derivation of a sentence in an SCFG can be interpreted as a stochastic string-rewriting process, in which each step consists of replacing a nonterminal in the sentential form with the right-hand side of a production, drawn randomly according to the rule probabilities. The rewriting process satisfies the Markovian assumption, i.e. the probability of rewriting a nonterminal depends only on the nonterminal itself and not on its surrounding context.

Probabilistic grammars have been originally proposed by Salomaa in [1], but the theory being used today is dedicated to Booth and Thompson [2] who revised the definition, introduced the concept of consistency and proved important properties of SCFGs using the theory of branching processes [3]. The most remarkable contribution in [2] is the necessary and sufficient condition for consistency. Recent studies show that SCFGs, as particular multitype branching processes (MTBPs), are closely related to recursive Markov chains [4] and probabilistic pushdown systems [5]. For example, a single-exit recursive Markov chain, where each component in the chain has exactly one exit, is equivalent to an SCFG.

Probabilistic grammars were originally thought for analyzing programming languages [6] but became widely accepted in many research and development areas. SCFGs are used in NLP [7,8], speech recognition [9,10] and bioinformatics [11] applications, mainly to create a simple, generative model representing a given set of sentences (corpus). Production probabilities are learned from parse trees [9,6,12] or directly from sentences [13]. The training algorithms, such as the Inside–Outside method [9,12] that was adapted from the forward–backward algorithm of hidden Markov models as well as the relative (weighted) frequency method [6,13], are known to produce always consistent SCFGs [14,15,13]. An alternative application of SCFGs is to generate random input data for testing [16]. In test data generation, however, there is no corpus for inferring probability parameters. Thus, the rule probabilities need to be assigned *a priori* such that the obtained SCFG shall produce adequate test data. The term adequate implies good error detection capabilities and coverage. The problem in test data generation is that, as opposed to the above-mentioned training methods, this kind of rule probability assignment is unlikely to produce consistent grammars.

* Corresponding author. Tel.: +36 302030881.
E-mail addresses: roland.gecse@ericsson.com (R. Gecse), attila.kovacs@compalg.inf.elte.hu (A. Kovács).

0895-7177/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mcm.2010.03.046

This article proposes a new criterion for checking SCFG consistency and compares it with the criteria in [2,4]. Then a new algorithm is introduced for transforming an inconsistent SCFG to be consistent. The practical applicability of the presented methods is demonstrated on some well-known grammars.

2. Definitions and notation

The reader is expected to be familiar with the fundamentals of formal languages and computer linguistics. The article uses terminology from [17].

Definition 1 (*Context-Free Grammar*)**.** A context-free grammar (CFG) is a quadruple $G = \langle \Sigma, N, R, S\rangle$ where Σ is a finite set of terminal symbols, N is a finite set of nonterminal symbols such that $N \cap \Sigma = \varnothing$, R is a finite subset of rewriting rules in form $A \to \omega$ where $A \in N$ and $\omega \in (N \cup \Sigma)^*$, and $S \in N$ is the distinguished start symbol.

The language $L(G)$ generated by a context-free grammar G is the set of all terminal strings derivable from S.

Definition 2 (*Stochastic Context-Free Grammar*)**.** A stochastic context-free grammar (SCFG) is a tuple $G = \langle \Sigma, N, R, S, P\rangle$, where $\langle \Sigma, N, R, S\rangle$ is a context-free grammar and $P : R \to [0, 1]$ is a probability distribution on the rules.

Definition 3 (*Proper CFG*)**.** A context-free grammar $G = \langle \Sigma, N, R, S\rangle$ is proper when it is

(1) cycle-free, i.e. does not contain productions in form $A \overset{+}{\Rightarrow} A$,
(2) ε-free, i.e. R has no ε-rules or R contains exactly one ε-rule $S \to \varepsilon$ and S does not appear on the right-hand side of any rules in R, and
(3) contains no useless symbols. The symbol X is useless if there is no derivation $S \overset{*}{\Rightarrow} wXy \overset{*}{\Rightarrow} wxy$ where $X \in N \cup \Sigma$ and $x, y, w \in \Sigma^*$.

Definition 4 (*Proper SCFG*)**.** A probabilistic context-free grammar $G = \langle \Sigma, N, R, S, P\rangle$ is proper if the CFG $\langle \Sigma, N, R, S\rangle$ is proper, and for all $N_i \in N$

$$\sum_{\omega \in (\Sigma \cup N)^*} \mathrm{Prob}(N_i \to \omega) = \sum_k p_{i,k} = 1$$

where $p_{i,k}$ is the probability assigned to production k of N_i.

These restrictions ensure that all nonterminals define probability measures over strings i.e. $\mathrm{Prob}(N_i \overset{*}{\Rightarrow} w)$ is a proper distribution over w for all $N_i \in N$ and $w \in \Sigma^*$.

Example 1. Let $G = \langle \Sigma, N, R, S, P\rangle$, $\Sigma = \{+, *, (,), a\}$, and $N = \{E, T, F\}$. Let furthermore the start symbol be $E \in N$ and the rewriting rules R have probability distributions in two different cases as follows.

	Case 1	Case 2
$R_{E,1} : E \to E + T$	$p_{1,1} = \frac{1}{2}$	$p_{1,1} = \frac{3}{5}$
$R_{E,2} : E \to T$	$p_{1,2} = \frac{1}{2}$	$p_{1,2} = \frac{2}{5}$
$R_{T,1} : T \to T * F$	$p_{2,1} = \frac{1}{2}$	$p_{2,1} = \frac{1}{2}$
$R_{T,2} : T \to F$	$p_{2,2} = \frac{1}{2}$	$p_{2,2} = \frac{1}{2}$
$R_{F,1} : F \to (E)$	$p_{3,1} = \frac{1}{2}$	$p_{3,1} = \frac{1}{6}$
$R_{F,2} : F \to a$	$p_{3,2} = \frac{1}{2}$	$p_{3,2} = \frac{5}{6}$.

The SCFGs in both cases are proper. □

Within the scope of this article each CFG and SCFG must always be proper. A recursive SCFG has an inherent problem with respect to the stochastic derivation process, i.e. derivation may or may not terminate having completed a finite number of rewritings [2,18,4,16]. This dilemma is expressed by the consistency property of SCFGs.

Definition 5 (*Consistent SCFG*)**.** SCFG G is consistent[1] if the probabilities assigned to all words derivable from G sum to 1.

[1] In NLP literature the *proper* term is used to denote consistency [7].

Loosely speaking, the SCFG is consistent if and only if the stochastic string-rewriting process terminates after a finite number of steps with probability 1. It is important to note that a non-recursive SCFG is always consistent since its corresponding language consists of finite number of finite length words.

Recall that a grammar is in Chomsky normal form (CNF) if it contains productions only in form $N_i \to N_j N_k$ and $N_i \to T_l$, where $N_i, N_j, N_k \in N$ and $T_l \in \Sigma$. Any CFG or SCFG can be converted into CNF which generates exactly the same language. In both languages the same sentences have the same probability and any parse in the original grammar is reconstructible from any parse in the CNF grammar. Consider a proper stochastic context-free grammar $G = \langle \Sigma, N, R, S, P \rangle$ expressed in CNF. Assign a random variable ξ_i to each nonterminal N_i describing the change in length caused by the applied production during rewriting the nonterminal N_i; ξ_1 shall be assigned to the start symbol S. Note that ξ_i is independent of the position of N_i within the sentential form. $E(\xi_i)$ describes the "expected word length" starting from N_i. It can be observed for the two kinds of productions of G that

$$E\left(\xi_i \mid N_i \to N_j N_k\right) = E\left(\xi_j\right) + E(\xi_k) \quad \text{and} \quad E(\xi_i \mid N_i \to T_j) = 1 \tag{1}$$

since the random variables ξ_i are independent. Denoting $l_i = E(\xi_i)$ the expected word length of the derivation process can be expressed with

$$l_i = \sum_{\forall j} m_{i,j} l_j + v_i \quad \text{where } m_{i,j} = \sum_{k=1}^{r_i} p_{i,k} n_{i,j,k}, \;\; v_i = \sum_{k=1}^{r_i} p_{i,k} t_{i,k} \quad \text{and} \tag{2}$$

r_i: Number of productions with $N_i \in N$ premise,
$p_{i,k}$: Probability assigned to production k of N_i,
$n_{i,j,k}$: Number of occurrence of $N_j \in N$ in production k of N_i,
$t_{i,k}$: Number of terminal symbols appearing in production k of N_i.

Calculating l_i for all i yields a system of linear equations which describes the expected length of words derivable starting from each nonterminal of the stochastic context-free grammar G. Eq. (2) can be written in matrix form

$$l = M \cdot l + v \tag{3}$$

by defining the column vectors $l = [l_i]$, $v = [v_i]$, and the matrix $M = [m_{i,j}]$, where $0 < i, j \leq |N|$.

Definition 6 (*Expectation Matrix of SCFG*)**.** The matrix $M = [m_{i,j}]$ is the expectation matrix of the SCFG.

In other words, the (i, j) element of the stochastic expectation matrix tells how many N_js to expect when rewriting N_i.

Example 2 (*Cont.*)**.** The general form of the expectation matrix for the expression grammar G in Example 1 is

$$\mathbf{M} = \begin{bmatrix} p_{1,1} & p_{1,1} + p_{1,2} & 0 \\ 0 & p_{2,1} & p_{2,1} + p_{2,2} \\ p_{3,1} & 0 & 0 \end{bmatrix} = \begin{bmatrix} p_{1,1} & 1 & 0 \\ 0 & p_{2,1} & 1 \\ p_{3,1} & 0 & 0 \end{bmatrix}.$$

In the two cases Eq. (3) can be written as

$$l^{(1)} = \begin{bmatrix} \frac{1}{2} & 1 & 0 \\ 0 & \frac{1}{2} & 1 \\ \frac{1}{2} & 0 & 0 \end{bmatrix} \cdot l^{(1)} + \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{3}{2} \end{bmatrix}, \quad \text{and} \quad l^{(2)} = \begin{bmatrix} \frac{3}{5} & 1 & 0 \\ 0 & \frac{1}{2} & 1 \\ \frac{1}{6} & 0 & 0 \end{bmatrix} \cdot l^{(2)} + \begin{bmatrix} \frac{3}{5} \\ \frac{1}{2} \\ \frac{7}{6} \end{bmatrix}.$$

The solutions of these linear systems are the expected word length of the derivation processes, namely $l^{(1)} = [-9, -5, -3]^T$ and $l^{(2)} = [59, 23, 11]^T$. □

3. Parameter assignment and consistency

Production probabilities play key role in SCFG consistency. SCFG applications typically

(1) seek the probability of generating a sentence,
(2) try to determine whether a given sentence belongs to the grammar,
(3) search for the most probable parse of a sentence,
(4) derive a set of random sentences, or
(5) are used to generate the given sentences with proper likelihood.

Problems (1)–(4) require properly set production probabilities. Problem (5) itself deals with the assignment of probabilities. The rule probabilities are obtained from a set of parse trees (treebank) or directly from a corpus of the language. Chaudhuri et al. prove in [14] that an SCFG is consistent if production probabilities are learned from samples of the language of the grammar by calculating the relative frequencies of using various productions during derivation of the sentences. Sánchez and Benedí prove in [15] that if production probabilities are learned from Inside–Outside algorithm then the obtained SCFG is always consistent. The relative weighted frequency method introduced in [13] also results consistent SCFGs both for initial parameter assignment and renormalization of inconsistent SCFG rule probabilities. It can thus be concluded that all cited training methods are known to produce always consistent SCFG.

In test data generation (4), however, neither the corpus nor a treebank is available, especially if the language under test is new. Unlike in problem (5), the goal here is to generate valid input[2] including not only common but rare sentences, too. Production probabilities can be set randomly or manually (e.g. by an expert) in order to achieve good fault detection. Without restricting the generality of this article, assume that rule probabilities are assigned such that each applicable production is chosen with equal chance $p_{i,k} = 1/r_k$. This way the SCFG is expected to produce large deviation, which contributes to an increase in both fault detection and stress of the implementation under test. In contrast to the above cited training algorithms, this intuitive parameter assignment does not guarantee that the resulting SCFG gets consistent. On the contrary, the obtained SCFG is often inconsistent.

Booth and Thompson define the notion of consistency and give a necessary and sufficient condition for its validity in [2]. SCFG consistency is usually checked either by definition or using the former criterion. The criterion states that a proper SCFG is consistent if the spectral radius $\rho = \rho(M)$ of the expectation matrix M, i.e. the modulus of the largest eigenvalue of M, is smaller than one. In case of $\rho = 1$ the stochastic derivation process is on the stability boundary and this criterion tells nothing about consistency. Etessami and Yannakakis, however, argue in [4] that a spectral radius of unit magnitude also yields consistent SCFG. This case, which is analogous to critical MTBPs, is unsatisfactory in practical applications since the expected length of the generated sentence can be arbitrary large. Therefore we restrict our investigation to *strongly consistent* SCFGs.

Definition 7 (*Strong Consistency*)**.** SCFG G is strongly consistent if and only if the spectral radius ρ of the expectation matrix of G is strictly smaller than 1.

Example 3 (*Cont.*)**.** The first SCFG in Example 1, for instance, has $\rho = 1.157$ meaning that the expression grammar augmented with equally distributed probabilities is inconsistent. The second SCFG has $\rho = 0.968$, therefore this grammar is (strongly) consistent. □

It can be observed that most grammars of programming languages branch rarely as most nonterminals contain only a couple of productions. The productions refer to relatively few other nonterminals, yielding a sparse expectation matrix. This observation can be extended with the following: the expectation matrix of an SCFG is mostly a random, asymmetric, sparse matrix with always non-negative coefficients. Booth and Thompson prove that the average word length of a language generated by a consistent SCFG is always positive [2]. On the other hand it was noted in [19] that the expected frequency of some words gets negative if the SCFG fails to be consistent. These properties combined with Eq. (3) lead to another consistency condition.

Theorem 1 (*Consistency Criterion*)**.** *A proper SCFG is strongly consistent if and only if there is exactly one positive solution of equation* (3)*.*

Proof. The following elementary proof does not require the theory of branching processes. Consider Eq. (3) as

$$u = A \cdot u + v, \tag{4}$$

where A is the stochastic expectation matrix of a proper SCFG. Clearly, A is non-negative. Moreover, since the SCFG is proper, therefore v is strictly positive. First suppose that the given SCFG is strongly consistent. We have to prove that there is a unique positive solution u of (4). Since $\rho(A) < 1$ therefore $(I - A)^{-1}$ exists and it can be expressed as a convergent series

$$(I - A)^{-1} = I + A + A^2 + \cdots + A^m + \cdots. \tag{5}$$

Hence

$$u = (I - A)^{-1}v = Iv + Av + A^2v + \cdots A^mv + \cdots,$$

which is an infinite sum of positive vectors, so u is unique and positive. Since the right-hand side of (5) is convergent therefore u is also finite. Now suppose that a unique solution u of (4) exists and it is positive. We have to prove that the SCFG is strongly consistent, i.e. $\rho(A) < 1$. If $(I - A)$ is singular then $\rho(A) = 1$ and Eq. (4) has no or infinitely many solutions. This case cannot happen. If $(I - A)$ is regular then $(I - A)^{-1}$ exists, therefore u is unique and finite. On the other hand the sequence

[2] Invalid input for negative test is not considered in this article.

$\langle u, Au, A^2u, A^3u, \ldots, A^ku, \ldots\rangle$ is a strictly decreasing sequence of positive vectors, which means that A^k is convergent as k tends to infinity. Let λ be an eigenvalue of A and x be the corresponding eigenvector. Suppose that there exists an eigenvalue λ of A for which $|\lambda| > 1$. Then the sequence $\langle x, Ax, A^2x, \ldots, A^kx, \ldots\rangle$ is divergent, which means that A^k tends to infinity as $k \to \infty$. This case cannot happen as well. Hence $\lim A^k = 0$ as k tends to infinity, by which $\rho(A) < 1$. □

4. Comparison of methods for checking the consistency

The consistency theorem in [2] and the algorithm in [4] require the spectral radius of the stochastic expectation matrix. The calculation of eigenvalues for non-symmetrical random matrix is a complicated procedure since there is no generic method. Solving eigenvalue problems always depend on the peculiarities of the given problem such as matrix size, sparseness, differences between magnitude of coefficients and so on. Eigenvalue calculation is also known to be highly sensitive to small changes in coefficients, which can lead to significant computational errors. Thus, the solution usually involves balancing and a number of well-chosen similarity transformations until some easy-to-solve matrix form is reached. Fortunately, the calculation of all eigenvalues is not necessary for getting the spectral radius. It is sufficient to ensure that the expectation matrix is contractive. Wetherell [6] and Sarkar [20] rely on the combination of the power method and Gerschgorin's algorithm (see e.g. [21]) in their consistency check implementations. This requires the calculation of matrix powers between successive iterations but, in worst case, tells nothing about the spectral radius, since the convergence of the Gerschgorin test depends highly on the matrix coefficients. Etessami and Yannakakis present a polynomial time algorithm for analyzing the consistency of SCFGs [4]. This algorithm is based on eigenvalue characterizations, including a conversion into Jacobian matrix form, together with graph theoretic techniques.

There are some better, always convergent methods to obtain the spectral radius. Let us denote the number of nonterminals by n. Then the Lehmer–Schur method (see e.g. in [22]), for instance, requires only $O(n^2)$ multiplications to determine if a complex polynomial of degree n has roots greater than 1 in magnitude. The drawback of this method is that it operates on the characteristic polynomial of the expectation matrix. The calculation of characteristic polynomial coefficients from an $n \times n$ expectation matrix itself is $O(n^3)$ complexity. Although, there is a wide variety of direct and iterative methods for determining the spectral radius, their worst-case complexity is at least $O(n^3)$.

Theorem 1 requires the computation of the roots of a system of linear equations. The direct methods, i.e. Gaussian or Gauss–Jordan elimination as well as LU decomposition are all of $O(n^3)$ operation complexity. It seems that all consistency criteria have the same asymptotic worst-case complexity. The operations count can, however, show significant difference due to the neglected constant factors in the O notation.

Let us compare our consistency criterion with those in [2,4], which rely on spectral radius calculation, using the SCFG model of some popular programming languages and formal notations such as ANSI C, ISO C++, C#, OMG Interface Description Language (IDL) and ETSI Test and Test Control Notation version 3 (TTCN-3)[3] by assigning evenly distributed probability parameters. The calculations required for consistency checks were carried out in MATLAB[4] with the results shown in Table 1.

The measurements conducted with sparse matrices show significant advantage of direct elimination methods over both direct and iterative spectral radius calculations. Although the asymptotic worst-case operations complexity is the same, the criterion in Theorem 1 is computationally more effective than the theorem given by Booth and Thompson [2] as the direct algorithms for solving systems of linear equations work significantly better for random, real-valued, asymmetric, sparse matrices than state-of-the-art iterative algorithms for spectral radius calculation.

5. Making an SCFG strongly consistent

Test data generation requires strongly consistent SCFG. If the input grammar is inconsistent then the derivation algorithm is likely to get into infinite loop while trying to generate an infinite length word. This section presents the construction of an algorithm for making an inconsistent SCFG strongly consistent without annihilating any rules of the grammar.

5.1. Categorizing rules and nonterminals

Consistency depends on rule structure and rule probability distribution. Let us investigate how structural grammar properties influence the consistency. If a rule with terminals only is picked during derivation, then the number of nonterminals decreases in that step by one. This kind of production is referred to as *good* rule. When a nonterminal has good rules only then it does not affect consistency regardless of the assigned probabilities. If a self-loop rule of form

$$A \to \omega : \omega \in (N \cup \Sigma)^* A (N \cup \Sigma)^* \quad \text{and} \quad A \in N \tag{6}$$

is chosen then the number of nonterminals cannot decrease. This kind of rule is called *bad* rule. It is important to mention that proper grammars must not contain nonterminals with bad rules only. In other situations, when a rule does neither

[3] The ANSI C grammar is attributed to Jeff Lee, the C++ grammar was obtained from the Ph.D. thesis of E.D. Willink, the C# grammar is a work of James Power. The IDL and TTCN-3 grammars are properties of Ericsson Hungary Ltd.

[4] MATLAB is the registered trademark of The MathWorks, Inc.

Table 1
Comparison of consistency criteria in MATLAB—spectral radius calculation vs. solving a system of linear equations. n represents the number of nonterminals of the grammar, which determines matrix size. Spectral radius is obtained in two different ways. The first method uses the `eig` function for direct eigenvalue calculation. The second method relies on the random iterative `eigs` function to obtain a spectral radius estimation. The latter method is also combined with sparse data representation. The system of linear equations is solved with Gaussian elimination. The dense and sparse columns contain calculations carried out in dense and sparse matrix data representations. The measurement results are given in *kflops*, as measured using the `flops` function of MATLAB. The displayed figures contain the mean of 100 measurements to avoid the variance of iterative methods.

	n	Spectral radius calculation			Gaussian elim.	
		`eig`	`eigs`	`eigs sparse`	Dense	Sparse
ANSI C	63	1 810	2 714	2096	199	5
IDL	126	2 703	3 562	2487	1 462	18
ANSI C++	211	43 604	10 783	4980	6 625	51
C#	235	43 289	14 287	6159	9 103	63
TTCN-3	463	253 883	19 975	4069	67 911	232

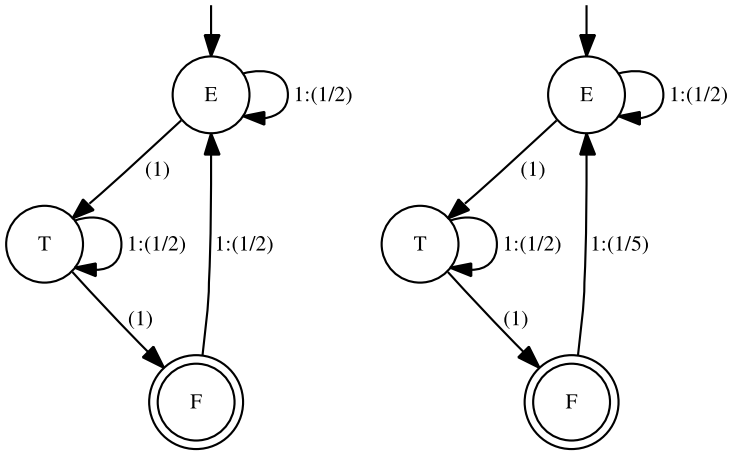

Fig. 1. Directed graph representation of the grammar in Example 1 with Case 1 probabilities (left) and with (strongly) consistent probabilities (right). The grammar was made strongly consistent by increasing the probability of the good rule of F, which is the only α-symbol of the grammar.

contain a self-loop (6) nor all symbols in its consequence are terminals, anything can happen. This kind of rule is named *neutral* rule. Each rule of the grammar is thus either a good, bad or neutral rule.

Nonterminals can also be categorized depending on their rules. Nonterminals with at least one good rule are called distinctively α-symbols. The rest of the nonterminals, i.e. those without any good rule, are called β-symbols.

5.2. Graphical representation of SCFG

SCFGs can be visualized using directed graphs, which not only show the expectations but also refer to the applicable rules, making the representation suitable for analyzing consistency.

Definition 8 (*Directed Graph Representation of SCFG*)**.** The nodes correspond to the nonterminals, α-symbols are distinguished from β-symbols with double circles, the start symbol is also marked. The edges represent rules, such that an (A, B) directed edge is present if A has some rules with B in their consequence: $R_{A,i} : A \to \omega$ and $\omega \in (N \cup \Sigma)^* B (N \cup \Sigma)^*$. The (A, B) edges are labeled with the sequence of i indices of the $R_{A,i}$ rules. Labels can be omitted when a nonterminal appears in the consequence of all of its rules. The entries of the expectation matrix appear on the edges in parentheses.

Example 4 (*Cont.*)**.** Fig. 1 presents the directed graph representations of the SCFG of Example 1. Expectations are shown in parentheses. □

5.3. Decomposing the problem

The directed graph representation may contain strongly connected components (SCCs). An important property of SCCs is that once the derivation exits a component it can never return. Decomposing the original SCFG into subgrammars corresponding to its SCCs leads to a simplification of the problem.

Definition 9 (*Subgrammar*)**.** Consider SCFGs $G = \langle \Sigma_G, N_G, R_G, S_G, P_G \rangle$ and $F = \langle \Sigma_F, N_F, R_F, S_F, P_F \rangle$. Let $\tau \notin (\Sigma_G \cup N_G)$. F is a subgrammar of G if $\Sigma_F \subseteq \Sigma_G \cup \{\tau\}$, $N_F \subseteq N_G$, $R_F \subseteq R_G$ such that $R_G : A \to \psi B \omega$ gets $R_F : A \to \psi \tau \omega$ if $A \in N_F$, $B \in N_G \setminus N_F$, $\psi, \omega \in (\Sigma_F \cup N_F)^*$, $S_F \in N_F$ and $P_F \subseteq P_G$.

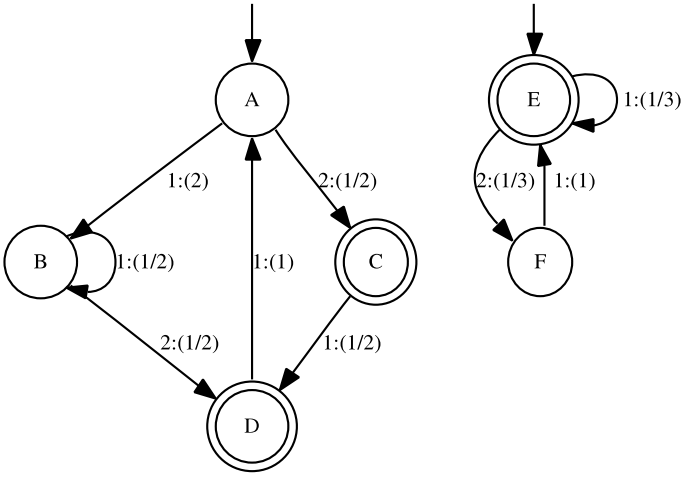

Fig. 2. Strongly connected components of the SCFG in Example 5.

Consider the SCCs of the directed graph representation of the SCFG. The subgrammar contains all nonterminals and productions belonging to the respective SCC and the terminals occurring in the inherited rules. Those nonterminals in the rules of the original SCFG, which are absent from the current SCC, are replaced with a single special terminal symbol τ. The start symbol of the subgrammar can be any nonterminal, which acts as an entry point of the SCC.

Example 5. The following artificial SCFG contains two SCCs components, which are shown in Fig. 2. The component to the left is inconsistent, while the one on the right is strongly consistent.

$$
\begin{array}{llll}
R_{A,1} : A \to BBBBE & p_{1,1} = \frac{1}{2} & R_{E,1} : E \to Ea & p_{5,1} = \frac{1}{3} \\
R_{A,2} : A \to aC & p_{1,2} = \frac{1}{2} & R_{E,2} : E \to aF & p_{5,2} = \frac{1}{3} \\
R_{B,1} : B \to Ba & p_{2,1} = \frac{1}{2} & R_{E,3} : E \to a & p_{5,3} = \frac{1}{3} \\
R_{B,2} : B \to aD & p_{2,2} = \frac{1}{2} & R_{F,1} : F \to Ea & p_{6,1} = 1 \\
R_{C,1} : C \to D & p_{3,1} = \frac{1}{2} & & \\
R_{C,2} : C \to a & p_{3,2} = \frac{1}{2} & & \\
R_{D,1} : D \to AA & p_{4,1} = \frac{1}{2} & & \\
R_{D,2} : D \to Ea & p_{4,2} = \frac{1}{2}. & &
\end{array}
$$

The subgrammar corresponding to the left component contains symbols A, B, C, D and their rules. Productions $A \to BBBBE$ and $D \to Ea$ refer to nonterminal E, which is outside the SCC, hence it is replaced in both rules with the special terminal symbol τ. The relevant rules in the subgrammar are changed to $A \to BBBB\tau$ and $D \to \tau a$. The subgrammar of the other component contains symbols E, F and their productions. There is no change in the rules because these refer to nonterminals within the SCC only. □

The SCCs of subgrammars are independent. Accordingly, the consistency of the original SCFG depends on the consistency of its subgrammars.

Lemma 1. *An SCFG is consistent if and only if all subgrammars corresponding to SCCs of the grammar are consistent.*

Proof. A real $n \times n$ matrix M with non-negative entries (such as an expectation matrix) is said to be irreducible if every element, labeled as a row–column pair (i, j), is greater than zero in some finite power of M. It means that in this case for every pair (i, j) there is a positive k such that $(M^k)_{ij} > 0$. The adjacency matrix A of a strongly connected graph component is irreducible since $(A^k)_{ij}$ is exactly the number of paths from i to j of length k. If a graph is not strongly connected then it is reducible. Perron–Frobenius theorem [23] tells us that the set of eigenvalues of a reducible matrix is the union of the

eigenvalues of its irreducible subgraphs. In order to see why, let us consider the reducible matrix

$$\mathbf{M} = \begin{bmatrix} C_{11} & D_{12} & \cdots & D_{1n} \\ 0 & C_{22} & \cdots & D_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & C_{nn} \end{bmatrix},$$

where C_{ii} is the adjacency sub-matrix of the SCC i. The off-diagonal elements D_{ij} represent the inter-SCC links. Using topological ordering every reducible matrix can be arranged in this way. The eigenvalues of M have a simple structure, namely $\det(M - \lambda I) = \det(C_{11} - \lambda I) \cdot \det(C_{22} - \lambda I) \cdots \det(C_{nn} - \lambda I)$. Thus, the eigenvalues of M are the union of the eigenvalues belonging to each of the irreducible components. This means that the connections between SCCs do not alter the eigenvalue spectrum. □

5.4. Elementary procedures of making the SCFG consistent

The easiest way to turn a subgrammar strongly consistent is to increase the probabilities of rules leading out from the SCC. The α-symbols always have at least one good rule by definition. The α-symbols can therefore be easily eliminated during derivation. Indeed, some SCFG can be made strongly consistent by increasing probabilities of these productions only.

Example 6. Inconsistency of the SCFG of Example 1 with Case 1 probabilities (Fig. 1 left) is caused by the dominance of the $F \to E$ feedback. Symbol F, which is the only α-symbol of the grammar, has a single good rule $F \to a$. Increasing the $p_{3,2}$ probability to 4/5 decreases the expectation on the (F, E) edge – and thereby the positive feedback caused by the $F \to (E)$ production – and the grammar gets strongly consistent (Fig. 1 right). □

Nevertheless, not only the α-symbols but all nonterminals of the grammar influence consistency. Therefore, the previous approach is not always sufficient to make an SCFG consistent. The solution is to involve the β-symbols in the process and *force* the derivation toward α-symbols. The easiest way of doing this is to prefer a single rule during derivation at each β-symbol. The preferred or best rule among the neutral rules of a β-symbol is the rule that requires the least number of rewriting steps to reach a sentential form consisting of α-symbols and terminals only. Observe that the best rule is not necessarily unique.

Definition 10 (*Hop Count*)**.** The hop count of some nonterminal A is the smallest number of rewriting steps required to reach a sentential form containing of α-symbols and terminals only. Formally:

$$\mathrm{hop}(A) = \min(k : A \overset{k}{\Rightarrow} \omega) \quad \text{such that } \omega \in (\alpha\text{-symbol} \cup \Sigma)^* \text{ and } A \in N.$$

The best rules of β-symbols can be found using Algorithm 1 employing a bottom–up technique involving the hop count of nonterminals. X_0 is initialized with the α-symbols of the grammar, which always have zero hop count. Next, the self-loop rules have to be removed because a bad rule can never be the best. The β-symbols are then taken one by one, such that all nonterminals within the rules of the chosen symbol (B) can be found in the solution set (this can always be done for proper SCFG). Calculate the sum of hop counts of all nonterminals for each rule of B. The best rule is the one with the lowest hop count sum. The hop count of B is one more than the hop count sum of its best rule. The best rule is marked, the solution set is expanded with B and the procedure continues until the best rule is found for all β-symbols.

The hop count can be obtained from the directed graph representation, too. In each hop one needs to choose a rule for rewriting the current nonterminal. When the selected rule contains more than one nonterminals then the graph branches and derivation has to follow all edges of the corresponding rule concurrently. The traversal finishes when all branches reach an α-symbol of the SCC. The hop count will be the number of edges traversed.

Example 7. Let us find the best rules of β-symbols for the first subgrammar of Example 5 as shown in Fig. 2. The β-symbols are A and B. The first rule of A refers to B, thus the best rule of B needs to be determined first. B has two rules. The first contains self-loop therefore the second rule must be preferred. The second rule $B \to aD$ refers to an α-symbol, thus the hop count of B will be 1. Proceeding with A, the first rule $A \to BBBB\tau$ introduces four Bs, implying hop count 5 for A. The second rule $A \to aC$, however, leads directly to an α-symbol, which means that the hop count of A is 1. The best rules of the two β-symbols are thus $B \to aD$ and $A \to aC$. □

5.5. The algorithm

This section presents an algorithm (Algorithm 2) for turning a proper SCFG strongly consistent. Our method first simplifies the problem by decomposing the input SCFG into subgrammars corresponding to its SCCs. The obtained subgrammars are processed one by one. When a subgrammar is not strongly consistent then Algorithm 2 first determines its α and β-symbols. Then, it marks all good rules of α-symbols and the best rules of each β-symbol using Algorithm 1. The principle of the

Algorithm 1 Finding the best rule of each β-symbol in a subgrammar.

function FINDBESTRULES(G) ▷ G is a proper SCFG
  $i \leftarrow 0$
  $M_i \leftarrow \varnothing$ ▷ The set of marked best rules
  $X_i \leftarrow \{\alpha\text{-symbols}\}$ ▷ Initialize the solution set
  $R^* \leftarrow R \setminus \{\text{self-loop rules}\}$ ▷ Removing self-loops
  repeat
    $i \leftarrow i + 1$
    $B \leftarrow$ next β-symbol such that $\forall j$ RHS of $R^*_{B,j} \in (X_{i-1} \cup \Sigma)^*$
    for all rule $R^*_{B,j}$ **do**
      $h_{B,j} \leftarrow \sum(\text{nonterminal hop counts in RHS of } R^*_{B,j})$
    end for
    $M_i \leftarrow M_{i-1} \cup \{R^*_{B,j} : h_{B,j} = \min(h_{B,j})\}$ ▷ Mark best rule of B
    $\text{hop}(B) \leftarrow \min(h_{B,j}) + 1$ ▷ Save hop count
    $X_i \leftarrow X_{i-1} \cup \{B\}$ ▷ Update the solution set
  until $X_i = X_{i-1}$ ▷ i.e. all β-symbols are processed
  return M_i
end function

algorithm relies on altering production probabilities such that the marked rules are taken more likely during derivation. It increases the probabilities of these rules using a set *SF* of strictly increasing amending functions. Then, normalization makes sure that the sum of all production probabilities remains 1 after applying the amending functions. It is important to note the greediness of the algorithm, which is manifested in the fact that the probabilities of all marked rules are increased *simultaneously*. This happens because it cannot be determined which productions are directly responsible for inconsistency. These undesired rules are, however, surely among the unmarked productions. Increasing the probabilities of marked rules will thus decrease the probabilities of the remaining neutral and bad rules. The execution takes as many iterations as necessary to make the subgrammar strongly consistent. The number of iterations as well as the convergence of the process depends on the employed amending functions. Theoretically, arbitrary strictly increasing, from above unbounded function can be used for this purpose. Exaggeratedly, one could select different amending functions for each rule as well. The practical choice depends on the peculiarities of the given problem. When the aim is, for instance, to find a solution swiftly then the employed set of *SF* functions should increase aggressively. If the expectation of the lengths of generated sentences should be large then it is recommended to pick slowly increasing amending functions instead. Experiments show that adequate results can be achieved by systematically applying the function $f(x) = 2x$ and then normalizing the probabilities. The altered probabilities of the strongly consistent subgrammar need to be written back into the original SCFG. When the algorithm finishes processing all subgrammars, the original SCFG will be strongly consistent.

Algorithm 2 Turning the grammar strongly consistent.

procedure MAKESTRONGLYCONSISTENT(G, SF)
  Detect SCCs
  for all SCC **do**
    Create subgrammar
    Check consistency
    if not strongly consistent **then**
      Categorize rules and symbols
      Mark all good rules of α-symbols
      Find and mark the best rule of β-symbols with Algorithm 1
      repeat
        Increase the probabilities of marked rules using the set *SF*
        Normalize rule probabilities
      until subgrammar gets strongly consistent
      Write back rule probabilities into the original grammar
    end if
  end for
  Output is a strongly consistent grammar
end procedure

Example 8. Applying Algorithm 2 to the grammar of Example 5 first detects the SCCs in Fig. 2 and spawns the respective subgrammars. The right component is strongly consistent thus probabilities of its subgrammar remain unchanged. The

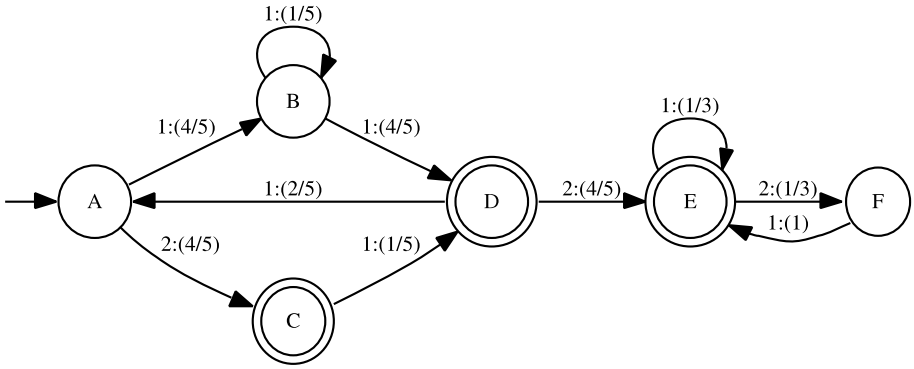

Fig. 3. Sample grammar of Example 4 with strongly consistent weights.

Table 2
Results of Algorithm 2 on the grammars of Table 1; the SCC column contains the SCCs of the grammar. Column N shows the number of nonterminals within the component. The next column notes whether the original subgrammar is strongly consistent or not. The last column holds the number of iteration steps taken in Algorithm 2 using systematically the function $f(x) = 2x$ for making the subgrammar strongly consistent.

Grammar	SCC	N	Strongly consistent	Steps
ANSI C	SCC-1	38	No	4
	SCC-2	2	No	1
	SCC-3	6	Yes	0
IDL	SCC-1	9	No	2
	SCC-2	5	Yes	0
	SCC-3	9	Yes	0
	SCC-4	3	Yes	0
ANSI C++	SCC-1	168	No	4
C#	SCC-1	5	Yes	0
	SCC-2	45	No	2
	SCC-3	28	Yes	0
	SCC-4	11	Yes	0
	SCC-5	5	Yes	0
TTCN-3	SCC-1	96	No	5
	SCC-2	12	Yes	0
	SCC-3	31	Yes	0
	SCC-4	3	Yes	0

component to the left is, however, inconsistent. While categorizing its nonterminals, C and D turn out to be α-symbols. Their good rules are $C \to a$ and $D \to \tau a$. Nonterminals A and B are β-symbols. Their best rules have been determined in Example 7. Multiplying these rules' probabilities using the suggested function f leaves the SCC inconsistent. It takes another iteration to make the subgrammar strongly consistent. Writing back the rule probabilities results the strongly consistent grammar in Fig. 3. □

Table 2 summarizes the result of measurements performed with Algorithm 2 on the grammars in Table 1.

Theorem 2 (*Turning SCFGs Strongly Consistent*)**.** *Every proper, inconsistent SCFG can be transformed into a strongly consistent one.*

Proof. Constructive. Algorithm 2 gradually increases the probability of those productions that do not contribute to recursions. Therefore the probability of engaging into a recursion loop decreases provided there is at least one production that leads out from each recursion. Since the CFG part of the observed SCFG is assumed to be proper, it satisfies this condition thus the introduced method works for all proper SCFGs. □

5.6. Performance considerations

The execution performance of Algorithm 2 can be estimated based on the computational complexity of its elementary operations. Detecting the SCCs of $G = (V, E)$ graph requires linear time $\Theta(V + E)$ if the graph is represented as an adjacency list. In case of the directed graph representation of SCFG the performance is $\Theta(|N| + |R|)$. Creating a subgrammar of an SCFG happens in linear time as well. The consistency check algorithm takes $O(|N|^3)$ execution time as discussed in Section 4. It is important to mention that consistency check runs every time on different $\mathbf{M}$ since some production probabilities are altered in each iteration step. The (re)construction of the expectation matrix takes $O(|N|^3)$ operations. Finding the best rules of

β-symbols is linear if the β-symbols can be taken in reverse breadth-first search order, which takes $O(|N| + |R|)$ computations. Hence, the execution performance of Algorithm 2 depends mainly on the set *SF*. The invocation of the expensive consistency check algorithm – and the rest of linear time operations within the body of the loop – can be kept low when the likelihood of marked rules is increased as suggested in Section 5.5. Then the procedure (i.e. turning the SCFG strongly consistent) converges fast as confirmed by our experiments (see results in Table 2).

6. Summary

We presented a new method for determining consistency, which is slightly better in both execution performance and accuracy than the consistency criterion of Booth and Thompson. The efficiency is based on the fact that solving a system of sparse linear equations with non-negative coefficients is faster than calculating the eigenvalue of highest magnitude.

We proved that a proper, inconsistent SCFG can always be transformed into a strongly consistent one just by altering some production probabilities. The execution time of our algorithm makes practical problems tractable. Potential applications include stochastic test data generation for automated testing.

Our results can also be applied to testing protocols with simple behavior but complicated syntax where data type hierarchy is modeled with SCFG and rule probabilities are assigned a priori. If the initial grammar is inconsistent then it needs to be made strongly consistent with the introduced algorithm. The obtained strongly consistent SCFG is used to generate syntactically correct type skeletons of messages. The type skeletons are then filled with values according to predefined constraints resulting in semantically valid messages. Remaining details are subject to further study.

Acknowledgement

The research of the second author was supported by TÁMOP-4.2.1/B-09/1/KMR-2010-003.

References

[1] A. Salomaa, Probabilistic and weighted grammars, Information and Control 15 (1969) 529–544.
[2] T.L. Booth, R.A. Thompson, Applying probability measures to abstract languages, IEEE Transactions on Computers C-22 (5) (1973) 442–450.
[3] T.E. Harris, The Theory of Branching Processes, Springer, 1963.
[4] K. Etessami, M. Yannakakis, Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations, in: Proc. of 22nd STACS'05, in: LNCS, vol. 3404, Springer, 2005, pp. 340–352.
[5] J. Esparza, A. Kučera, R. Mayr, Model checking probabilistic pushdown automata, in: Proc. of 19th IEEE LICS'04, 2004.
[6] C.S. Wetherell, Probabilistic languages: a review and some open questions, ACM Computing Surveys 12 (4) (1980) 361–379.
[7] C.D. Manning, H. Schütze, Foundations of Statistical Natural Language Processing, MIT Press, Cambridge, MA, 1999.
[8] D. Jurafsky, J.H. Martin, Speech and Language Processing, Prentice Hall, 2000.
[9] J. Baker, Trainable grammars for speech recognition, in: Speech Communication Papers for the 97 Meeting of Acoustical Society of America, 1979, pp. 547–550.
[10] F. Jelinek, J.D. Lafferty, R.L. Mercer, Basic methods of probabilistic context-free grammars, in: P. Laface, R.D. Mori (Eds.), Speech Recognition and Understanding: Recent Advances, Trends, and Applications, in: NATO Science Series F: Computer and Systems Sciences, vol. 75, Springer, 1992, pp. 345–360.
[11] B. Knudsen, J. Hein, RNA secondary structure prediction using stochastic context-free grammars and evolutionary history, Bioinformatics 15 (6) (1999) 446–454.
[12] K. Lari, S.J. Young, Applications of stochastic context-free grammars using the inside–outside algorithm, Computer Speech and Language 5 (1991) 237–257.
[13] Z. Chi, Statistical properties of probabilistic context-free grammars, Computational Linguistics 25 (1) (1999) 131–160.
[14] R. Chaudhuri, S. Pham, O.N. Garcia, Solution of an open problem on probabilistic grammars, IEEE Transactions on Computers C-32 (8) (1983) 748–750.
[15] J.-A. Sánchez, J.-M. Benedí, Consistency of stochastic context-free grammars from probabilistic estimation based on growth transformations, IEEE Transactions on Pattern Analysis and Machine Intelligence 19 (9) (1997) 1052–1055.
[16] P.M. Maurer, Generating test data with enhanced context-free grammars, IEEE Software 7 (4) (1990) 50–55.
[17] A.V. Aho, J.D. Ullman, The Theory of Parsing, Translation, and Compiling, in: Parsing of Series in Automatic Computation, vol. I, Prentice Hall, Englewood Cliffs, New Jersey, 1972.
[18] N.R. Hansen, A note on stochastic context-free grammars, termination and the EM-algorithm, Preprint, 2005.
[19] A. Stolcke, J. Segal, Precise *n*-gram probabilities from stochastic context-free grammars, in: ACL, 1994, pp. 74–79.
[20] A. Sarkar, Conditions on consistency of probabilistic tree adjoining grammars, in: COLING-ACL, 1998, pp. 1164–1170.
[21] A. Ralston, A First Course in Numerical Analysis, McGraw-Hill, 1965.
[22] F.S. Acton, Numerical Methods that Usually Work, Mathematical Association of America, Washington, DC, USA, 1990, Updated and revised from the 1970 Edition.
[23] R.S. Varga, Matrix Iterative Analysis, Prentice Hall, Englewood Cliffs, New Jersey, 1962.