Random Graph Generation in Context-Free Graph Languages

Federico Vastarini
University of York
York, UK
federico.vastarini@york.ac.uk

Detlef Plump
University of York
York, UK
detlef.plump@york.ac.uk

We present a method for generating random hypergraphs in context-free hypergraph languages. It is obtained by adapting Mairson's generation algorithm for context-free string grammars to the setting of hyperedge replacement grammars. Our main results are that for non-ambiguous hyperedge replacement grammars, the method generates hypergraphs uniformly at random and in quadratic time. We illustrate our approach by a running example of a hyperedge replacement grammar generating term graphs.

1 Introduction

We present a novel approach to the generation of random hypergraphs in user-specified domains. Our approach extends a method of Mairson for generating strings in context-free languages [11] to the setting of context-free hypergraph languages specified by hyperedge replacement grammars. Generating (or "sampling") graphs and hypergraphs according to a given probability distributions is a problem that finds application in testing algorithms and programs working on graphs. Molecular biology and cryptography are two fields of potential application where our methods could find a concrete use besides the mere software testing. In [9] Kajino presents a novel approach for the representation of molecules through hypergraphs. Specifically adapting our method to this setting would provide an instrument for the exploration of new compounds in the field of molecular biology. The uniformity of the distribution of our method is a fundamental requirement for the development of cryptographic protocols. In [6] and [12] we may find some useful insights on how to model graph based algorithms in that domain. In the setting of hyperedge replacement grammars, we believe that there is an opportunity for the development of one-way functions.

Our generation algorithm uses as input a hyperedge replacement grammar in Chomsky normal form [2] and a positive integer n. The former specifies the hypergraph language to sample from, the latter the size of the hypergraph to be generated. The algorithm then chooses a hypergraph at random from the slice of the language consisting of all members of size n. We show that if the grammar is non-ambiguous, the generated samples are uniformly distributed. The only requirements for our method are that the properties sought for the generated hypergraphs are representable by a hyperedge replacement language and that, to guarantee a uniform distribution, a non-ambiguous grammar is used as input.

We also show that our method generates a random hypergraph of size n in time $O(n^2)$. This is the same time bound established by Mairson (for the first method) in the setting of random string generation in context-free languages.

S. Alves and I. Mackie (Eds.): 13th International Workshop
on Developments in Computational Models 2023 (DCM'23).
EPTCS ??, 2024, pp. 87–100, doi:10.4204/EPTCS.??.7

© Federico Vastarini, Detlef Plump
This work is licensed under the
Creative Commons Attribution License.

2 Hyperedge Replacement Grammars

This section gives a concise overview of the definitions needed to understand the generation process. We also introduce our running example of a language of term graphs specified by hyperedge replacement. For comprehensive treatments of the theory of hyperedge replacement grammars and languages, we refer to Courcelle [3], Drewes et al. [4] and Engelfriet [5].

Let $type\colon C \to \mathbb{N}_0$ be a typing function for a fixed set of labels C, then a *hypergraph* over C is a tuple $H = (V_H, E_H, att_H, lab_H, ext_H)$ where V_H is a finite set of *vertices*, E_H is a finite set of *hyperedges*, $att_H\colon E_H \to V_H^*$ is a mapping assigning a sequence of *attachment nodes* to each $e \in E_H$, $lab_H\colon E_H \to C$ is a function that maps each hyperedge to a *label* such that $type(lab_H(e)) = |att_H(e)|$, $ext_H \in V_H^*$ is a sequence of pairwise distinct *external nodes* (Figure 1). The class of all hypergraphs over C is denoted by $\mathscr{H}_C$.

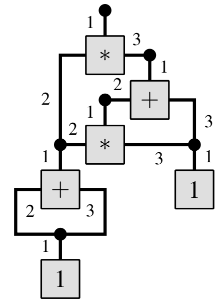


Figure 1: A term graph

We write $type(H)$ for $|ext_H|$ and call H an n-hypergraph if $type(H) = n$. The length of the sequence of *attachments* $|att_H(e)|$ is the *type* of e. Hyperedge e is an m-hyperedge if $type(lab(e)) = m$. We also write $type(e) = m$ if the context is clear. If an n-hypergraph has exactly 1 hyperedge and all its nodes are external, that is $E_H = \{e\}$ and $|V_H| = n$, it is called the *handle* induced by e and denoted by I_e. Moreover if $type(e) = n$ and $ext_H = att(e)$ such a hypergraph is called the handle induced by A and denoted by $A^\bullet$. We write $att_H(e)_i$ for the ith attachment node of $e \in E_H$ and $ext_{H,i}$ for the ith external node of H. The set $E_H^X = \{e \in E \mid lab_H(e) \in X\}$ is the subset of E_H with labels in $X \subseteq C$. We define $|H| = |V_H| + |E_H|$ as the *size* of H and we call H a size-n-hypergraph if $|H| = n$.

Figure 1 shows a *term graph*, a form of acyclic hypergraphs that represent functional expressions with possibly shared subexpressions. (See [13] for an introduction to the area of term graph rewriting.) Grey boxes represent hyperedges labelled with the function symbols $*$, $+$ and 1, while nodes are drawn as black bullets. Lines connect hyperedges with their attachment nodes, whose position in the attachment sequence is given by small numbers.

Two hypergraphs $H, H' \in \mathscr{H}_C$ are *isomorphic*, denoted $H \cong H'$, if there are bijective mappings $h_V\colon V_H \to V_{H'}$ and $h_E\colon E_H \to E_{H'}$ such that $h_V^*(att_H(e)) = att_{H'}(h_E(e))$ and $lab_H(e) = lab_{H'}(h_E(e))$ for each $e \in E_H$ and $h_V^*(ext_H) = ext_{H'}$. Two isomorphic hypergraphs are considered to be the same. A hypergraph H is a subgraph of H', denoted as $H \subseteq H'$ if $V_H \subseteq V_{H'}$ and $E_H \subseteq E_{H'}$.

Hypergraphs are generated by replacement operations. Let $H \in \mathscr{H}_C$ and $B \subseteq E_H$ and let $repl\colon B \to \mathscr{H}_C$ be a mapping with $type(repl(e)) = type(e)$ for each $e \in B$. Then the *replacement* of the hyperedges in B with respect to $repl(e)$ is defined by the operations: remove the subset B of hyperedges from E_H; for each $e \in B$, disjointly add the vertices and the hyperedges of $repl(e)$; for each $e \in B$ and $1 \leq i \leq type(e)$, fuse the ith external node $ext_{repl(e),i}$ with the ith attachment node $att_B(e)_i$.

We denote the resulting hypergraph by $H[e_1/R_1,\dots,e_n/R_n]$, where $B=\{e_1,\dots,e_n\}$ and $repl(e_i)=R_i$ for $1\le i\le n$, or $H[repl]$. The replacement preserves the external nodes, thus $ext_{H[repl]}=ext_H$.

Given the subsets $\Sigma,N\subseteq C$ used as terminal and non-terminal labels, with $\Sigma\cap N=\emptyset$, we denote E_H^{Σ} and E_H^{N} respectively the subsets of terminal and non-terminal hyperedges of H.

The replacements applied during the generation are defined in productions: $p=(A,R)$ is a *production* over N, where $lhs(p)=A\in N$ is the label of the replaced hyperedge and $rhs(p)=R\in\mathscr{H}_C$ is a hypergraph with $type(R)=type(A)$. If $|ext_R|=|V_R|$ and $E_R=\emptyset$, then p is said to be *empty*.

Let $H\in\mathscr{H}_C$ and let $p=(lab(e),R)$, with $e\in E_H$, then a *direct derivation* $H\Rightarrow_p H'$ is obtained by the replacement $H'=H[e/R]$.

A sequence d of direct derivations $H_0\Rightarrow_{p_1}\cdots\Rightarrow_{p_k}H_k$ of length k with $(p_1,\dots,p_k)\in P$ is denoted as $H\Rightarrow^k H_k$ or $H\Rightarrow_P^* H_k$ if the length is not relevant. We denote it as $H\Rightarrow^* H_k$ if the sequence is clear from the context.

A derivation $H\Rightarrow^* H'$ of length 0 is given if $H\cong H'$.

Given an ordered set $\{\alpha_1,\dots,\alpha_n\}$ where $a_i<a_j$ if $i<j\in\mathbb{N}$ we define a *hyperedge replacement grammar*, or *HRG* as a tuple $G=(N,\Sigma,P,S,(mark_p)_{p\in P})$ where $N\subseteq C$ is a finite set of non-terminal labels, $\Sigma\subseteq C$ is a finite set of terminal labels with $N\cap\Sigma=\emptyset$, P is a finite set of productions, $S\in N$ is the starting symbol, $(mark_p)_{p\in P}$ is a family of functions $mark_p: E_R\to\{\alpha_1,\dots,\alpha_n\}$ assigning a mark to each hyperedge in the right-hand side of a production p (Figure 2). For each pair $e_i,e_j\in E_R$ with $i\neq j$, $mark(e_i)\neq mark(e_j)$.

We denote as $P^A\subseteq P$ the subset of productions where $lhs(p)=A$. We call a production $p=(A,R)\in P_N\subseteq P$ *non-terminal* if $E_R^N\neq\emptyset$ or *terminal* if $p=(A,R)\in P_\Sigma=P\backslash P_N$.

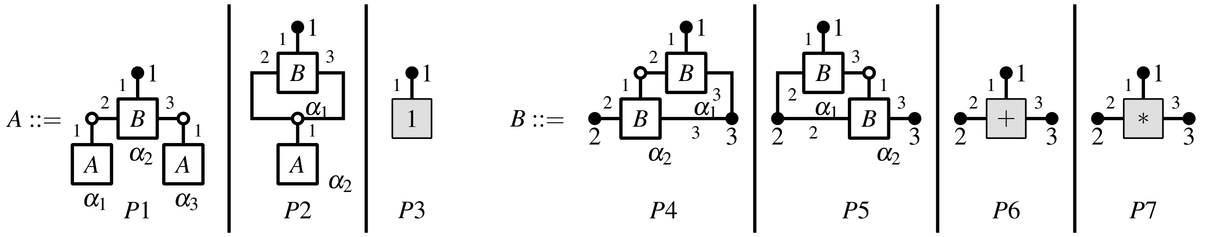


Figure 2: An ambiguous hyperedge replacement grammar for term graphs

The marking of the hyperedges in the *rhs* of each production, represents the order in which the replacements are carried out $(\alpha_1,\alpha_2,\dots,\alpha_{n-1},\alpha_n)$. It allows for the definitions of ordered derivation tree and leftmost derivation.

Given a set of productions P, we denote by T_P the set of all ordered trees over P which is inductively defined as follows: for each $p\in P$, $p\in T_P$; for $t_1,\dots,t_n\in T_P$ and $p\in P$, $p(t_1,\dots,t_n)\in T_P$.

Then, given an *HRG G*, an *ordered derivation tree* t for e such that $lab(e)=X\in N$, is a tree $p(t_{\alpha_1},\dots,t_{\alpha_n})$ in T_P, such that $p=(X,R)$ is a production in P, and $t_{\alpha_1},\dots,t_{\alpha_n}$ are derivation trees for $e_1\dots e_n$, such that $X_1\dots X_n$ are the labels of the non-terminal hyperedges in R marked with $\alpha_1\dots\alpha_n$, respectively (Figure 11).

We define the *yield* of an ordered derivation tree t, denoted with $yield(t)$, as the sequence of replacements: $yield(p(t_{\alpha_1},\dots,t_{\alpha_n}))=rhs(p)[e_1/yield(t_{\alpha_1}),\dots,e_n/yield(t_{\alpha_n})]$.

Let t be an ordered derivation tree for a hypergraph H obtained from a derivation $d=S^\bullet\Rightarrow_P^* H$ and $trav(t)$ its pre-ordered visit. Then d is said to be a *leftmost derivation*, denoted as $lmd(H)$, if and only if the order of the applied productions of d corresponds to $trav(t)$.

Since we need a measure to ensure the termination of the proposed algorithm, we define an *HRG* to be *non-contracting* if for each direct derivation $H \Rightarrow_p H'$, $|H| \leq |H'|$. We call a grammar *essentially non-contracting* if there exists $p = (S,R) \in P$ such that p is the empty production.

The *hyperedge replacement language* (*HRL*) generated by an *HRG* is the set $L(G) = \{H \in \mathscr{H}_\Sigma \mid S^\bullet \Rightarrow_P^* H\}$. We define for each $A \in N$, $L^A(G) = \{H \in \mathscr{H}_\Sigma \mid A^\bullet \Rightarrow_P^* H\}$. We also define for $n \in \mathbb{N}$, $L_n^A(G) = \{H \in \mathscr{H}_\Sigma \mid A^\bullet \Rightarrow_P^* H \wedge |H| = n\}$. Clearly $L_n^A(G) \subseteq L^A(G)$. We denote as $|L_n^A|$ the size of the set of all size-n-hypergraphs in L that can be derived from $A^\bullet$.

For example, the hyperedge replacement grammar in Figure 2 generates the class of all term graphs with function symbols in $\{*,+,1\}$. Note that hyperedges with non-terminal labels are depicted as white boxes. A derivation with the Chomsky normal form version of this grammar is given in Figure 10.

We define a grammar G to be *ambiguous* if there are ordered derivation trees $t_1, t_2 \in T_P$, such that $t_1 \neq t_2$ and $yield(t_1) \cong yield(t_2)$, or equivalently, if there exist $H, H' \in L(G)$ such that $H \cong H'$ and $lmd(H) \neq lmd(H')$. If $yield(t_1), yield(t_2) \in L_n(G)$ we say that G is *n-ambiguous*. A non-ambiguous version of the term graphs grammar is given in Figure 3.

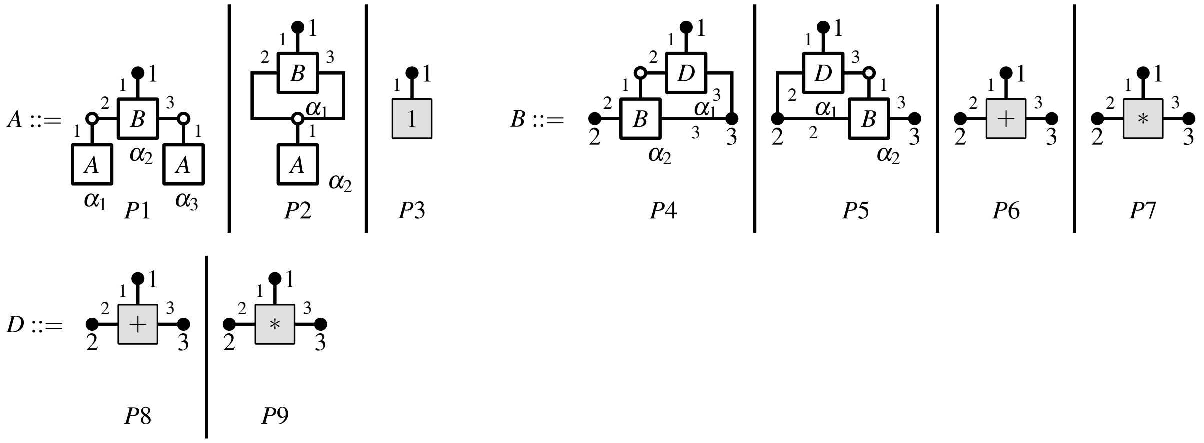


Figure 3: A non-ambiguous hyperedge replacement grammar for term graphs

3 Random hypergraph generation

In 1994, Mairson proposed a pair of methods for the sampling of strings from context-free grammars [11] . His approach requires, as input, a grammar G in Chomsky normal form and the length n of the word to be generated. He proves that, if G is non-ambiguous, such a word is generated uniformly at random. The first method has a time complexity of $O(n^2)$ while requiring $O(n)$ space, and vice versa, the second method runs in linear time using quadratic space. In the following we adapt the first of Mairson's methods to hyperedge replacement grammars. We use our running example of a term graph language to illustrate the generation process.

We define a *Chomsky normal form* (*CNF*) for hyperedge replacement grammars as a tuple $G_{CNF} = (N, \Sigma, P, S, (mark_p)_{p \in P})$ where:

- $N \subseteq C$ is a finite set of non-terminal labels
- $\Sigma \subseteq C$ is a finite set of terminal labels with $N \cap \Sigma = \emptyset$
- P is a finite set of productions
- $S \in N$ is the starting symbol

- $(mark_p)_{p \in P}$ is a family of functions $mark_p : E_R \to \{\alpha, \beta\}$ assigning a mark to each hyperedge in the right-hand side of a production p

Each production $p = (A, R) \in P$ satisfies one of the following constraints:

- $E_R = \{e_1, e_2\}$ where $lab(e_1), lab(e_2) \in N$ and $mark(e_1) \neq mark(e_2)$, in which case the replacement is firstly carried out on the hyperedge marked with α, then on the one marked with β
- $E_R = \{e_1\}$ where $lab(e_1) \in \Sigma$ and $mark(e_1) = \alpha$
- $E_R = \emptyset$, $|V_R| > |ext_R|$
- $A = S$, p is the empty production and for each $q \in P$, for each $e \in rhs(q)$, $lab(e) \neq S$

Note that in the first two cases, $rhs(p)$ contains either exactly two non-terminal hyperedges or a single terminal hyperedge and may also contain isolated nodes. Productions according to the third case are considered as *terminal* productions. The last case specifies that the empty production is only allowed if there is no other production having the starting symbol in its right-hand side. The grammar in Figure 4 is the *CNF* version of the term graph grammar in Figure 2.

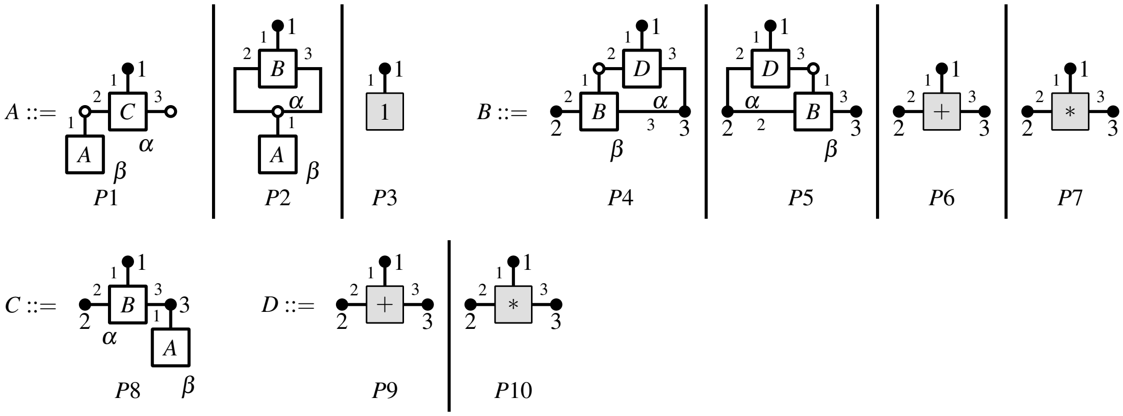


Figure 4: *CNF* of the grammar in Figure 2

Lemma 3.1. *There exists an algorithm that for every hyperedge replacement grammar G produces a grammar G' in CNF such that $L(G) = L(G')$.*

Proof. We present a set of rules to transform any grammar G, into an equivalent grammar G' such that, for each direct derivation $H \Rightarrow_p H'$ with $p \in P_G$, it exists an equivalent derivation $H \Rightarrow^*_Q H'$ with $Q \subseteq P_{G'}$. The proof is provided with a running example showing the application of the rules. The grammar in Figure 5 contains productions that are not in *CNF*: $P1$ has more than 2 hyperedges; $P2$ has a single non-terminal hyperedge; $P3$ is an empty production, but its *lhs* is not S; $P4$ has 2 hyperedges one of which is terminal.

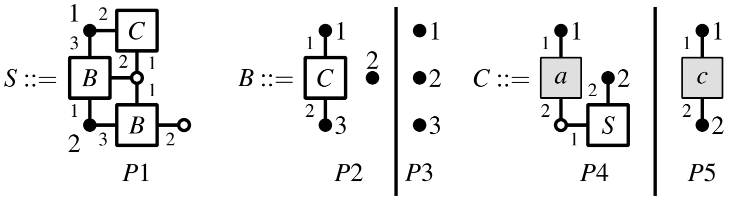


Figure 5: Starting grammar for the proof of *CNF* equivalence.

For a production $p = (A,R) \in P$, that is not already in *CNF*, we consider the following set of rules, applied in this order, to obtain a corresponding equivalent set of productions P' in *CNF*:

1. If p is the empty production, for each production $q = (B,X) \in P$ having $e \in E_X$ with $lab(e) = A$ in its *rhs*, for each production $q' = (A,Y) \in P$ having A in its *lhs* we apply the substitution $R' = X[e,Y]$ and add the productions $p = (B,R')$. We then remove the productions that are no longer needed. The proof of equivalence of the derivations $H \Rightarrow_q H' \Rightarrow_{q'} H''$ and $H \Rightarrow_{p'} H''$ is the following: if e' with $lab(e') = B$ is the hyperedge involved in the derivation $H \Rightarrow_q H'$ then $H'' = H[e'/X[e/Y]] = H[e'/R']$ since $R' = X[e,Y]$.

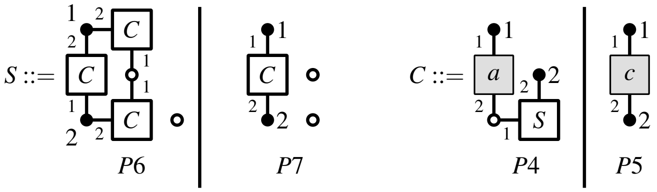


Figure 6: Removal of the empty production $P3$.

In order to remove the empty production $P3$ (Fig. 6) we apply the replacements of all the productions having B as their *lhs* to all the productions having a hyperedge labelled as B in their *rhs*. We remove $P1$ and introduce the productions $P6$ and $P7$. We then remove $P2$ and $P3$ since they are no longer needed.

2. If $E_R = \{e'\}$ with $lab(e') \in N$ for each production $q = (lab(e'),X) \in P$ we add the production $p' = (lab(e),R')$ with $R' = R[e'/X]$. If $E_{R'} = \{e''\}$ with $lab(e'') \in N$ this step is iterated and terminates when $|E_{R'}| > 1$ or $E_{R'} = \{e_t\}$ with $lab(e_t) \in \Sigma$ or $|E_{R'}| = 0$ and $|V_{R'}| > ext_{R'}$. The proof of equivalence of the derivations $H \Rightarrow_p H' \Rightarrow_q H''$ and $H \Rightarrow_{p'} H''$ is the following: if e' is the hyperedge involved in the derivation $H' \Rightarrow_q H''$ then $H'' = H'[e/R[e'/X]] = H[e/R']$ since $R' = R[e'/X]$.

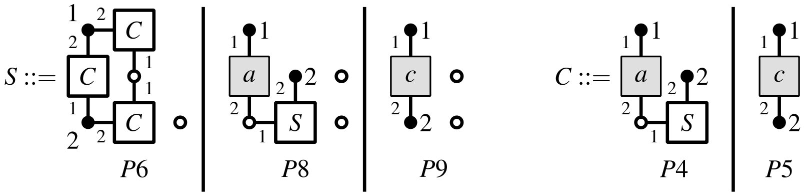


Figure 7: Removal of production $P7$

Since $P7$ has a single non-terminal hyperedge C (Fig. 7), we apply a replacement for each production that has C on its *lhs*. In our case, using the replacements of $P4$ and $P5$, we obtain $P8$ and $P9$. The production $P7$ is removed from the grammar.

3. If $|E_R| = k > 2$ we consider the subgraph X of R composed by the subset $E_X \subset E_R$ of hyperedges $e_2, \ldots, e_k$ and their attachment nodes. We introduce a new label T so that $N' = N \cup \{T\}$ and a new handle $T^\bullet$ of e_T with $ext(e_T) = \bigcup_{2 \leq i \leq k} att(e_i)$ such that $type(e_T) = type(X)$. We then consider the hypergraph R' composed by $R \backslash X$ and $T^\bullet$ where $V_{R'} = V_{R \backslash X} \cup V_{T^\bullet}$ and $E_{R'} = E_{R \backslash X} \cup E_{T^\bullet}$. Finally we add the productions $p' = (A,R'), p'' = (T,X)$ to P'. If $|E_X| > 2$ this step is iterated. The proof of equivalence of the derivations $H \Rightarrow_p H'$ and $H \Rightarrow^*_{P'} H'$ is the following: if e_a is the handle

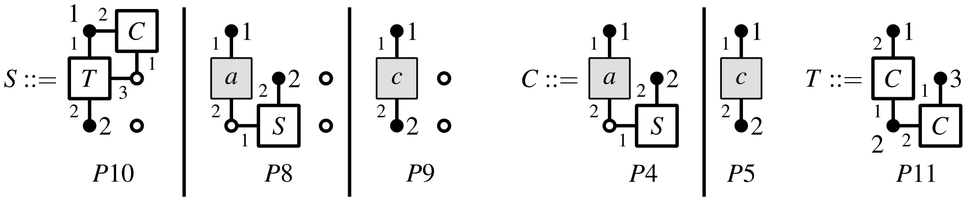


Figure 8: Removal of production $P6$

of the *lhs* of p we consider the following equivalence of the replacements then $H' = H[e_a/R] = H[e_a/R'[e_T/X]]$ since $R = R'[e_T/X]$.

Since production $P6$ has three non-terminal hyperedges (Fig. 8), we create a new label T, a new handle $T^\bullet$ and the production $P11$. Then we add the production $P10$ so that the replacement of the hyperedge labelled as T by the *rhs* of $P11$ results in the *rhs* of $P6$. The production $P6$ is then removed from the grammar.

4. If $|E_R| > 1$ and exists $e' \in E_R$ such that $lab(e') \in \Sigma$ a new label T is introduced so that $N' = N \cup \{T\}$. We add 2 new productions $p' = (A, R')$ to P' where $R' = R$ with $lab(e') = T$ and $p'' = (T, e'^\bullet)$. This step is repeated for each $e' \in E_R$ with $lab(e') \in \Sigma$. Due to the confluence property [3] of *HRG*s the order in which the terminal hyperedges are chosen is irrelevant. The proof of equivalence of the derivations $H \Rightarrow_p H''$ and $H \Rightarrow_{p'} H' \Rightarrow_{p''} H''$ is the following: if $e' \in E_R$ with $lab(e') \in \Sigma$ is the hyperedge involved in the derivation $H \Rightarrow_p H''$ then $H' = H[e/R] = H[e/R'[e'/e'^\bullet]]$ since $R = R'[e'/e'^\bullet]$.

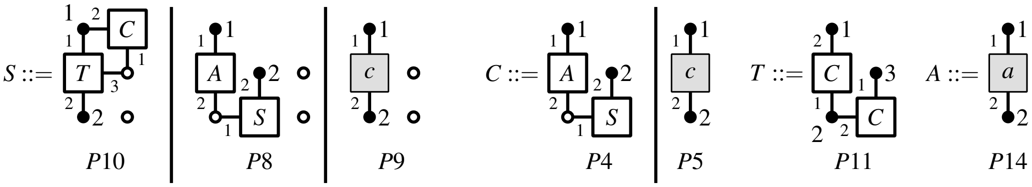


Figure 9: Removal of productions $P8$ and $P4$

Both *rhs* of productions $P4$ and $P8$ are composed by a terminal and a non-terminal hyperedge. We introduce a new label A and a its handle $A^\bullet$ along with the production $P14$ (Figure 9). We then add the productions $P12$ and $P13$ resulting from the substitution of the terminal hyperedges labelled with a by the non-terminal hyperedges labelled with A. Productions $P4$ and $P8$ are then removed from the grammar.

□

From this point on, if not explicitly specified, we always refer to an *HRG* as an *HRG* in *CNF*. We stress that the input of the method must be already provided in this form, that is, the time required for the transformation is not taken into account during the evaluation of the time complexity.

In order to complete the adaptation of the grammar we propose a more suitable short-hand representation of the productions that only extracts the necessary information, so, for each $p = (A, R) \in P$ and $i \in \mathbb{N}_0$ we use the following notations:

- $A \xrightarrow{p} BC, i$ for a *non-terminal* production where $B, C \in N$ are the labels of the marked hyperedges $e_\alpha, e_\beta \in R$ with $mark(e_\alpha) = \alpha$, $mark(e_\beta) = \beta$ and $i = |V_R \backslash ext_R|$.

- $A \xrightarrow{p} a, i$ for a *terminal* production where $a \in \Sigma$ is the label of the marked hyperedge $e_\alpha \in R$ and $i = |V_R \backslash ext_R|$.
- $A \xrightarrow{p} \lambda, i$ for a *terminal* production where $E_R = \emptyset$ and $i = |V_R \backslash ext_R|$.

Matrix M_2 (Tab. 1) shows the short-hand representation of the productions of the grammar in Figure 4. Considering a second input n as the size of the hypergraph to be generated, we are ready to describe a pair of algorithms (**Pre**, **Gen**) for the random sampling of a hypergraph H from a grammar G. Such a hypergraph is sampled in $L_n^A(G)$, where $A \in N$ is the non-terminal we begin the sampling from. If $A = S$ and G is n-unambiguous, H is sampled uniformly at random among all the hypergraphs in $L_n(G)$.

3.1 Pre-processing phase

The Pre-processing phase is used to construct a pair of matrices M_1, M_2 needed in the generation phase. Let $G = (N, \Sigma, P, S, (mark_p)_{p \in P})$ be an *HRG*, let $n \in \mathbb{N}$ be the size of the hypergraph $H \in L_n(G)$ we would like to generate, then the algorithm **Pre** (Alg. 1) produces the structures required for the generation.

Table 1: Matrices M_1 and M_2 resulting from $\textbf{Pre}(G', 12)$

M_1

N	1	2	3	4	5	6	7	8	9	10	11	12
A	1	0	2	0	14	0	92	0	616	0	3920	0
B	2	0	8	0	32	0	128	0	256	0	512	0
C	0	0	2	0	32	0	76	0	488	0	2928	0
D	2	0	0	0	0	0	0	0	0	0	0	0

M_2

P	1	2	3	4	5	6	7	8	9	10	11	12
$A \xrightarrow{P1} CA, 1$	0	0	0	0	2	0	16	0	128	0	992	0
$A \xrightarrow{P2} BA, 1$	0	0	2	0	12	0	76	0	488	0	2928	0
$B \xrightarrow{P4} DB, 1$	0	0	4	0	16	0	64	0	128	0	256	0
$B \xrightarrow{P5} DB, 1$	0	0	4	0	16	0	64	0	128	0	256	0
$C \xrightarrow{P8} BA, 1$	0	0	2	0	12	0	76	0	488	0	2928	0
$A \xrightarrow{P3} 1, 0$	1	0	0	0	0	0	0	0	0	0	0	0
$B \xrightarrow{P6} +, 0$	1	0	0	0	0	0	0	0	0	0	0	0
$B \xrightarrow{P7} *, 0$	1	0	0	0	0	0	0	0	0	0	0	0
$D \xrightarrow{P9} +, 0$	1	0	0	0	0	0	0	0	0	0	0	0
$D \xrightarrow{P10} *, 0$	1	0	0	0	0	0	0	0	0	0	0	0

We begin initializing the entries of two matrices $M_1 = (N \times \mathbb{N})$ and $M_2 = (P \times \mathbb{N})$ to 0. Each entry (A, ℓ) of M_1, also denoted as $A[\ell]$, represents the number of derivations yielding a hypergraph of size $\ell + type(A)$, from a non-terminal $A \in N$. Each entry (p, ℓ) of M_2, also denoted as $p[\ell]$ represents the number of derivations yielding a hypergraph of size $\ell + |ext_R|$, from a production $p \in P$. According to the type of production they are also denoted as $A \xrightarrow{p} \lambda, i[\ell]$ or $A \xrightarrow{p} a, i[\ell]$ for terminal productions and $A \xrightarrow{p} BC, i[\ell]$ for a non-terminal production. Considering each terminal production $p \in P_T$, either yielding a single terminal hyperedge $A \xrightarrow{p} a, i$ or at least a single isolated node $A \xrightarrow{p} \lambda, i$, the corresponding M_2 entry $p[i+1]$ in the former case, or $p[i]$ in the latter, is set to 1. Then, for each $\ell \in \mathbb{N}$ in $1 \leq \ell \leq n$, for each non-terminal $A \in N$, $A[\ell] = \sum_{p \in P^A} p[\ell]$ and for each production $p \in P_N$, $p[\ell] = \sum_{0<k<\ell} B, [k] \cdot C[\ell - k]$.

The matrices can be used to generate hypergraphs in $L^A(G)$ of size $\ell + type(A)$, with $1 \leq \ell \leq n$ from any non-terminal $A \in N$. If the non-terminal A is chosen before the pre-processing phase we can reduce the size of the tables to $n - type(A)$. Table 1 shows the result of running the algorithm **Pre** using the grammar G' in Figure 4 and a size of 12 as input.

3.2 Generation phase

In the generation phase a non-terminal $\bar{A} \in N$ is chosen and a size-$\bar{n}$-hypergraph H, with $1 \leq \bar{n} \leq n + type(A)$, is generated using the data collected in the matrices M_1, M_2 and a pseudo-random number generator *RNG*. The algorithm **Gen** (Alg. 2) describes this process.

On input $\mathbf{Gen}(G, \langle M_1, M_2 \rangle, \bar{A}, \bar{n} - type(A))$, if $\bar{A}[\bar{n} - type(A)] = 0$ the generating algorithm fails, otherwise, having $\bar{A}^\bullet$ as a basis, the algorithm recursively calls the function **derH** proceeding through the following steps:

1. The *RNG* is used to choose a production $p \in P^A$ with probability $p[\ell]/A[\ell]$.
2. If $p \in P^A_\Sigma$, the replacement of e, the *handle* of A, with the hypergraph R in $rhs(p)$ is returned.
3. If $p \in P^A_T$ the *RNG* is used again to choose a "split" $0 < k < \ell'$ with $\ell' = \ell - i$ and probability $B[k] \cdot C[\ell' - k]/A \xrightarrow{p} BC, i[\ell]$. The hypergraph $rhs(p)[e_\alpha/\mathbf{derH}(B,k), e_\beta/\mathbf{derH}(C, \ell' - k)]$ produced by the replacement of e_α with the result on the recursive function on input $\mathbf{derH}(B,k)$ and the replacement of e_β with the result of the recursive function on input $\mathbf{derH}(C, \ell' - k)$ is computed. Then, the replacement of the hyperedge e, the *handle* of A, with the aforementioned hypergraph is returned. We use the notation $B_k C_{\ell' - k}$ to indicate such a split.

The derivation $d = A^\bullet \Rightarrow^*_P H$ in Figure 10 corresponds to the sequence of replacements computed by the recursive function **derH** to generate the size-12-hypergraph H in Figure 1, using non-terminal A as input. For each step we show the probability of the production p to be chosen and the choice of the split and its probability if $p \in P^N$. Since G is non-ambiguous, the first step shows that $|L_{12}(G)| = 3920$, that is, there are 3920 unique size-12-hypergraphs to choose from, each having a different ordered derivation tree. Figure 11 shows the tree t for which $yield(t) = H$, so that $trav(t)$, or equivalently $lmd(H)$, corresponds to the unique sequence of productions applied by the generation algorithm to produce H. In the figure are also indicated the starting symbol A and the replaced hyperedges e_α and e_β, respectively on the edges connecting the left and right child of each node. The proof of termination of the Generation algorithm is based on the assumption that the input grammar is non-contracting:

Algorithm 1: Pre - Pre-processing phase

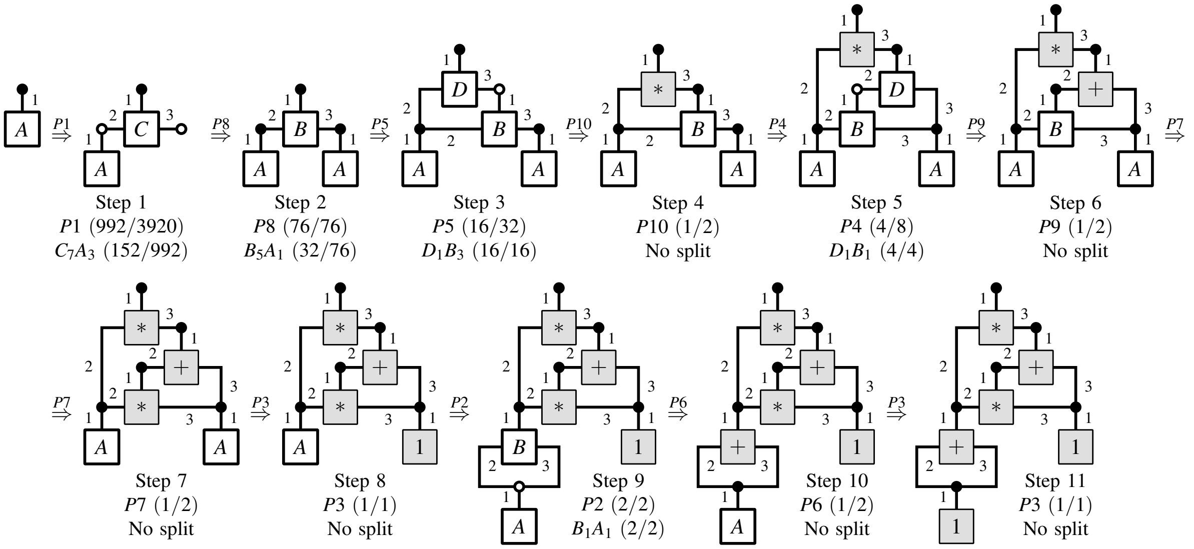
```
Input: $(G,n)$, where $G = (N, \Sigma, P, S, (mark_p)_{p \in P})$
  and $n \in \mathbb{N}$, $n \geq 1$
Output: $\langle M_1, M_2 \rangle$

for $1 \leq \ell \leq n$ do
    foreach $A \in N$ do
        $A[\ell] := 0$;
    end
    foreach $p \in P$ do
        $p[\ell] := 0$;
    end
end
foreach $A \xrightarrow{p} a, i \in P_\Sigma$ do
    $A \xrightarrow{p} a, i[i+1] := 1$;
end
foreach $A \xrightarrow{p} \lambda, i \in P_\Sigma$ do
    $A \xrightarrow{p} \lambda, i[i] := 1$;
end
for $1 \leq \ell \leq n$ do
    foreach $A \in N$ do
        foreach $p \in P^A$ do
            $A[\ell] := A[\ell] + p[\ell]$;
        end
    end
    foreach $A \xrightarrow{p} BC, i \in P_N$ do
        for $1 \leq k < \ell$ do
            $A \xrightarrow{p} BC, i[\ell + i] := A \xrightarrow{p}$
              $BC, i[\ell + i] + B[k] \cdot C[\ell - k]$;
        end
    end
end
```

Proof. Let's consider a measure equivalent to the size of a hypergraph $|H|$. To each application of the recursive function **derH** in each step of the algorithm **Gen**, corresponds a direct derivation between two

Algorithm 2: Gen - Generation phase

```
Input: $(G, \langle M_1, M_2\rangle, \bar{A}, \bar{n})$, where $G = (N, \Sigma, P, S, (mark_p)_{p \in P})$, $\langle M_1, M_2\rangle := \mathbf{Pre}(G, n)$, $\bar{A} \in N$
  and $\bar{n} \in \mathbb{N}$, $1 \leq \bar{n} \leq n + type(\bar{A})$
Output: $H \in L_{\bar{n}}^{\bar{A}}(G)$
$\ell = \bar{n} - type(\bar{A})$
if $\bar{A}[\ell] = 0$ then
    return $\perp$;
end
Recursively generate $H$ using $(\bar{A}, \ell)$ as first input as follows:
function derH $(A,\ \ell)$:
    $p \longleftarrow RNG$ with $p \in P^A$ and probability $p[\ell]/A[\ell]$;
    if $p \in P_T$ then
        return $A^\bullet[e/R]$;
    else
        $\ell' = \ell - i$;
        $k \longleftarrow RNG$ with $0 < k < \ell'$ and probability $B[k] \cdot C[\ell' - k]/(A \xrightarrow{p} BC, i)[\ell]$;
        return $A^\bullet[e/R[e_\alpha/\mathbf{derH}(B, k), e_\beta/\mathbf{derH}(C, \ell' - k)]]$;
    end
end function
```



Step 1
$P1$ (992/3920)
C_7A_3 (152/992)

Step 2
$P8$ (76/76)
B_5A_1 (32/76)

Step 3
$P5$ (16/32)
D_1B_3 (16/16)

Step 4
$P10$ (1/2)
No split

Step 5
$P4$ (4/8)
D_1B_1 (4/4)

Step 6
$P9$ (1/2)
No split

Step 7
$P7$ (1/2)
No split

Step 8
$P3$ (1/1)
No split

Step 9
$P2$ (2/2)
B_1A_1 (2/2)

Step 10
$P6$ (1/2)
No split

Step 11
$P3$ (1/1)
No split

Figure 10: A derivation $d = A^\bullet \Rightarrow_P^* H$ using the grammar of Figure 4

sentential forms $F \Rightarrow F'$ such that $F \leq F'$. Since the grammar is in *CNF*, at each step there are two possible cases:

1. **derH** chooses a non-terminal production. In this case a single hyperedge $e \in F$ is replaced with a hypergraph $R \subseteq F'$ containing 2 hyperedges and 0 or more internal nodes. Clearly $|F| < |F'|$, meaning that the size of the sentential forms gets progressively close to n.
2. **derH** chooses a terminal production. A hyperedge is replaced by a terminal hyperedge or a single node and 0 or more additional internal nodes. In this case $|F| \leq |F'|$. Even if the size is not incremented, being a terminal production, the recursion does not progress any further.

If it is not possible to generate a size-n-hypergraph using the input grammar G the algorithm trivially ends in one step. □

4 Uniform distribution and time complexity

We now state our first main result, the uniform generation guarantee for Algorithm 2.

Theorem 4.1. *Given a grammar $G = (N, \Sigma, P, S, (mark_p)_{p \in P})$, Algorithm 2 generates from every non-terminal $A \in N$ a size-n-hypergraph $H \in L_n^A(G)$, provided that $L_n^A(G) \neq \emptyset$. If G is n-unambiguous and RNG is a uniform random number generator, the hypergraph is chosen uniformly at random.*

Proof. Let G be an n-unambiguous grammar in *CNF*, the recursive function **derH** derives a hypergraph $H \in L_{\bar{n}}^{\bar{A}}(G)$ simulating $trav(t)$ where $yield(t) = H$ and let $P(c_j)$ denote the probability of the jth choice c made using the *RNG* at each step of the recursion, for a production or a split, according to $lmd(H)$.

Let's recall that for the parallelization, confluence and associativity properties of context-free hyperedge replacement grammars [3], the sequence of replacements associated to a derivation preserves the result of the derivation, despite of the order in which the replacements are applied. Thus, we are able to discuss each of its steps independently.

By definition, since the grammar is n-unambiguous, for any non-terminal $A \in N$ we know that the set of hypergraphs that can be generated using different productions $p \in P^A$ are pairwise distinct. Otherwise, there would exist $trav(t') \neq trav(t'')$ for which $yield(t') \cong yield(t'')$.

From algorithm **Pre** (Alg. 1) we know that $\sum_{p \in P^A} p[\ell] = A[\ell]$ and so the probability of the choice c_j of each production in $lmd(H)$ can be expressed by $P(c_j) = p[\ell]/A[\ell]$. Also, if $p \in P_N$, since the grammar is n-unambiguous the subsets of hypergraphs that can be derived by choosing different splits are also pairwise distinct. For a production $p \in P_N$ then $\sum_{0<k<\ell} B[k] \cdot C[\ell' - k] = A \xrightarrow{p} BC, i[\ell]$, thus a split can be chosen with probability $P(c_j) = B[k] \cdot C[\ell' - k]/p[\ell]$.

Knowing that for an *lmd*, if the grammar is n-unambiguous, both the choices of productions and splits are made from independent sets, considering the corresponding derivation tree t, the probabilities associated to the choice of a node $P(c)$ and the ones associated to its children $P(c')$ and $P(c'')$ are of the form $\frac{m}{q}$, $\frac{m'}{q'}$ and $\frac{m''}{q''}$ with $m, m', m'', q, q', q'' \in \mathbb{N}$ and $q'q'' = m$. Moreover, the probabilities of two consecutive choices $P(c)$ and $P(c')$ are bound to the law of compound probabilities [10], that is, the choice of a node given the choice of its parent is of the form $P(c'|c) = P(c' \cap c)/P(c)$. Then, considering their independence, $P(c'|c) = (P(c')P(c))/P(c) = P(c')$. The same applies for $P(c'')$. The overall probability of the choice of a node and its children is then $P(c)P(c')P(c'') = \frac{m}{q}\frac{m'}{q'}\frac{m''}{q''} = \frac{m'm''}{q}$.

Finally, considering the chain of probabilities described by an *lmd*, since for $\bar{A}$ $q = |L_{\bar{n}}(G)|$ and for each terminal production $p \in P_\Sigma$ $m = 1$, then for each $H \in L_{\bar{n}}(G)$ we can define its probability $P(H)$ to be generated as the productory of independent choices:

$$P(H) = \prod_{j=1}^{k} P(c_j) = \frac{m_1}{|L_{\bar{n}}(G)|} \cdot \frac{m_2}{q_2} \cdot \frac{m_{k-1}}{q_{k-1}} \dots \frac{1}{q_k} = \frac{1}{|L_{\bar{n}}(G)|}$$

Each hypergraph $H \in L_{\bar{n}}(G)$ is generated over a uniform distribution given the uniformity of the sampling of the underlying *RNG*.

□

For the complexity analysis we consider the time required by the algorithm **Gen** (Alg. 2) for the generation of the hypergraph and the space required by the algorithm **Pre** (Alg. 1) to store the required data, taking into account that the input grammar is already provided in the correct *CNF* and the query to the *RNG* and the replacement operations are performed in unit time. The gaps present in the tables, that are not encountered in string method, are due to the possibility of a production to increase the size of the resulting hypergraph by more than 1 in a single step.

Theorem 4.2. *With the assumptions of Theorem 4.1, the size-n-hypergraph H is generated by* **Gen** *(Alg. 2) in time $O(n^2)$.*

Proof. The proof of Theorem 4.2 is based on the analysis of the following recurrence relation for the function **derH**: $T(n) \leq cn + \max\limits_{1 \leq k < (n-i)} [T(k) + T(n-k-i)]$, where $T(k)$ and $T(n-k-i)$ are the computational steps required to process the result of the split and i is the number of internal nodes of the current production. In the worst case, we consider that $i = 0$ and that $k = 1$. A simple example is the discrete hypergraph language in which every iteration may generate a terminal hyperedge from e_α and the rest of the resulting hypergraph from e_β without adding any node. Since the choice of the production is constant, while the choice of a split is linear in n, choosing a split n times leads to a quadratic behavior.

Since $i \ll n$, we may rewrite the recursion as:

$$T(n) \leq cn + \max_{1 \leq k < n} [T(k) + T(n-k)]$$

Then, considering the worst case $k = 1$, for the next step of the recursion we obtain:

$$T(n-1) \leq c(n-1) + \max_{1 \leq k < (n-1)} [T(k) + T(n-k-1)]$$

That is, at each step the choice of a split happens on an input of size $n-1$. Since this choice requires linear time and it is taken n times, the relation has solution $O(n^2)$. □

We omit a discussion of the time complexity of the pre-processing phase (Alg. 1) which can be shown to be linear, considering that given a grammar G in *CNF*, being its size $|G|$ constant, for each production p a short form containing the information about the labels and the internal nodes is obtained in constant time.

5 Conclusion

Our main results, presented in Section 4, are that the method generates hypergraphs uniformly at random and in quadratic time. A topic for future work is to design an alternative generation algorithm that runs in linear time and quadratic space, following Mairson's second method in [11].

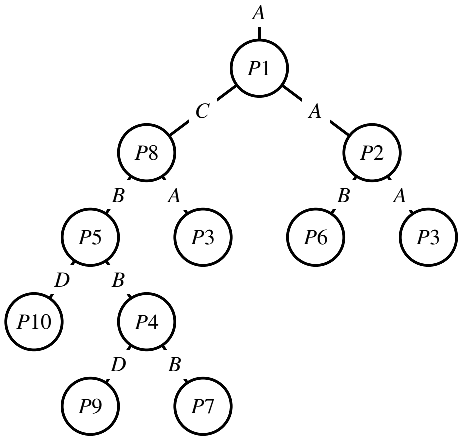


Figure 11: Ordered tree t for the derivation d in Figure 10

Another interesting topic is to extend the quasi-polynomial-time approximation algorithm of Gore et al. [7] from strings to hypergraphs. This algorithm guarantees an approximated uniform distribution even for ambiguous grammars.

Our method allows to generate strings uniformly at random in some non-context-free string languages because hyperedge replacement grammars can specify certain *string graph* languages that are not context-free. For example, this applies to the language $\{a^n b^n c^n \mid n \geq 0\}$. Moreover, our method is able to generate strings uniformly at random for a range of inherently ambiguous context-free languages.

The practically most promising application of our generation approach is the testing of programs in arbitrary programming languages that work on graphs. If the inputs of such programs are graphs in a context-free graph language, our method can generate test graphs uniformly at random in the domain of interest. This should allow to refine random testing approaches such as [1, 8].

References

[1] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. Tse. Adaptive random testing: The art of test case diversity. *Journal of Systems and Software*, 83(1):60–66, 2010. doi:10.1016/j.jss.2009.02.022.

[2] N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, 1959. doi:10.1016/S0019-9958(59)90362-6.

[3] B. Courcelle. An axiomatic definition of context-free rewriting and its application to NLC graph grammars. *Theoretical Computer Science*, 55(2):141–181, 1987. doi:10.1016/0304-3975(87)90102-2.

[4] F. Drewes, A. Habel, and H.-J. Kreowski. Hyperedge replacement graph grammars. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1, pages 95–162. World Scientific, 1997. doi:10.1142/3303.

[5] J. Engelfriet. Context-free graph grammars. In *Handbook of Formal Languages*, volume 3, pages 125–213. Springer, 1997. doi:10.1007/978-3-642-59126-6_3.

[6] O. Goldreich. Candidate one-way functions based on expander graphs. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 76–87. Springer, 2011. doi:10.1007/978-3-642-22670-0_10.

[7] V. Gore, M. Jerrum, S. Kannan, Z. Sweedyk, and S. Mahaney. A quasi-polynomial-time algorithm for sampling words from a context-free language. *Information and Computation*, 134(1):59–74, 1997. doi:10.1006/inco.1997.2621.

[8] R. Hamlet. Random testing. In *Encyclopedia of Software Engineering*. John Wiley and Sons, 2002. doi:10.1002/0471028959.sof268.

[9] H. Kajino. Molecular hypergraph grammar with its application to molecular optimization. In Proceedings *36th International Conference on Machine Learning (ICML 2019)*, volume 97 of *Proceedings of Machine Learning Research*, pages 3183–3191. PMLR, 2019. URL `https://proceedings.mlr.press/v97/kajino19a.html`.

[10] P. S. le Marquis de Laplace. Théorie analytique des probabilités. In *Œuvres Completes de Laplace*, volume 7, pages 181–192. Gauthier-Villars, Imprimeur-Librarie, 3rd edition, 1820. URL `http://eudml.org/doc/203444`.

[11] H. G. Mairson. Generating words in a context-free language uniformly at random. *Information Processing Letters*, 49(2):95–99, 1994. doi:10.1016/0020-0190(94)90033-7.

[12] S. Micali and R. L. Rivest. Transitive signature schemes. In Proceedings *Topics in Cryptology (CT-RSA 2002)*, volume 2271 of *Lecture Notes in Computer Science*, pages 236–243. Springer, 2002. doi:10.1007/3-540-45760-7_16.

[13] D. Plump. Term graph rewriting. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2, pages 3–61. World Scientific, 1999. doi:10.1142/9789812815149_0001.