# Random Grammar-based Testing for Covering All Non-Terminals

Aloïs Dreyfus, Pierre-Cyrille Héam and Olga Kouchnarenko
FEMTO-ST - CNRS UMR6174 - Université de Franche-Comté - INRIA CASSIS
16 route de Gray - 25030 Besançon, France
Email: firstname.lastname@femto-st.fr

*Abstract*—**In the context of software testing, generating complex data inputs is frequently performed using a grammar-based specification. For combinatorial reasons, an exhaustive generation of the data – of a given size – is practically impossible, and most approaches are either based on random techniques or on coverage criteria. In this paper, we show how to combine these two techniques by biasing the random generation in order to optimise the probability of satisfying a coverage criterion.**

*Keywords*-**Random testing, Grammar-based testing.**

## I. Introduction

### A. Motivation

Producing trusted software is a central issue in software engineering. Testing remains an inescapable step to ensure software quality. In reaction to the limitations of manual testing, recent years have seen a rise in the research interest for systematic testing frameworks grounded in theory. Random testing is a natural approach, empirically known to detect many kinds of bugs. However, by definition, low-probability behaviours cannot be adequately tested in that way. Conversely, non-random testing tends to focus on a few edge cases of particular interest to the tester, at the expense of all others. Indeed, it can cover various behaviours, but their choice depends on tester's priorities and, in general, each behaviour is tested in a unique way.

In [1], it is explained how to bias a uniform random testing approach using constraints given by a coverage criterion, in order to optimise the probability of satisfying this criterion. The technique is developed for path generation in a graph. The contribution of the present paper consists in enriching this approach with a coverage criterion on non-terminal symbols of the grammar, allowing the user to apply it to grammar-based testing.

### B. Related Work

Grammar-based testing is frequently used for generating structured inputs, as in [2] for parser testing or in [3] to test refactoring engines (program transformation software). Systematic combinatorial approaches [4] lead to a huge number of sequences, and symbolic approaches are frequently preferred [5], [6], [7]. In [8], a generic tool for generating test data from grammars has been proposed. This tool does not provide any random feature but is based on rule coverage algorithms and techniques, as defined in [2], [9], [10], [11].

Random test generation techniques – initially proposed in [12], [13] – are frequently used for practical reasons, as in [14], [15], [1]. Combining random generation and grammar-based testing is explored in [16], [17], [18], [19], [20], [21], without exploiting any coverage criteria, or using an isotropic random walk as in [22].

### C. Layout

Section II presents the notions and notations used in this paper. Section III explains how to optimise random testing to satisfy a given coverage criterion. The theoretical contributions are provided in Section IV, which shows how to use this technique to optimise the coverage of non-terminal symbols in a grammar-based testing context. An illustrating example is developed in Section V. Finally, Section VI concludes.

## II. Formal Background

### A. Context-free Grammars and Random Generation

In this paper, the cardinality of a finite set $\mathcal{S}$ is denoted $|\mathcal{S}|$.

*a) Context-free Grammars:* A *context-free grammar* is a tuple $G = (\Sigma, \Gamma, S_0, R)$, where $\Sigma$ and $\Gamma$ are disjoint finite alphabets, $S_0 \in \Gamma$ is the initial symbol, and $R$ is a finite subset of $\Gamma \times (\Sigma \cup \Gamma)^*$. The elements of $\Sigma$ are called *terminal symbols*, and the elements of $\Gamma$ are called *non-terminal symbols*. An element $(X, u)$ of $R$ is called *a rule* of the grammar and is frequently denoted $X \rightarrow u$. A word $w \in (\Sigma \cup \Gamma)^*$ is a *successor* of $v \in (\Sigma \cup \Gamma)^*$ for the grammar $G$ if there exist $v_0 \in \Sigma^*$, $v_1, v_2 \in (\Sigma \cup \Gamma)^*$, $S \in \Gamma$ such that $v = v_0 S v_1$ and $w = v_0 v_2 v_1$ and $S \rightarrow v_2 \in R$. A *complete derivation*[1] of the grammar $G$ is a finite sequence $x_0, \ldots, x_k$ of words of $(\Sigma \cup \Gamma)^*$ such that $x_0 = S_0$, $x_k \in \Sigma^*$ and for every $i$, $x_{i+1}$ is a successor of $x_i$. A *derivation tree* of $G$ is a finite tree whose internal nodes are labelled by letters of $\Gamma$, whose leaves are labelled by elements of $\Sigma \cup \{\varepsilon\}$, whose root is labelled by $S_0$ and satisfying: if a node is labelled by $X \in \Gamma$ and its children are labelled by $\alpha_1, \ldots, \alpha_k$ (in this order), then either $\alpha_1 = \varepsilon$ and $k = 1$, or all the $\alpha_i$'s are in $\Gamma \cup \Sigma$ and $(X, \alpha_1 \ldots \alpha_k) \in R$. The size of a derivation tree is given by the number of tree nodes.

**Example 1 – Context-free grammar.** *Let us consider the grammar $G = (\{a, b\}, \{S, T\}, S, R)$, with $R = \{S \rightarrow$*

[1]As $v_0 \in \Sigma^*$, this derivation is obviously a left-most derivation.

*$Tb, S \to aSb, T \to \varepsilon\}$). The sequence $S, aSb, aTbb, abb$ is a complete derivation of the grammar. The associated derivation tree is*

$$\begin{tikzcd}
& a & & \\
S \arrow[ur, dash] \arrow[r, dash] \arrow[dr, dash] & S \arrow[r, dash] \arrow[dr, dash] & T \arrow[r, dash] & \varepsilon \\
& b & b &
\end{tikzcd}$$

Note that there is a bijection between the set of complete derivations of a grammar and the set of derivation trees of this grammar. For a context-free grammar $G$, $E_n(G)$ denotes the number of derivation trees of $G$ with $n$ nodes. A derivation tree *covers* an element $X$ of $\Gamma$ if at least one of its nodes is labelled by $X$. For instance, for the tree in Example 1, the elements $S$ and $T$ are covered since they appear in the derivation tree.

*b) Uniform Random Generation:* The present issue is, given a positive integer and a context-free grammar, to compute randomly with a uniform distribution a derivation tree of size $n$ of this grammar. We will briefly explain here how to tackle this problem by using well-known counting techniques [23]. Notice that more advanced techniques allow a faster computation, like in [24].

As usual, the non-terminals symbols are denoted by capital letters. Given a context-free grammar $G = (\Sigma, \Gamma, S_0, R)$, a non-terminal symbol $X$ in $\Gamma$, and a positive integer $i$, the number of derivation trees of size $i$ generated by $(\Sigma, \Gamma, X, R)$ is denoted by $x(i)$, i.e., using the corresponding lowercase letter.

Given a positive integer $n$, for each symbol $S \in \Gamma$, the sequence of positive integers $s(1), \ldots, s(k), \ldots$ is introduced. The recursive computation of these $s(i)$'s is as follows. For each strictly positive integer $k$ and each rule $r = (S, w_1 S_1 \ldots w_n S_n w_{n+1}) \in R$, with $w_j \in \Sigma^*$ and $S_i \in \Gamma$, let us set

$$\begin{cases} \beta_r = 1 + \sum_{i=1}^{n+1} |w_i| \\ \alpha_r(k) = \sum_{i_1+i_2+\ldots+i_n = k-\beta_r} \prod_{j=1}^{j=n} s_j(i_j) \quad \text{if } n \neq 0 \\ \alpha_r(k) = 0 \quad \text{if } n = 0 \text{ and } k \neq \beta_r \\ \alpha_r(\beta_r) = 1 \quad \text{if } n = 0. \end{cases}$$

It is known [23, Theorem I.1] that $s(k) = \sum_{r \in R \cap (S \times (\Sigma \cup \Gamma)^*)} \alpha_r(k)$.

Since, by hypothesis, there is no rule of the form $(S, T)$ in $R$, with $S, T \in \Gamma$, all $i_j$'s involved in the definition of $\beta_r$ are strictly less than $k$. This way, the $s(i)$'s can be recursively computed. Consider for instance the grammar $(\{a, b\}, \{X\}, X, \{r_1, r_2, r_3\})$ with $r_1 = (X, XX)$ $r_2 = (X, a)$ and $r_3 = (X, b)$. One has $\beta_{r_1} = 1 + 0 = 1$, $\beta_{r_2} = 1 + 1 = 2$, $\beta_{r_3} = 1{+}1 = 2$. Therefore $x(k) = \sum_{i+j=k-1} x(i)x(j)$ if $k \neq 2$, and $x(2) = 1 + 1 + \sum_{i+j=2-1} x(i)x(j) = 2$, otherwise. It follows that $x(1) = 0$, $x(2) = 2$, $x(3) = x(1)x(1) = 0$, $x(4) = x(1)x(2) + x(2)x(1) = 0$, $x(5) = x(2)x(2) = 4$, etc. The two derivation trees of size 2 are $\overset{X}{\underset{a}{|}}$ and $\overset{X}{\underset{b}{|}}$. The four derivation trees of size 5 are the trees of the form $\overset{X}{\underset{Z_1\ Z_2}{/\backslash}}$ where both $Z_1$ and $Z_2$ are derivation trees of size 2.

In order to generate derivation trees of size $n$, all $s(i)'s$, for $S \in \Gamma$ and $i \leq n$, have to be computed with the above method. This can be performed in polynomial time. Afterwards, the random generation is done recursively using the given algorithm in Fig. 1.

**Random Generation**

**Input:** $G = (\Sigma, \Gamma, X, R)$ a context-free grammar, $n$ a strictly positive integer.

**Output:** a derivation tree $t$ of $G$ size $n$.

**Algorithm:**

```
1. Let $\{r_1, r_2, \ldots, r_\ell\}$ be set of the elements of $R$ whose first element is $X$.
2. If $\sum_{j=1}^{j=\ell} \alpha_{r_j}(n) = 0$, then return "Exception".
3. Pick $i \in \{1, \ldots, \ell\}$ with probability $Prob(i = j) = \frac{\alpha_{r_i}(n)}{\sum_{j=1}^{j=\ell} \alpha_{r_j}(n)}$.
4. Let $r_i = (X, Z_1 \ldots Z_k)$, with $Z_j \in \Sigma \cup \Gamma$.
5. Root symbol of $t$ is $X$.
6. Children of $t$ are $Z_1, \ldots, Z_k$ in this order.
7. Let $\{i_1, \ldots, i_m\} = \{j \mid Z_j \in \Gamma\}$.
8. Pick $(x_1, \ldots, x_m) \in \mathbb{N}^m$ such that $x_1 + \ldots + x_m = n - \beta_{r_i}$ with probability
       $$Prob(x_1 = \ell_1, \ldots, x_m = \ell_m) = \frac{\prod_{j=1}^{j=m} z_{i_j}(\ell_j)}{\alpha_{r_i}(n)}.$$
9. For each $i_j$, the $i_j$-th sub-tree of $T$ is obtained by running the Random Generation algorithm on $(\Sigma, \Gamma, Z_{i_j}, R)$ and $\ell_j$.
10. Return $t$.
```

Fig. 1. Random Generation algorithm

It is known [23] that this algorithm provides a uniform generation of derivation trees of size $n$, i.e. each derivation tree occurs with the same probability. Note that an exception is raised at Step 2 if there is no element of the given size. For the example presented before, there is no element of size 3, then it is impossible to generate a derivation tree of size 3. Running the algorithm on this example with $n = 2$, one considers at Step 1 the set $\{r_1, r_2, r_3\}$ since all these rules have $X$ as left element. Since $\alpha_{r_1}(2) = 0$, $\alpha_{r_2}(2) = 1$, $\alpha_{r_3}(2) = 1$, at Step 3 the probability that $i = 1$ is 0, the probability that $i = 2$ is 1/2 and the probability that $i = 3$ is 1/2. If $i = 2$ is picked, the generated tree has $X$ as root symbol and $a$ as unique child. Running the algorithm on this example with $n = 3$ stops at Step 2 since there is no tree of size 3. When running the algorithm on this example with $n = 5$, the set $\{r_1, r_2, r_3\}$ is considered at Step 1. Since $\alpha_{r_1}(5) = 4$, $\alpha_{r_2}(5) = 0$, $\alpha_{r_3}(5) = 0$, $i = 1$ is picked with probability 1. Therefore, the tree has $X$ as root symbol, and its two children are both labelled by $X$. Therefore, at Step 7, the considered set is $\{1, 2\}$. At Step 8, one has $n - \beta_{r_1} = 5 - 1 = 4$. The probability that $i_1 = 1$ and $i_2 = 3$ is 0 since $x(1) = 0$. Similarly, the probability that $i_1 = 3$ and $i_2 = 1$ is 0 too. Now the probability that $i_1 = 2$ and $i_2 = 2$ is 1. Afterwards the algorithm is recursively executed on each child with $n = 2$:

each of the 4 trees is chosen with probability $1/4$.

## III. Mixing Random Testing and Coverage Criteria

In a context of functional testing, the strength of random testing is to quickly provide many different test data, for each behaviour of the system. Moreover, these test data are independent of the choices of the test designer, and consequently they can catch problem (s)he did not anticipate. For instance, fuzz testing is particularly relevant for testing security requirements [7]. However, random testing can miss an important behaviour occurring with a very small probability. To exploit the advantages of both random testing and deterministic testing, a solution is to combine random generation and coverage criteria.

The general schema for this combination, as described in [1], is the following: considering a random generation algorithm of test data of size $n$ and a coverage criteria $C$ (each element of $C$ is or is not covered by each possible test), the goal is to use the generation algorithm $N$ times in order to optimise the probability of covering all elements of $C$. For each element $e \in C$, we denote by $p_{e,n}$ the probability that a generated test of size $n$ covers $e$. One can easily check that generating $N$ test data independently of $C$ provides a probability of covering $C$ of $1-(1-p_{\min})^N$, where $p_{\min} = \min_{e\in C}\{p_{e,n}\}$. This probability is the way to measure the quality of the testing approach, relatively to $C$. A better way is to repeat $N$ times the following procedure:

1) Pick at random an element $e \in C$ with a probability $\pi_e$, and
2) Generate uniformly a test of size $n$ covering $e$.

This procedure requires to know how to uniformly generate a test of size $n$ covering a given element, and to choose the probabilities $\pi_e$'s to optimise the probability of covering all elements of $C$.

Following [1], the optimisation requires solving the following constraint system: maximise $p$ satisfying

$$\begin{cases} p \leq \sum_{e\in C} \pi_e \frac{p_{e,f,n}}{p_{e,n}} \text{ for all } f \in C \\ \sum_{e\in C} \pi_e = 1 \end{cases}$$

where $p_{e,f,n}$ is the probability that a randomly generated test of size $n$ covers both $e$ and $f$. This linear programming problem can be solved in an efficient way, using simplex-like approaches.

In summary, in order to combine random testing and a coverage criterion, it is required to solve a constraint system and to know 1) how to randomly generate a test of a given size covering a given element, 2) how to compute the $p_{e,n}$'s; and 3) how to compute the $p_{e,f,n}$'s.

The rest of the paper is dedicated to the problem of the random generation of execution trees of a grammar, with the coverage criterion *All non-terminal symbols*. More precisely, given a grammar $G = (\Sigma, \Gamma, S_0, R)$, the coverage criterion being $\Gamma$, a test of size $n$ being a derivation tree of $G$ of size $n$, we say that $X \in \Gamma$ is covered by a test if the derivation tree covers $X$.

## IV. Computing $p_{X,n}$ and $p_{X,Y,n}$

In this section $G = (\Sigma, \Gamma, S_0, R)$ is a context-free grammar. We denote by $E_n(G)$ the set of derivation trees of size $n$ of $G$. We respectively denote by $E_{X,n}(G)$ and $E_{X,Y,n}(G)$ the set of derivation trees of size $n$ of $G$ covering $X$, and covering both $X$ and $Y$ .

Let $p_{X,n}$ be the probability of a randomly generated derivation tree of size $n$ to cover $X$. Clearly, if $E_n(G)$ is empty then $p_{X,n} = 0$ [resp. $p_{X,Y,n} = 0$]. Otherwise, $p_{X,n} = \frac{|E_{X,n}(G)|}{|E_n(G)|}$ [resp. $p_{X,Y,n} = \frac{|E_{X,Y,n}(G)|}{|E_n(G)|}$].

Therefore, computing the probability defined in Section III – needed to solve the linear constraint program – reduces to the computation of the cardinality of sets $E_{X,n}(G)$ and $E_{X,Y,n}(G)$.

### A. Computing $|E_{X,n}(G)|$ and $|E_{X,Y,n}(G)|$

To compute $|E_{X,n}(G)|$, we build a grammar $G_X$ such that $E_n(G_X)$ and $E_{X,n}(G)$ are in bijection (and therefore have the same number of elements).

For every $w \in (\Gamma \cup \Sigma)^*$, $[w]_0$ is recursively defined by: $[\varepsilon]_0 = \varepsilon$, $[Zw]_0 = (Z, 0)[w]_0$ (with $Z \in \Gamma$) and $[aw]_0 = a[w]_0$ (with $a \in \Sigma$). Intuitively, $[w]_0$ is obtained from $w$ by changing each letter of $w$ in $\Gamma$ by the corresponding pair with 0 as second element. For instance, with the grammar of Example 1, one has $[aSbbT]_0 = a(S, 0)bb(T, 0)$. For every $w \in (\Gamma \cup \Sigma)^*$, $[w]_2$ is defined exactly in the same way, changing all 0's by 2's.

For every $w \in (\Gamma\cup\Sigma)^*$, $\{w\}_{1,2}$ is defined as the set of words $w' \in (\Sigma \cup \Gamma \times \{1, 2\})^*$ obtained from $w$ by replacing occurrence of each letter $Z$ of $\Gamma$ either by $(Z, 1)$ or by $(Z, 2)$, with the restriction that at least one is replaced by $(Z, 1)$. The letters in $\Sigma$ remain unchanged. For instance, if $w = aSbT$, then $\{w\}_{1,2} = \{a(S, 1)b(T, 1), a(S, 2)b(T, 1), a(S, 1)b(T, 2)\}$. Notice that if $w \in \Sigma^*$ then $\{w\}_{1,2} = \emptyset$ since the constraint is not satisfied.

Let $G_X = (\Sigma, \Gamma\times\{0, 1, 2\}, (S_0, 1), R_X)$ where $R_X = R_0\cup R_1 \cup R'_1 \cup R_2$ with:

- $R_0 = \{(Z, 0) \to [w]_0 \mid Z \to w \in R\}$,
- $R_1 = \{(Z, 1) \to w' \mid Z \neq X$ and $\exists Z \to w \in R$ such that $w' \in \{w\}_{1,2}\}$,
- $R'_1 = \{(X, 1) \to [w]_0 \mid X \to w \in R\}$,
- $R_2 = \{(Z, 2) \to [w]_2 \mid Z \to w \in R$ and $Z \neq X\}$.

Intuitively, adding the value 0 to a symbol in $\Gamma$ means that if this rule is used, there exists an occurrence of $X$ at an upper position in the derivation tree. Adding the value 1 to a symbol in $\Gamma$ means that there is no occurrence of $X$ at an upper position, but there exists an occurrence of $X$ at this or a lower position in the derivation tree. The value 2 means there is no occurrence of $X$ appearing in the tree at an upper or lower position.

**Example 2 – $G_X$.** *Consider the grammar $G = (\{a, b\}, \{S, T, X\}, S, R)$ with $R = \{S \to SS, S \to aT, S \to Xb, T \to aa, X \to TX, X \to b\}$. The grammar $G_X$ has the set of rules as follows: $\{(S, 0) \to (S, 0)(S, 0), (S, 0) \to a(T, 0), (S, 0) \to$*

$(X,0)b, (T,0) \to aa, (X,0) \to b, (X,0) \to (T,0)(X,0)\} \cup \{(S,1) \to (S,1)(S,1), (S,1) \to (S,1)(S,2), (S,1) \to (S,2)(S,1), (S,1) \to a(T,1), (S,1) \to (X,1)b\} \cup \{(X,1) \to b, (X,1) \to (T,0)(X,0)\} \cup \{(S,2) \to (S,2)(S,2), (S,2) \to a(T,2), (S,2) \to (X,2)b, (T,2) \to aa\}$.

**Proposition 1 – Bijection.** *There exists a bijection between $E_n(G_X)$ and $E_{X,n}(G)$.*

Example 3 illustrates several elements of the following proof.

*Proof:* Let $\varphi$ be the function from $(\Gamma \times \{0,1,2\} \cup \Sigma)^*$ into $(\Gamma \cup \Sigma)$ inductively defined by: $\varphi(\varepsilon) = \varepsilon$ and $\varphi(aw) = a\varphi(w)$ if $a \in \Sigma \cup \Gamma$ and $\varphi((Z,\alpha)w) = Z\varphi(w)$ if $Z \in \Gamma$ and $\alpha \in \{0,1,2\}$. Intuitively, $\varphi$ is a projection deleting the components in $\{0,1,2\}$.

By construction of $G_X$, if $(Z,\alpha) \to w$ is a rule of $G_X$ then $\varphi((Z,\alpha)) \to \varphi(w)$ is a rule of $G$. Therefore, if $x_0, \ldots, x_k$ is complete derivation of $G_X$, then $\varphi(x_0), \ldots, \varphi(x_k)$ is a complete derivation of $G$. Moreover, the initial symbol of $G_X$ is $(S,1)$ and all rules of $R_X$ with a left hand side in $(\Gamma \setminus \{X\}) \times \{1\}$ have a right hand side where an element of $\Gamma \times \{1\}$ occurs. Therefore, since $x_k \in \Sigma^*$, the only way to destroy the component 1 is to use a rule with the left hand side $(X,1)$. It follows that the derivation tree associated to $\varphi(x_0), \ldots, \varphi(x_k)$ covers $X$.

Consequently, $\varphi$ induces a function from $E_n(G_X)$ into $E_{X,n}(G)$. Let $x_0, \ldots, x_k$ and $x'_0, \ldots, x'_k$ be complete derivations of $G_X$, such that $\varphi(x_0), \ldots, \varphi(x_k) = \varphi(x'_0), \ldots, \varphi(x'_k)$. Assuming that $x_0, \ldots, x_k \neq x'_0, \ldots, x'_k$, there exists a minimal index $i_0$ such that $x_{i_0} \neq x'_{i_0}$. Since $x_0 = (S_0, 1) = x'_0$, $i_0 \geq 1$. Therefore $x_{i_0-1} = x'_{i_0-1}$ exists. Set $x_{i_0-1} = v_0(Z,\alpha)v_1$, with $Z \in \Gamma$ and $\alpha \in \{0,1,2\}$. One of the following cases arises:

- If $\alpha = 0$, then there exist $Z \to w$ and $Z \to w'$ in $R$ such that $x_{i_0} = v_0[w]_0v_1$ and $x'_{i_0} = v_0[w']_0v_1$. Since $\varphi(x_{i_0}) = \varphi(x'_{i_0})$, it follows that $\varphi([w]_0) = \varphi([w\prime]_0)$. But $\varphi([w]_0) = w$ and $\varphi([w']_0) = w'$, proving that $x_{i_0} = x'_{i_0}$, a contradiction.
- If $\alpha = 2$, then the same proof holds, replacing 0 by 2.
- If $\alpha = 1$ and $Z = X$, then, again, the same proof holds.
- If $\alpha = 1$ and $Z \neq X$, then there exist $Z \to w$ and $Z \to w'$ in $R$ such that $x_i = v_0w_1v_1$ and $x'_i = v_0w_2v_1$, with $w_1 \in \{w\}_{1,2}$ and $w_2 \in \{w'\}_{1,2}$. Since $\varphi(x_{i_0}) = \varphi(x'_{i_0})$, one has $w = w'$. Therefore, $w_1, w_2 \in \{w\}_{1,2}$. Since $w_1 \neq w_2$, let $j$ be the first letter of $w_1$ which is different from the corresponding letter in $w_2$. By construction of $\{w\}_{1,2}$, this letter must be in $\Gamma \times \{1,2\}$ in both $w_1$ and $w_2$, for instance $(T,\beta_1)$ and $(H,\beta_2)$. Now, since $\varphi((T,\beta_1)) = \varphi((H,\beta_2))$, one has $T = H$. Therefore, without loss of generality we may assume that $\beta_1 = 1$ and $\beta_2 = 2$. Consequently, $x_{i_0}$ has a prefix of the form $v_0(T,1)$: in the derivation tree corresponding to $x_0, \ldots, x_k$, the subtree rooted in this $(T,1)$ contains an $X$ (by construction of $R_1$). Conversely, $x'_{i_0}$ has a prefix of the form $v_0(T,2)$: in the derivation tree corresponding to $x'_0, \ldots, x'_k$, the subtree rooted in this $(T,2)$ does not contain any $X$ (by construction of $R_2$). It follows that the two corresponding derivations cannot have the same image by $\varphi$, a contradiction.

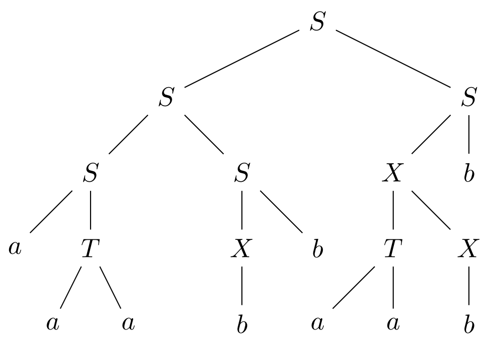

Fig. 2. Derivation tree of $G$ - Example 3

It follows that $\varphi$ induces an injective function from $E_n(G_X)$ into $E_{X,n}(G)$.

Now let $y_0, \ldots, y_k$ be complete derivations of $G$ whose corresponding tree $t$ is in $E_{X,n}(G)$. We consider the tree $t'$ labelled in $\Gamma \times \{0,1,2\} \cup \Sigma$ which has exactly the same structure (the same set of positions) than $t$ and such that:

- If a node of $t$ is labelled by a letter of $\Sigma$, then the corresponding node in $t'$ has the same label.
- If a node $\rho$ of $t$ is labelled by a letter $T \in \Gamma$, then the node $\rho$ in $t'$ is labelled by $(T,1)$ if there is no $X$ on the path from the root to $\rho$ (excluding $\rho$), and if the subtree rooted in $\rho$ (including $\rho$) contains one $\rho$, at least. It is labelled by $(T,0)$ if there is at least one $X$ on the path from the root to $\rho$. Otherwise, it is labelled by $(T,2)$.

One can check that $t'$ corresponds to a complete derivation tree of $G_X$ whose image by $\varphi$ is exactly the complete execution corresponding to $t$, proving that $\varphi$ is surjective, which concludes the proof. ■

**Example 3 – Illustration of the proof of Prop. 1.** *Consider the grammar $G = (\{a,b\}, \{S,T,X\}, S, R)$ with $R = \{S \to SS, S \to aT, S \to Xb, T \to aa, X, \to T, X \to b\}$ of Example 2. Consider the derivation tree of $E_{X,19}(G)$ depicted in Fig. 2, corresponding to the complete derivation $S, SS, SSS, aTSS, aaaSS, aaaXbS, aaabbS, aaabbXb, aaabbTXb, aaabbaaXb, aaabbaabb$. The associated derivation in $G_X$ is $(S,1), (S,1)(S,1), (S,2)(S,1)(S,1), a(T,2)(S,1)(S,1), aaa(S,1)(S,1), aaa(X,1)b(S,1), aaabb(S,1), aaabb(X,1)b, aaabb(T,0)(X,0)b, aaabbaa(X,0)b, aaabbaabb$, whose derivation tree from $E_{19}(G_X)$ is depicted in Fig. 3.*

Using Proposition 1 and the results described in Section I, it is possible to compute $|E_{X,n}(G)|$. If we denote by $\ell$ the maximal number of elements of $\Gamma$ (with multiplicity) occurring in a right hand side of $G$, then $G_X$ has $O(2^\ell|R|)$ rules, whose sizes are bounded by the maximal size of the rules of $G$. Therefore if $\ell$ is reasonable, the computation of $|E_{X,n}(G)|$ is tractable in practice, even for a quite large value of $n$. As mentioned above, the computation of $|E_{X,n}(G)|$ immediately provides $p_{X,n}$. It is also important to point out that $G_X$ allows

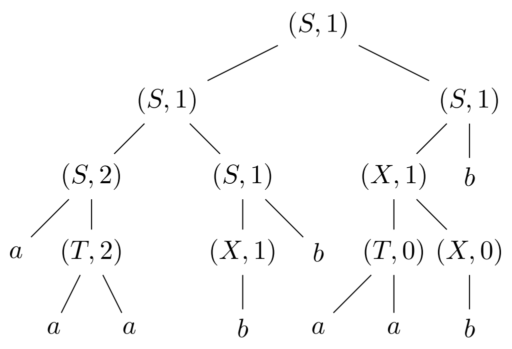

Fig. 3. Derivation tree of $G_X$ - Example 3

the uniform random computation of execution trees of $G$ of a given size and covering $X$.

Since $E_{X,X,n}(G) = E_{X,n}(G)$, computing $|E_{X,X,n}(G)|$ is a direct application of the above techniques. Computing $|E_{X,Y,n}(G)|$, with $Y \neq X$, can almost be done by a similar construction: the difference is that the construction of the rules of the grammar $G_{XY}$, from the grammar $G_X$, must take into account that both $X$ and $Y$ have to appear in the derivation. Let $G_{XY} = (\Sigma, \Gamma \times \{0,1,2\} \times \{0,1,2\}, ((S_0,1),1), R_{XY})$ where $R_{XY} = R_0 \cup R_1 \cup R'_1 \cup R_2$ with:

- $R_0 = \{((Z,i),0) \to [w]_0 \mid (Z,i) \to w \in R_X\}$,
- $R_1 = \{((Z,i),1) \to w' \mid Z \neq Y$ and $\exists (Z,i) \to w \in R_X$ such that $w' \in \{w\}_{1,2}\}$,
- $R'_1 = \{((Y,i),1) \to [w]_0 \mid (Y,i) \to w \in R_X\}$,
- $R_2 = \{((Z,i),2) \to [w]_2 \mid (Z,i) \to w \in R_X$ and $Z \neq Y\}$.

A proof similar to the one of Proposition 1 allows showing that there is a computable bijection between $E_n(G_{XY})$ and $E_{X,Y,n}(G)$. Note that the size of $G_{XY}$ is approximatively $4^{\ell}$ times greater than the size of $G$.

## V. Experiments

The approach has been evaluated on a simplified version of the grammar of JSON[2] (for JavaScript Object Notation) – a language independent common format for declaring objects. Formally, let us consider the grammar $G = (\Sigma, \Gamma, Object, R)$ with $\Sigma$ having the eight following elements $\Sigma = \{, , \{, :, \}, letter, digit, [, ]\}$. The set $\Gamma$ of non-terminal symbols[3] is composed of the elements $''Object''$, $''Members''$, $''Pair''$, $''Array''$, $''Elements''$ and $''Value''$. Finally, the set $R$ contains the following rules:

- $Object \to \{\} \mid \{Members\}$
- $Members \to Pair \mid Pair, Members$
- $Pair \to letter : Value$
- $Array \to [\,] \mid [Elements]$
- $Elements \to Value \mid Value, Elements$
- $Value \to letter \mid Object \mid digit \mid Array$

[2]http://www.json.org/

[3]To provide a more readable specification, the convention consisting in using capital letters for non-terminal symbols is not entirely respected here.

In order to optimise the coverage criterion, we have to solve the following system while maximising $p$ satisfying

$$\left\{\begin{array}{l}
p \leq \pi_{Object}\frac{p_{Object,Object,n}}{p_{Object,n}} + \pi_{Members}\frac{p_{Members,Object,n}}{p_{Members,n}} \\
\quad +\pi_{Pair}\frac{p_{Pair,Object,n}}{p_{Pair,n}} + \pi_{Array}\frac{p_{Array,Object,n}}{p_{Array,n}} \\
\quad +\pi_{Elements}\frac{p_{Elements,Object,n}}{p_{Elements,n}} + \pi_{Value}\frac{p_{Value,Object,n}}{p_{Value,n}} \\
p \leq \pi_{Object}\frac{p_{Object,Members,n}}{p_{Object,n}} + \pi_{Members}\frac{p_{Members,Members,n}}{p_{Members,n}} \\
\quad +\pi_{Pair}\frac{p_{Pair,Members,n}}{p_{Pair,n}} + \pi_{Array}\frac{p_{Array,Members,n}}{p_{Array,n}} \\
\quad +\pi_{Elements}\frac{p_{Elements,Members,n}}{p_{Elements,n}} + \pi_{Value}\frac{p_{Value,Members,n}}{p_{Value,n}} \\
p \leq \pi_{Object}\frac{p_{Object,Pair,n}}{p_{Object,n}} + \pi_{Members}\frac{p_{Members,Pair,n}}{p_{Members,n}} \\
\quad +\pi_{Pair}\frac{p_{Pair,Pair,n}}{p_{Pair,n}} + \pi_{Array}\frac{p_{Array,Pair,n}}{p_{Array,n}} \\
\quad +\pi_{Elements}\frac{p_{Elements,Pair,n}}{p_{Elements,n}} + \pi_{Value}\frac{p_{Value,Pair,n}}{p_{Value,n}} \\
p \leq \pi_{Object}\frac{p_{Object,Array,n}}{p_{Object,n}} + \pi_{Members}\frac{p_{Members,Array,n}}{p_{Members,n}} \\
\quad +\pi_{Pair}\frac{p_{Pair,Array,n}}{p_{Pair,n}} + \pi_{Array}\frac{p_{Array,Array,n}}{p_{Array,n}} \\
\quad +\pi_{Elements}\frac{p_{Elements,Array,n}}{p_{Elements,n}} + \pi_{Value}\frac{p_{Value,Array,n}}{p_{Value,n}} \\
p \leq \pi_{Object}\frac{p_{Object,Elements,n}}{p_{Object,n}} + \pi_{Members}\frac{p_{Members,Elements,n}}{p_{Members,n}} \\
\quad +\pi_{Pair}\frac{p_{Pair,Elements,n}}{p_{Pair,n}} + \pi_{Array}\frac{p_{Array,Elements,n}}{p_{Array,n}} \\
\quad +\pi_{Elements}\frac{p_{Elements,Elements,n}}{p_{Elements,n}} + \pi_{Value}\frac{p_{Value,Elements,n}}{p_{Value,n}} \\
p \leq \pi_{Object}\frac{p_{Object,Value,n}}{p_{Object,n}} + \pi_{Members}\frac{p_{Members,Value,n}}{p_{Members,n}} \\
\quad +\pi_{Pair}\frac{p_{Pair,Value,n}}{p_{Pair,n}} + \pi_{Array}\frac{p_{Array,Value,n}}{p_{Array,n}} \\
\quad +\pi_{Elements}\frac{p_{Elements,Value,n}}{p_{Elements,n}} + \pi_{Value}\frac{p_{Value,Value,n}}{p_{Value,n}} \\
\pi_{Object} + \pi_{Members} + \pi_{Pair} + \pi_{Array} + \pi_{Elements} + \pi_{Value} = 1
\end{array}\right.$$

Using a slightly modified version of the Hoa tool ([25]), the computation of the probabilities $p_{X,n}$ and $p_{X,Y,n}$ for all $X, Y \in \Gamma$ and $n = 20$ has been performed efficiently (a few seconds). The system becomes as below, and we have then to solve it while maximising $p$ satisfying

$$\left\{\begin{array}{l}
p \leq \pi_{Object}\frac{12}{12} + \pi_{Members}\frac{12}{12} + \pi_{Pair}\frac{12}{12} + \pi_{Array}\frac{11}{11} \\
\quad +\pi_{Elements}\frac{8}{8} + \pi_{Value}\frac{12}{12} \\
p \leq \pi_{Object}\frac{12}{12} + \pi_{Members}\frac{12}{12} + \pi_{Pair}\frac{12}{12} + \pi_{Array}\frac{11}{11} \\
\quad +\pi_{Elements}\frac{8}{8} + \pi_{Value}\frac{12}{12} \\
p \leq \pi_{Object}\frac{12}{12} + \pi_{Members}\frac{12}{12} + \pi_{Pair}\frac{12}{12} + \pi_{Array}\frac{11}{11} \\
\quad +\pi_{Elements}\frac{8}{8} + \pi_{Value}\frac{12}{12} \\
p \leq \pi_{Object}\frac{11}{12} + \pi_{Members}\frac{11}{12} + \pi_{Pair}\frac{11}{12} + \pi_{Array}\frac{11}{11} \\
\quad +\pi_{Elements}\frac{8}{8} + \pi_{Value}\frac{11}{12} \\
p \leq \pi_{Object}\frac{8}{12} + \pi_{Members}\frac{8}{12} + \pi_{Pair}\frac{8}{12} + \pi_{Array}\frac{8}{11} \\
\quad +\pi_{Elements}\frac{8}{8} + \pi_{Value}\frac{8}{12} \\
p \leq \pi_{Object}\frac{12}{12} + \pi_{Members}\frac{12}{12} + \pi_{Pair}\frac{12}{12} + \pi_{Array}\frac{11}{11} \\
\quad +\pi_{Elements}\frac{8}{8} + \pi_{Value}\frac{12}{12} \\
\pi_{Object} + \pi_{Members} + \pi_{Pair} + \pi_{Array} \\
\quad +\pi_{Elements} + \pi_{Value} = 1
\end{array}\right.$$

This linear programming problem can be solved in an efficient way, using simplex-like approaches. We have used the tool lp_solve[4] to solve it, and the result is that $p = 1$ if $\pi_{Object} = 0$, $\pi_{Members} = 0$, $\pi_{Pair} = 0$, $\pi_{Array} = 0$, $\pi_{Elements} = 1$, and $\pi_{Value} = 0$. It means that, for this simple example, the optimised approach to cover all the non-terminals symbols, consists in generating derivation trees covering $Elements$. Indeed, in this grammar, the generation of a derivation tree covering the non-terminal symbol $Elements$ provides a tree covering all the other non-terminal symbols.

## VI. Conclusion

In this paper, we have presented a method for exploiting a coverage criterion together with random testing in the context of grammar-based testing. This automatic method lies in building a grammar and then in resolving a linear constraint system, which can be done by adapted tools, even for large values. In the future, we plan to extend the approach to other coverage criteria such as rules coverage, and also to handle attribute grammars with constraints formalising the semantics of context-free languages.

## References

[1] A. Denise, M.-C. Gaudel, S.-D. Gouraud, R. Lassaigne, J. Oudinet, and S. Peyronnet, "Coverage-biased random exploration of large models and application to testing," *STTT*, vol. 14, no. 1, pp. 73–93, 2012.

[2] P. Purdom, "A sentence generator for testing parsers," *BIT*, vol. 12, no. 3, pp. 366–375, 1972.

[3] B. Daniel, D. Dig, K. Garcia, and D. Marinov, "Automated testing of refactoring engines," in *ESEC/FSE 2007: Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. New York, NY, USA: ACM Press, September 2007.

[4] D. Coppit and J. Lian, "Yagg: an easy-to-use generator for structured test inputs," in *ASE*, D. F. Redmiles, T. Ellman, and A. Zisman, Eds. ACM, 2005, pp. 356–359.

[5] R. Lämmel and W. Schulte, "Controllable combinatorial coverage in grammar-based testing," in *TestCom*, ser. LNCS, M. Uyar, A. Duale, and M. Fecko, Eds., vol. 3964. Springer, 2006, pp. 19–38.

[6] R. Majumdar and R.-G. Xu, "Directed test generation using symbolic grammars," in *ASE*, R. E. K. Stirewalt, A. Egyed, and B. F. 0002, Eds. ACM, 2007, pp. 134–143.

[7] P. Godefroid, A. Kiezun, and M. Levin, "Grammar-based whitebox fuzzing," in *PLDI*, R. Gupta and S. P. Amarasinghe, Eds. ACM, 2008, pp. 206–215.

[8] Z. Xu, L. Zheng, and H. Chen, "A toolkit for generating sentences from context-free grammars." in *Software Engineering and Formal Methods*, ser. IEEE, 2010, pp. 118–122.

[9] R. Lämmel, "Grammar testing," in *FASE*, ser. Lecture Notes in Computer Science, H. Hußmann, Ed., vol. 2029. Springer, 2001, pp. 201–216.

[10] L. Zheng and D. Wu, "A sentence generation algorithm for testing grammars," in *COMPSAC (1)*, S. Ahamed, E. Bertino, C. Chang, V. Getov, L. Liu, H. Ming, and R. Subramanyan, Eds. IEEE Computer Society, 2009, pp. 130–135.

[11] T. Alves and J. Visser, "A case study in grammar engineering," in *SLE*, ser. Lecture Notes in Computer Science, D. Gasevic, R. Lämmel, and E. V. Wyk, Eds., vol. 5452. Springer, 2008, pp. 285–304.

[12] J. Duran and S. Ntafos, "A report on random testing," in *ICSE '81: Proceedings of the 5th international conference on Software engineering*. Piscataway, NJ, USA: IEEE Press, 1981, pp. 179–183.

[13] R. Hamlet, "Random testing," in *Encyclopedia of Software Engineering*. Wiley, 1994, pp. 970–978.

[14] P. Godefroid, N. Klarlund, and K. Sen, "DART: directed automated random testing," in *PLDI'05: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*. New York, NY, USA: ACM, 2005, pp. 213–223.

[15] C. Oriat, "Jartege: A tool for random generation of unit tests for java classes," in *QoSA/SOQUA*, ser. Lecture Notes in Computer Science, R. Reussner, J. Mayer, J. Stafford, S. Overhage, S. Becker, and P. Schroeder, Eds., vol. 3712. Springer, 2005, pp. 242–256.

[16] B. McKenzie, "Generating string at random from a context-free grammar," Univ ersity of Canterbury, Tech. Rep. TR-COSC 10/97, 1997.

[17] T. J. Hickey and J. Cohen, "Uniform random generation of strings in a context-free language," *SIAM J. Comput.*, vol. 12, no. 4, pp. 645–655, 1983.

[18] P. Maurer, "The design and implementation of a grammar-based data generator," *Softw., Pract. Exper.*, vol. 22, no. 3, pp. 223–244, 1992.

[19] P.-C. Héam and C. Nicaud, "Seed: An easy-to-use random generator of recursive data structures for testing," in *ICST*. IEEE Computer Society, 2011, pp. 60–69.

[20] F. Dadeau, J. Levrey, and P.-C. Héam, "On the use of uniform random generation of automata for testing," *Electr. Notes Theor. Comput. Sci.*, vol. 253, no. 2, pp. 37–51, 2009.

[21] P.-C. Héam and C. Masson, "A random testing approach using pushdown automata," in *TAP*, ser. Lecture Notes in Computer Science, M. Gogolla and B. Wolff, Eds., vol. 6706. Springer, 2011, pp. 119–133.

[22] I. Enderlin, F. Dadeau, A. Giorgetti, and F. Bouquet, "Grammar-based testing using realistic domains in php," in *ICST*, G. Antoniol, A. Bertolino, and Y. Labiche, Eds. IEEE, 2012, pp. 509–518.

[23] P. Flajolet and R. Sedgewick, *Analytic Combinatorics*. Cambridge University Press, 2008.

[24] A. Denise and P. Zimmermann, "Uniform random generation of decomposable structures using floating-point arithmetic," *Theor. Comput. Sci.*, vol. 218, no. 2, pp. 233–248, 1999.

[25] I. Enderlin, "Hoa project, a set of php libraries." *URL: http://hoa-project.net*.

[4]http://lpsolve.sourceforge.net/