Int J Software Informatics, Vol.5, No.XX, XXXXXX 2011, pp. 659–676 E-mail: ijsi@iscas.ac.cn
International Journal of Software and Informatics, ISSN 1673-7288 http://www.ijsi.org
 Tel: +86-10-62661040

A Toolkit for Generating Sentences from Context-Free Grammars

Zhiwu Xu[1,2], Lixiao Zheng[1,2], and Haiming Chen[1]

[1]State Key Laboratory of Computer Science,

Institute of Software, Chinese Academy of Sciences, China

[2]Graduate University, Chinese Academy of Sciences

Abstract Producing sentences from a grammar, according to various criteria, is required in many applications. It is also a basic building block for grammar engineering. This paper presents a toolkit for context-free grammars, which mainly consists of several algorithms for sentence generation or enumeration and for coverage analysis for context-free grammars. The toolkit deals with general context-free grammars. Besides providing implementations of algorithms, the toolkit also provides a simple graphical user interface, through which the user can use the toolkit directly. The toolkit is implemented in `Java` and is available at `http://lcs.ios.ac.cn/~zhiwu/toolkit.php`. In the paper, the overview of the toolkit and the major algorithms implemented in the toolkit are presented, and experimental results and preliminary applications of the toolkit are also contained.

Key words: context-free grammars; sentence generation; sentence enumeration; coverage criteria; toolkit

1 Introduction

Grammars, especially context-free grammars, are fundamental structures in computer science. Producing sentences from a grammar, according to various coverage criteria or some other constraints, is required in many applications, such as parser/compiler testing, natural language processing, grammar validation, test case generation, bioinformatics, etc. Recently, grammar engineering has been recognized as an emerging field of software engineering due to the fact of lacking solid engineering methods and techniques for grammars on one hand, and due to the importance of grammars in computer science and in other areas on the other hand [1]. Techniques for generating sentences form a basic building block for grammar engineering [2].

Thus a toolkit package for generating sentences is useful. However, presently only very limited abilities to generate sentences are provided in a few grammar tools [3, 4]. On the other hand, there are research issues on sentence generation or enumeration algorithms that need to explore. Based on the research on the algorithms by the authors and others, this paper presents a toolkit for context-free grammars, which mainly consists of several algorithms for sentence generation or enumeration and for coverage analysis for context-free grammars.

The commonly used algorithm for sentence generation from context-free grammars

† This work is sponsored by the National Natural Science Foundation of China under Grant Nos. 61070038, 60573013.
‡ Corresponding author: Haiming Chen, Email: chm@ios.ac.cn
♯ Manuscript received 2011-03-04; revised 2011-04-12; accepted 2011-04-29

is the one proposed by Purdom [5], which produces a small set of sentences satisfying rule coverage. We found out through analysis and experiments that in most cases, Purdom's algorithm produces too few sentences and the sentences differ too much in length, i.e., some of them are much longer and more complex in structure than others. These sentences are inadequate in many practical applications. To avoid this, an extended algorithm [6] is proposed and implemented which generates more and simpler sentences. A more precise criteria is context-dependent rule coverage [2]. However, as noted in Ref. [7], implementation and computation of this metric are more involved. A sentence generation algorithm fulfilling context-dependent rule coverage based on Purdom's algorithm is proposed and implemented [8], which, to the best of the authors' knowledge, is the first algorithm for context-dependent rule coverage. An extended algorithm, similar to the idea of the above improvement of Purdom's algorithm, is also proposed and implemented in the toolkit.

The toolkit contains algorithms [9, 10] for sentence and parse tree enumeration for context-free grammars, which are the first linear algorithms and answer the open problem [11] about the existence of such algorithms. The algorithms also lead to several methods of sentence generation for context-free grammars, generally called user-controlled methods in the paper, such as random generation, bounded range enumeration, and structure-sensitive generation of sentences.

The toolkit deals with general context-free grammars, which have no restrictions on grammars, and is implemented in Java as Java classes. Besides providing implementations of algorithms, it also provides a simple graphical user interface, through which the user can use the toolkit directly.

The algorithms implemented in the toolkit have been tested through experiments. Some of the experimental results are presented in the paper. The toolkit has also been used in some preliminary applications, which are briefly introduced in the paper.

This paper is the full version of our previous publication with the same title at SEFM'10 [12], the main additional materials being the descriptions of the algorithms contained in the toolkit (Section 4, Section 5 and Section 6), the implementation of the toolkit and some of the experimental results (Section 7), and an additional application (Subtype Checker in Section 8).

The rest of this paper is organized as follows. In Section 2, we review the basic concepts and notations of context-free grammars. An overview of the toolkit is given in Section 3. Section 4 and Section 5 present the sentence generation and enumeration algorithms contained in the toolkit respectively. Section 6 introduces sentence analysis algorithms contained in the toolkit. The implementation of the toolkit, the description of the graphical user interface and some of the experimental results are presented in Section 7. Section 8 gives some applications that use our toolkit. We discuss some related work in Section 9. Finally, Section 10 concludes.

2 Context-Free Grammar

This section reviews the basic concepts and notations of context-free grammars, most of which refer to Ref. [13].

A *context-free grammar* is a tuple $G = \langle N, T, P, S\rangle$, where N is a finite set of *nonterminals*, T a finite set of *terminals* ($N \cap T = \emptyset$),$P \subseteq N \times (N \cup T)^*$ a finite set

of *rules* and $S \in N$ the *start symbol*. In the remaining, assume that X, Y range over N, u, v and w over T^*, and $\alpha, \beta, \gamma, \delta$ and η over $(N \cup T)^*$.

We call X the left-hand side and η the right-hand side for a rule $(X, \eta) \in P$. For convenience, we write $X \to \eta$ for a rule $(X, \eta) \in P$, and $X \to \eta_1 | \cdots | \eta_n$ for several rules with the same left-hand side. Moreover, we call a rule p with at least one nonterminal at the right-hand side a nonterminal rule, while a rule without nonterminals at the right-hand side a terminal one.

We write $\alpha \Rightarrow \beta$ if there exist γ, δ, η and X such that $\alpha = \gamma X \delta$, $\beta = \gamma \eta \delta$, and $X \to \eta \in P$. We write $\alpha \overset{*}{\Rightarrow} \beta$ if there exist $\alpha_0, \ldots, \alpha_n (n \geq 0)$ such that $\alpha = \alpha_0 \Rightarrow \ldots \Rightarrow \alpha_n = \beta$. If $S \overset{*}{\Rightarrow} w$, we say w is a *sentence* of G. Thus, the *context-free language* defined by G, denoted by $\mathcal{L}(G)$, is composed of all sentences of G. A *parse tree* of a sentence w is a tree presentation of a derivation $S \overset{*}{\Rightarrow} w$ whose root is S, leaf nodes are terminals and interior nodes are nonterminals.

In the paper, *height* of a parse tree is defined as the maximum path length which is the number of nonterminals in a path traversed from the root to the leaf node, and *length* of a sentence w is defined to be the number of nodes in its parse tree [5], representing the derivation complexity of w. Let W be a sentence set. The *volume data* $|W|$ is defined to be the number of sentences contained in W, and $||W||$ denotes the maximal sentence length in W.

A nonterminal X is said to be *useless* if it generates no sentences, or *unreachable* if there is not a derivation $S \overset{*}{\Rightarrow} \alpha X \beta$. A context-free grammar is said to be *well-defined*, if it has neither useless nor unreachable nonterminals.

A sentence $w \in \mathcal{L}(G)$ is said to cover a rule $p = X \to \eta \in P$ if there is a derivation $S \overset{*}{\Rightarrow} \alpha X \beta \overset{p}{\Rightarrow} \alpha \eta \beta \overset{*}{\Rightarrow} w$. A sentence set $W \subseteq \mathcal{L}(G)$ is said to achieve *rule coverage* [5] for G, if for each $p \in P$ there is a $w \in W$ which covers p.

If $Y \to \alpha X \beta \in P$, then $Y \to \alpha \boxed{\text{X}} \beta$ is called a direct *occurrence* of X in G. A sentence $w \in \mathcal{L}(G)$ is said to cover a rule $p = X \to \eta \in P$ for the occurrence $Y \to \alpha \boxed{\text{X}} \beta$ if there is a derivation $S \overset{*}{\Rightarrow} \gamma Y \delta \overset{q}{\Rightarrow} \gamma \alpha X \beta \delta \overset{p}{\Rightarrow} \gamma \alpha \eta \beta \delta \overset{*}{\Rightarrow} w$ with $q = Y \to \alpha X \beta \in P$. A sentence set $W \subseteq \mathcal{L}(G)$ is said to achieve *context-dependent rule coverage* [2] for G, if all $p \in P$ for all occurrences are covered.

Let $N = \{X_1, \ldots, X_m\}$. We refer to $\mathcal{L}_i$ as the language of $G_i = \langle N, T, P, X_i \rangle$, where $i = 1, \ldots, m$. If there exists a derivation $X \overset{*}{\Rightarrow} \alpha Y \beta \overset{*}{\Rightarrow} \alpha v \beta \overset{*}{\Rightarrow} w$, we say v is a *sub-sentence* of w. If $X \overset{p}{\Rightarrow} u_0 X_1 u_1 \ldots X_r u_r \overset{*}{\Rightarrow} u_0 v_1 u_1 \ldots v_r u_r = w$, where r is the number of nonterminals at the right-hand side of rule p, we say v_i is the i-th (from left to right) *direct sub-sentence* of w.

If $X \overset{*}{\Rightarrow} \alpha Y \beta$, we say that Y is *successor* of X and write $X \triangleright^* Y$. The successor relation induces an equivalence relation on the nonterminals and partitions them into a collection of equivalence classes, known as *grammatical levels*. For different levels L_1 and L_2, if there exists $X \in L_1$ and $Y \in L_2$ such that $X \triangleright^* Y$, we write $L_1 \succ L_2$. Based on these definitions, there are five *complexity measures* [14], including size measures and structural complexity measures, for a context-free grammar:

VAR	The number of nonterminals
PROD	The number of rules
LEV	The number of grammatical levels
DEP	The number of nonterminals in the largest grammatical level
HEI	The maximum length of the chain of levels $L_0 \dots L_n$ such that $L_i \succ L_{i+1}$ holds for $0 \leq i < n$

3 The Toolkit: an Overview

In this section, we give an overview of our toolkit. The toolkit contains the following functionalities and algorithms:

- Manipulations of context-free grammars
  - Editing of context-free grammars
  - Checking the usefulness and reachability of nonterminals
- Sentence generation with coverage criteria
  - Purdom's algorithm [5]
  - An extension of Purdom's algorithm [6]
  - CDRC-P algorithm [8]
  - An extension of CDRC-P algorithm
- Sentence enumeration
  - Dong's algorithms [9, 10]
  - Sentence generation methods based on sentence enumeration
- Sentence analysis
  - Earley's algorithm [15]
  - Coverage analysis algorithms
- Graphical user interface (GUI)
  - Sentence generation panel
  - Sentence enumeration panel
  - Sentence analysis panel

The toolkit provides an editor for the user to input and edit a context-free grammar. The editor can check whether the grammar is well-defined. The toolkit focuses on sentence generation, sentence enumeration and coverage analysis, which constitute the main functionalities of the toolkit. The introductions of the correlative algorithms are presented in the next three sections. The toolkit also contains a simple graphical user interface, which consists of three panels corresponding to the three main functionalities and through which the user can use the toolkit directly. The toolkit is available at `http://lcs.ios.ac.cn/~zhiwu/toolkit.php`.

To illustrate the algorithms and to describe the graphical user interface, we use the following grammar as the sample grammar throughout the paper. We use angle brackets to distinguish nonterminals from terminals.

```
<S> -> <E>                    <E> -> <E> + <T> | <T>
<T> -> <T> * <F> | <F>        <F> -> id | (<E>)
```

4 Sentence Generation

The sentence generation algorithms provided in our toolkit are based on grammar coverage criteria, including rule coverage (RC) [5] and context-dependent rule coverage (CDRC) [2]. For each coverage criterion, two algorithms are implemented, one with a sentence length control mechanism and the other without.

4.1 Sentence Generation with Rule Coverage

The notion of rule coverage as a coverage criterion for context-free grammars was introduced by Purdom [5]. Rule coverage simply means that a sentence set explores all the rules of a grammar. Note that the sentence set satisfying rule coverage for a given grammar is not unique.

4.1.1 Purdom's Algorithm

Purdom described a fast algorithm for automatically generating a sentence set that achieves rule coverage for a context-free grammar [5]. Purdom's algorithm takes a context-free grammar as input and produces a small set of sentences such that each rule of the grammar is used at least once. The algorithm proceeds in two distinct phases.

The first phase statically collects necessary information from the grammar and stores them in some tables. The information includes: length of the shortest sentence that can be derived from each nonterminal and each rule, length of the shortest sentence which uses a nonterminal X in its derivation, which rule to use to derive the shortest sentence from a nonterminal X, which rule to use to introduce a nonterminal X into the shortest derivation, and so on.

The second phase dynamically generates sentences by utilizing the information collected in the first phase. A table known as ONCE calculates the next rule to be used for each nonterminal. The algorithm terminates when all the grammar rules have been exploited. Readers are referred to Ref. [5, 16] for a complete interpretation of this algorithm.

Example 4.1. The sentence set generated by Purdom's algorithm for the sample grammar is $\{id * (id) + id\}$, containing one sentence with length 19.

4.1.2 The Extension of Purdom's Algorithm

We found out through analysis and experiments that in most cases, Purdom's algorithm produces too few sentences and some of them are long and complex, i.e., with complicated derivation structures. These sentences are inadequate in many practical applications. For example, in applications to grammar testing or test case generations for the systems based on grammars, such sentences may not be helpful for error location and debugging purposes [6, 17].

To avoid the shortcoming of Purdom's algorithm, Ref. [6] proposed an improved algorithm which still accomplishes the rule coverage goal but generates more and simpler sentences. The algorithm builds upon Purdom's with two main extensions.

First, a *reference length* is used in the sentence generation process as a reference to control the length of the generated sentence. The reference length for a given grammar G, denoted as $\mathcal{RL}(G)$, is selected to be the minimal $||W||$ for all $W \subseteq \mathcal{L}(G)$ that achieves rule coverage for G.

Second, the length of the shortest sentence derivable from the current derivation is forecasted in the sentence generation process. When choosing a rule to use, it compares this shortest length with the reference length and takes corresponding *length control* strategies. Interested readers are referred to Ref. [6] for a detailed description of this algorithm and its comparison with Purdom's original algorithm.

Example 4.2. The sentence set generated by the extension of Purdom's algorithm for the sample grammar is $\{id + id;\ (id) * id\}$, containing two sentences with lengths 10 and 14 respectively.

4.2 Sentence Generation with Context-Dependent Rule Coverage

Rule coverage explores a grammar's structure in a relatively weak sense since it considers each rule independently. To achieve more preciseness, Lämmel proposed a generalization of rule coverage such that the context in which a rule is covered is also taken into account. This is known as *context-dependent rule coverage* [2].

In our toolkit we provide two algorithms to achieve this coverage. One is based on Purdom's algorithm, called CDRC-P algorithm [8], and the other extends CDRC-P algorithm with a length control mechanism.

4.2.1 CDRC-P Algorithm

[8] modified Purdom's algorithm by changing table ONCE from calculating the next rule to use for each nonterminal to calculating the next rule to use for each direct occurrence of each nonterminal. In the sentence generation process, it records the context (occurrence) of each encountered nonterminal and then consults table ONCE to choose the right rule to rewrite that nonterminal. When all the rules for all the occurrences in the grammar have been covered, the generation process ceases. The sentences generated by this algorithm are in general more complex than those of Purdom's.

Example 4.3. The sentence set generated by CDRC-P algorithm for the sample grammar is $\{id * id * ((id) + id * id) + id + id;\ id\}$, containing two sentences with lengths 42 and 5 respectively.

4.2.2 The Extension of CDRC-P Algorithm

As a modified version of Purdom's algorithm, CDRC-P preserves most of the features of Purdom's algorithm, i.e., the generated sentences are relatively few and some of them may be rather long and complicated (see example 4.3). Therefore we adopted the same *length control* idea as in the extension of Purdom's algorithm to CDRC-P algorithm so that more and simpler sentences achieving context-dependent rule coverage will be produced. But the reference length is selected to be the minimal $||W||$ for all $W \subseteq \mathcal{L}(G)$ that achieves context-dependent rule coverage for G.

Example 4.4. The sentence set generated by the extension of CDRC-P algorithm for the sample grammar is $\{id+id+id;\ (id)*id*id;\ (id)*id+id;\ (id+id)*id;\ id*(id)+id;\ id;\ id+id*id\}$, containing seven sentences with lengths 15, 18, 19, 19, 19, 5 and 14 respectively.

5 Sentence Enumeration

The sentence enumeration algorithm provided in our toolkit is the one proposed by Dong [9, 10], which is the first linear algorithm for general context-free grammars and answers the open problem [11] about the existence of such algorithm. Dong's algorithm enumerates the sentences of a general context-free grammar in hierarchical lexicographic order, and can be easily extended to enumerate the parse trees of a general context-free grammar [10], which is also contained in the toolkit.

5.1 Introduction to Dong's Algorithm

According to the height of the parse tree, the set of sentences of a context-free grammar is partitioned into (in)finite hierarchies, such that sentences of the same height belong to the same hierarchy. The 0-th hierarchy consists of sentences of height 1, which is determined by the grammar directly. The n-th ($n > 0$) hierarchy consists of sentences of height $n+1$, which is constructed by utilizing the previous well-constructed hierarchies mechanically.

5.1.1 Hierarchical Lexicographic Order

To partition the sentences into hierarchies, Dong's algorithm constructs four different kinds of sentence sets:

- $heap_n^i$, containing the sentences in $\mathcal{L}_i$ whose height is no more than $n+1$,
- $hierarchy_n^i$, containing the sentences of height $n+1$ in $\mathcal{L}_i$,
- $cluster_n^p$, containing the sentences of height $n+1$ using the nonterminal rule p as the first one in the derivation,
- $cube_n^{p,t}$, containing the sentences in $cluster_n^p$ whose t-th (from right to left) direct sub-sentence is from the $(n-1)$-th hierarchy.

Corresponding to the four kinds of sentence sets, there are four kinds of volume data: heap volume data, hierarchy volume data, cluster volume data and cube volume data. Consider the sample grammar. Since there are no terminal rules (sentences of height 1) for nonterminals S, E and T, all their hierarchy volume data on the 0-th hierarchy equal to 0. While there is one and only one terminal rule for F, so the hierarchy volume data $|hierarchy_0^F|$ is 1.

Apparently, the hierarchies in $\mathcal{L}_i$ are disjoint finite sets, and every hierarchy (except the 0-th hierarchy) consists of some disjoint clusters and every cluster consists of some disjoint cubes. Thus, the set of sentences can be sorted either set (i.e., hierarchy, cluster and cube) by set or by lexicographic order. The order relation is called *hierarchical lexicographic order*.

5.1.2 Enumeration Procedures

The most important enumeration procedure is the `next` procedure, which generates the next sentence of a given sentence. There are four possible different cases for a given sentence and its next sentence:

- they are in different hierarchies,
- they are in the same hierarchy but different clusters,
- they are in the same cluster but different cubes,
- they are in the same cube.

Two modes are presented for this procedure: bottom-up or top-down. The top-down mode runs from the different hierarchies case to the same cube case, while the bottom-up one does reversely. The time complexities of them are both $O(n)$. But in practice, the time needed by the bottom-up one is usually less than that needed by the top-down one, as the same cube case occurs more frequently.

The `previous` procedure, which generates the previous sentence of a given sentence, is similar to the `next` one and we do not discuss it here.

There are two procedures for the conversion between sentences and natural numbers, namely `N2L` and `L2N` respectively. The `N2L` procedure converts a natural number to the corresponding sentence, while the `L2N` procedure does the opposite. Note that when computing a natural number corresponding to a given sentence, it needs to ensure that the sentence belongs to the grammar.

5.2 Sentence Generation

Dong's algorithm can be used to implement several methods of sentence generation, generally called user-controlled methods in the paper, such as random generation, bounded range enumeration, and structure-sensitive generation of sentences. With a random number generator, Dong's algorithm can generate sentences randomly by using `N2L`. Given a bounded range indicated by, for example, two numbers, Dong's algorithm can enumerate all the sentences in this range. Hierarchy, cluster, and cube can be used to specify restrictions on sentence generation as well, which is called structure-sensitive generation. For instance, the sentences generated from some cube have a similar structure. The structure-sensitive generation and the two sentence generations above are orthogonal. This indicates that we can generate some random sentences or a bounded range of sentences in some hierarchy, cluster or cube.

Further, Dong's algorithm can also be used for sentence generation with coverage criteria. To cover a rule p, it needs to generate such a sentence that contains a sub-sentence from the cluster $cluster_n^p$. Similarly, to exploit a rule $p = X \to \eta$ for an occurrence $q = Y \to \alpha \boxed{\text{X}} \beta$, it needs to generate a sentence containing a sub-sentence from those sentences, which are in the cluster $cluster_n^q$ and whose t-th direct sub-sentence (assume that X is the t-th nonterminal at the right-hand side of rule q) is from the cluster $cluster_m^p$ instead of the hierarchy $hierarchy_{n-1}^X$ or the heap $heap_{n-1}^X$ (or $heap_{n-2}^X$). However, we found out through analysis and experiments that it is time-consuming to generate a sentence set that satisfies rule coverage or context-dependent rule coverage by enumeration starting from the first sentence, as it produces a lot of sentences that repeatedly cover already-used rules. It likely needs to collect some

effective information as that used in Purdom's algorithm, such as which rule to use to introduce a nonterminal into the shortest derivation. We will consider this problem in the future.

6 Sentence Analysis

There are several parsing algorithms for determining whether a sentence belongs to a grammar. As our toolkit deals with general context-free grammars, we use Earley's algorithm [15] for sentence parsing. Earley's algorithm is an efficient parsing algorithm for general context-free grammars and is implemented in our toolkit as follows: given a grammar and a sentence, if the sentence belongs to the grammar, the toolkit extracts all possible parses of the sentence. This is useful for checking sentence ambiguity.

Based on Earley's algorithm, the coverage of a sentence set, including rule coverage and context-dependent rule coverage, can be analyzed. The coverage analysis algorithm parses each sentence in a sentence set using Earley's algorithm, and marks all the exploited rules and occurrences used in the derivation of the sentence. Once some sentence is rejected by the grammar, the algorithm exits by marking the sentence unaccepted. When the parsing of the sentence set ends, the algorithm checks whether all the rules and occurrences have been exploited. If a sentence set does not satisfy rule coverage, it would not satisfy context-dependent rule coverage either. In addition, if the given grammar is ambiguous, the algorithm picks only one of the derivations of a sentence for coverage analysis. Therefore, in this case, a sentence set generated according to some criterion may be stated not to satisfy the same criterion.

7 Implementation and Experiments

The toolkit is implemented entirely in Java. The architecture of the toolkit is shown in Figure 1, where solid rectangles represent modules, while arrows represent dependency relations between modules. The architecture of the toolkit consists of six main modules, each of which is implemented as a java package. The parser module parses a input text into a grammar, and then passes this grammar to the cfg module. The cfg module checks whether this grammar is a context-free grammar, and if it is, whether it is well-defined, and so on. The cfg module is a basic one, which contains a sub-module grammar to deal with a general grammar and is used by most of the other modules. Once a well-defined context-free grammar is ready, the generation module, which implements the algorithms for sentence generation, can be used for sentence generation with coverage criteria. Similar to the generation module, the enumeration module implements the algorithms for sentence generation based on sentence enumeration, while the analysis module implements the algorithms for sentence analysis. These three modules form the toolkit's main functionalities. Finally, the gui module implements a simple graphical user interface for the user to use the toolkit directly, which consists of three separate sub-modules corresponding to the three main functionalities and is described in Section 7.1. As the toolkit is implemented as several modules, it is easy for us (and the user) to extend with more algorithms and functionalities.

The toolkit is distributed both as a stand-alone tool, which can be used directly through the graphical user interface, and as a library, containing the cfg, generation, enumeration and analysis modules (the dotted rectangle in Figure 1). By adding the

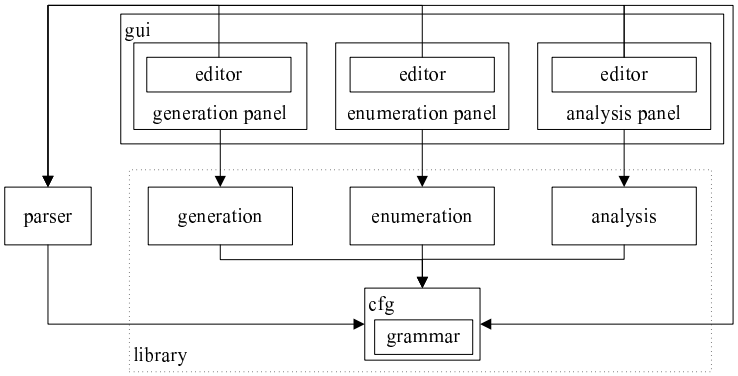

Fig. 1. Architecture of toolkit

toolkit into the classpath, the user can invoke the functions and classes implemented in the toolkit in his own programs. The javadoc API specification is available online as well.

In addition, two implementations are provided for Dong's algorithm: one with the specific volume data and the other without. There are essentially no differences between the two implementations except the type of volume data. The first one uses the *BigInteger* type to compute and store the specific volume data, while the other one uses the *Boolean* type to denote the emptiness of the volume data. Clearly, the first one needs more time and space than the other in practice. Both implementations can be used to compute sentences one by one and to enumerate sentences in order. While with the specific volume data, the conversion between sentences and natural numbers and the top-down version of `next` procedure can be implemented. Thus, we use the one with the specific volume data in the graphical user interface to provide more functionalities, namely, to query the volume data for an arbitrary hierarchy, to do operations about a sentence, such as the generation of the next or previous sentence for a given sentence and the conversion between sentences and natural numbers, to enumerate a bounded range of sentences, which can also be restricted in some hierarchy, some cluster or some cube, and so on.

7.1 Graphical User Interface

The interface consists of three panels, which correspond to the three main functionalities: sentence generation, sentence enumeration and sentence analysis. Each panel is implemented independently as a module, so that they can be used independently.

7.1.1 Editor and Console

Each panel contains a grammar editor which enables the user to edit, load or save a grammar and to check whether a grammar is well-defined, and an output console which outputs the outcomes and extra information like sentence length, running time and so on.

The grammar rules are written as the following form: $< X > -> \eta_1 | \ldots | \eta_n$, where nonterminals are bracketed by angle brackets to be distinguished from terminals. Once a grammar is input, the editor will check whether it is well-defined. By default, the

Fig. 2. Sentence generation with rule coverage and length control

start symbol corresponds to the left-hand side of the first rule. It can be also indicated by the user through the "Start" dropdown box.

7.1.2 Sentence Generation Panel

The sentence generation panel allows the user to generate a sentence set that satisfies rule coverage or context-dependent rule coverage. A length control mechanism can as well be configured on in the sentence generation process by marking the "Length Control" check box. After generation, the number of generated sentences, the information of sentences' lengths, and the total time for generating sentences are also counted. For instance, we generate a sentence set that satisfies rule coverage from the sample grammar with the length control. The result is shown in Figure 2.

7.1.3 Sentence Enumeration Panel

The sentence enumeration panel consists of three tabs. Figure 3 shows a general view of the panel.

The first tab enables the user to query the volume data among a bounded hierarchical range by giving the "From" hierarchy index and the "To" hierarchy index. It also allows the user to choose which kinds of volume data to query by marking the corresponding check boxes.

The second tab allows the generation of the previous or next sentence for a given sentence and the conversion between sentences and natural numbers. As mentioned in Section 5.1.2, there are two modes, bottom-up and top-down, for computing the next or previous sentence of a given sentence. The user can choose either one through the "Next Mode" dropdown box. Thanks to the specific volume data, the user can also computer a sentence corresponding to a sequence number and vice verse.

The third tab allows the user to enumerate a bounded range of sentences by giving the sequence numbers of the first and last sentences. The range can be restricted in a hierarchy, cluster or cube, by giving the corresponding indices in which case only the sentences in the intersection of the range and the restriction are enumerated. The hierarchies, clusters and cubes are used for expert users. For normal users, we will consider how to give the restriction by giving some rules or structures in the future. After enumeration, the total time for enumerating sentences, the actual number

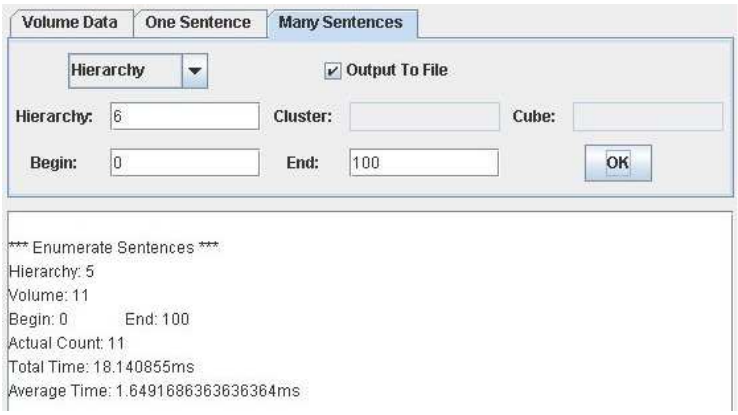

Fig. 3. Enumerating the sentences in $[0, 100]$ in the 5-th hierarchy

Fig. 4. Analyzing the coverage of a sentence set

of enumerated sentences and the average time for enumerating a sentence are also counted. For instance, we enumerate the sentences in the range $[0, 100]$ in the 5-th hierarchy for the sample grammar. Since the 5-th hierarchy volume data is 11 (< 100), only 11 sentences are enumerated. The result is given in Figure 3.

As the enumerated sentences may be too many, the output console wouldn't display them. Additionally, the toolkit allows the user to export them into a file. This file contains the corresponding sequence number and parse tree for each enumerated sentence, as well as the statistic information mentioned above.

7.1.4 Sentence Analysis Panel

Through the sentence analysis panel, the user can parse a sentence or analyze the coverage of a sentence set, including rule coverage and context-dependent rule coverage. Sentences are written as sequences of tokens separated by blanks, and one sentence one line. Since a sentence set has been generated from the sample grammar with rule coverage and length control (see Figure 2), we parse this sentence set as an instance. Figure 4 shows the result of the coverage analysis for the sentence set.

Table 1. Complexity measures of grammars

Grammar	VAR	PROD	LEV	DEP	HEI
sample	4	7	2	3	1
bExp	4	9	2	3	1
polishExp	6	12	6	1	3
elemFunc	14	38	9	6	4
miniPascal	47	81	33	8	11
ANSI C	65	214	22	38	14
Java	110	282	67	24	19

Table 2. Sentences generated by (CDRC-P / the extension) algorithm.

Grammar	No. of rules	No. of sentences	Avg len. of sentences	Min. of lengths	Max. of lengths
bExp	9	2/8	32.0/16.1	9/13	55/19
polishExp	12	2/3	18.5/16.0	5/15	32/17
elemFunc	38	2/35	256.5/28.5	8/12	505/33
miniPascal	81	4/19	191.0/65.9	23/23	329/82
ANSI C	214	75/149	115.2/87.5	8/8	2131/148
Java	282	222/471	85.4/62.5	6/6	4889/102

7.2 Experiments

We have used the toolkit to conduct some experiments for the algorithms contained in the toolkit. Most of the experiments are conducted on those grammars whose complexity measures are given in Table 1. The first one is the sample grammar used in the paper. The next five grammars are the same as those used in Ref. [6]. The `Java` grammar is retrieved from the comp.compilers FTP at `ftp://ftp.iecc.com/pub/file/`. From the table, we can see that our toolkit can be used for grammars with a relatively large complexity measure (such as the `Java` grammar).

To illustrate the features of the two algorithms with context-dependent rule coverage and their differences, we present the results of our experiments conducted on a few grammars, as shown in Table 2. For each grammar, we recorded the number of sentences, as well as the average, minimal and maximal length of the sentences generated by each algorithm. The number of rules for each grammar is also presented for comparison of grammar size with the number of generated sentences.

One experiment for Dong's algorithm is conducted on the sample grammar. We enumerated the first 1000 sentences of each hierarchy from the 7-th hierarchy to the 15-th hierarchy (the numbers were selected arbitrarily) and chose the top-down mode to generate the next sentence. The experiments were performed on a PC of P4 3.2G CPU and 2G RAM. These experimental results are given in Figure 5, which shows that the real computing time for generating the next sentence of height n is $O(n)$.

8 Applications

The toolkit has been used in several research projects, namely SAQ [18], grammar testing methods [19], testing of LFC [20, 21, 22], and Subtype Checker [23].

SAQ supports construction and validation of specifications consisting of context-

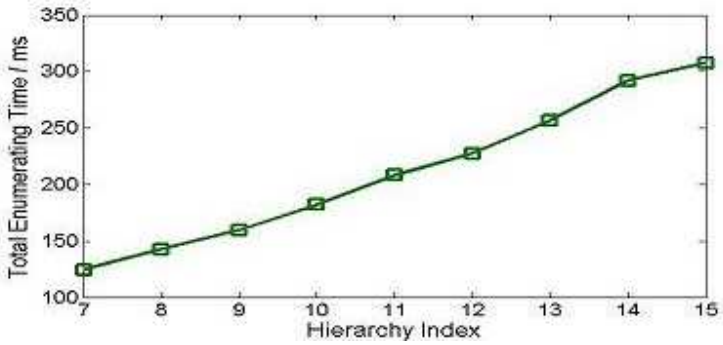

Fig. 5. Total time of enumerating sentences on the sample grammar

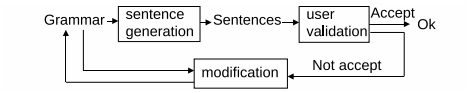

Fig. 6. Test w.r.t. a generator

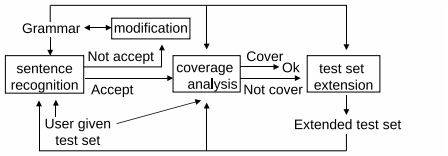

Fig. 7. Test w.r.t. a recognizer

free grammars and operations upon context-free languages. Initially SAQ supports grammar validation but quite weak [18]. We enhanced it by adding several sentence generation and enumeration algorithms contained in this toolkit. In the experiments, several errors in a number of grammars in SAQ bases were found. It is observed that the bounded range enumeration is very useful for this kind of grammar validation.

The second project is an extension of the research of the grammar validation of SAQ, which aims at systematic methods and techniques for testing context-free grammars. Generally, there are two classes of faults with a grammar G with respect to an intended language L [2]: incorrect (i.e., $\mathcal{L}(G) \nsubseteq L$) and incomplete (i.e., $L \nsubseteq \mathcal{L}(G)$). We implemented a systematic framework for testing context-free grammars [19], which contains two test methods that fit well with these two classes of faults with grammars. These two test methods are derived from the two usages of grammars, that is, a generator which generates sentences and a recognizer which accepts valid sentences. The processes of these two test methods are given in Figure 6 and Figure 7 respectively. As the figures show, sentence generation, sentence recognition and coverage analysis, which are provided in the toolkit, are three main techniques involved in the testing process. Thus they can be implemented using the algorithms in the toolkit. We used the framework to conduct some experiments on the C and Java grammars that are from comp.compilers FTP at `ftp://ftp.iecc.com/pub/file/`. Through experiments some errors were found, one of which is the sentence "indentifer {}" generated from the C grammar. This sentence is intended to define an empty C function but has syntax errors: the function name "indentifer" should be followed by "( )" or "(parameters)". The error is due to that the grammar uses the same declarator to specify function declaration and variable declaration. Indeed, these two declarators should be distinguished. Readers are referred to Ref. [19] for more detail.

LFC is a language based on recursive functions on context-free languages [20,

Table 3. Results for fault detection using CDRC-P algorithm.

Programs		Mutants		
Name	LoC[1]	Total	Killed	Ratio
bEval	8	23	9	39.1%
pEval	11	122	104	85.3%
Diff	34	130	120	92.3%
Diff2	46	273	118	43.2%
idCount	60	650	516	79.4%

21, 22]. It uses context-free grammars to specify data types, and supports pattern matching definition of functions, where patterns represent structures of the grammars. LFC was designed to support formal specification acquisition and was used in SAQ. Test data for LFC programs can be derived directly by using the sentence generation algorithms provided by this toolkit. To investigate the usefulness of the generated test data in terms of fault detection capability, we added a testing module to conduct mutation testing [24] on the test data. In particular, we took CDRC-P algorithm as test data generator and conducted mutation testing on five LFC programs, which are taken from SAQ. We applied five mutation operators [25] to each program and compared the result of each mutant with the original result to check whether a mutant is killed by the test data. Experimental results are summarized in Table 3. From the last column we can see that the ratio of the killed mutants to the total mutants is above 30% for all the five programs, which reflects the relatively good capability for fault detection of the test data generated by the CDRC-P algorithm in this toolkit.

Subtype Checker is a type-checker prototype of Ref. [23], which is a research project to study parametric polymorphism for a type system with recursive, product, union, intersection, negation and function types. According to Definition 3.1 in Ref. [23], type expressions can be expressed as a context-free grammar. Thus, we used sentence generation methods based on sentence enumeration provided in the toolkit (i.e., random generation and structure-sensitive generation) to generate arbitrary type expressions to test the prototype, in particular, to check the termination. Through experiments we found several programming errors and conjectured that Subtype Checker terminates on all recursive types. Moreover, it is required that the recursion must traverse a type constructor. In other words, once a rule for recursive types (e.g., $<T> -> mu\ x.\ <T>$) is used, then the sentence generated from this rule (e.g., $<T>$ at the right hand) must cover either a rule for product types (e.g., $<T> -> (<T>, <T>)$) or a rule for function types (e.g., $<T> -><T>\rightarrow<T>$). This can be done using the structure-sensitive generation, the feature provided in our toolkit but not in other tools as we known. Without this feature, we have to carefully and accurately define a special input grammar, such as to distinguish the non-recursive types and recursive types by different nonterminals, wherein it is error-prone and redundant. However, the structure-sensitive generation provided in our toolkit is simple as it only considers one-step derivation. An alternative approach is to use sentence generation with rule coverage. But it yields a sentence set that covers all the rules rather than a sentence that covers some required rules. In the future, we will think

[1] Line of Codes

about how to extend the structure-sensitive generation with many-steps derivation and how to generate a sentence (set) that covers some required rules.

9 Related Work

Bilska et al. [26] presented a collection of tools for formal languages and automata theory. The tools include `JFLAP` for creating and simulating finite automata, pushdown automata and Turing machines, `PumpLemma` for proving specific languages are not regular, and some others. Zeph Grunschlag implemented a `Java` package `JavaCFG` [27], which is a command-line program for viewing parse trees of context free grammars. Møler [28] developed a grammar tool which contains a parser and an ambiguity analyzer for context-free grammars. The parser parses a sentence without giving parses. This tool is used mainly for analyzing grammar ambiguity. Almeida et al. [29] presented an interactive graphical environment `CGM` for the manipulation of context-free languages. It allows the editing of context-free grammars, the conversion to Chomsky normal form, sentence parsing and the construction of parse trees. The above tools do not provide sentence generation, sentence enumeration, or coverage analysis as our toolkit.

Programming Languages Laboratory at University of Calgary provided an online tool `Context Free Grammar Checker` [3] to check the basic properties of context free grammars, including the usefulness and reachability of the nonterminals, the first sets, the follow sets and so on. After checking, it generates no more than 20 sentences, which are the first 20 ones ordered by the sentence length (token number). Compared with ours, its sentence generation is too simple.

Another related work is the sentence generation tool `Forson` [4], implemented by Alfonso Tarantini. `Forson` takes a Bison grammar file as input and provides random sentence generation, following Grow's algorithm (may not terminate), or sentence generation with rule coverage, following Purdom's algorithm, for the grammar defined in the input file. It only covers a small part of the functionalities of our toolkit. Moreover, `Forson` is a batch program, which means that it does not provide a graphical user interface.

10 Conclusion

The paper presents a toolkit for generating sentences from context-free grammars. The toolkit supports sentence generation with coverage criteria, sentence enumeration and sentence analysis. It also contains a simple graphical user interface, through which the user can use the toolkit directly. The toolkit has been used in several research projects. The toolkit provides richer functionalities for sentence generation, enumeration and analysis than existing tools, thus is expected to have more applications.

Our ongoing work includes the study of the unresolved problems mentioned in the paper, a web interface for the toolkit and experiments on more grammars. In the future, we plan to further enrich the toolkit by adding more functionalities, such as a graphical parse tree editor and algorithms for regular grammars.

Acknowledgment

The authors thank Yunmei Dong for his suggestions on developing the toolkit and on implementing, improving and experimenting with the sentence enumeration algorithm and its GUI.

References

[1] Klint, P., Lämmel, R., Verhoef, C.: Towards an engineering discipline for grammarware. ACM Transaction on Software Engineering and Methodology **14**(3) (2005) 331–380

[2] Lämmel, R.: Grammar testing. In: Proceedings of Fundamental Approaches to Software Engineering (FASE'01). Volume 2029. (2001) 201–216

[3] Programming Languages Laboratory, U.o.C.: Context free grammar checker. `http://smlweb.cpsc.ucalgary.ca/`

[4] Tarantini, A.: Forson. `http://sourceforge.net/projects/forson/`

[5] Purdom, P.: A sentence generator for testing parsers. BIT **12**(3) (1972) 366–375

[6] Zheng, L., Wu, D.: A sentence generation algorithm for testing grammars. In: Proceedings of the 33rd Annual International Computer Software and Applications Conference (COMPSAC'09). Volume 1. (2009) 130–135

[7] Alves, T.L., Visser, J.: A case study in grammar engineering. In: Proceedings of the 1st International Conference on Software Language Engineering (SLE'08). Volume 5452. (2008) 285–304

[8] Shen, Y., Chen, H.: Sentence generation based on context-dependent rule coverage. Computer Engineering and Applications **41**(17) (2005) 96–100 In Chinese.

[9] Dong, Y.: Counting and hierarchical lexicographic enumeration of CFL sentences. Science in China (Series E) **36** (2006) 1375–1413 In Chinese.

[10] Dong, Y.: Linear algorithm for lexicographic enumeration of CFG parse trees. Science in China (Series F - Information Science) **52**(7) (2009) 1177–1202

[11] Dömösi, P.: Unusual algorithms for lexicographical enumeration. Acta Cybernet **14**(3) (2000) 461–468

[12] Xu, Z., Zheng, L., Chen, H.: A toolkit for generating sentences from context-free grammars. In: Proceedings of the 2010 8th IEEE International Conference on Software Engineering and Formal Methods. (2010) 118–122

[13] Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. 3rd edn. Addison Wesley (2006)

[14] Csuhaj-Varjú, E., Kelemenová, A.: Descriptional complexity of context-free grammar forms. Theor. Comput. Sci. **112** (May 1993) 277–289

[15] Earley, J.: An efficient context-free parsing algorithm. Communications of the ACM **13**(2) (1970) 94–102

[16] Malloy, B.A., Power, J.F.: An interpretation of purdom's algorithm for automatic generation of test cases. In: Proceedings of the 1st International Conference on Computer and Information Science (ICIS'01). (2001) 3–5

[17] Woo1, G., Chae, H.S., Jang, H.: An intermediate representation approach to reducing test suites for retargeted compilers. In: Proceedings of the 12th International Conference on Reliable Software Technologies. (2007) 100–113

[18] Dong, Y., Li, K., Chen, H., Hu, Y., Zhang, R., Tang, R.: Design and implementation of the formal specification acquisition system SAQ. In: Proceedings of the Conference on Software: Theory and Practice, IFIP 16th World Computer Congress 2000. (2000) 201–211

[19] Zheng, L., Chen, H.: A systematic framework for grammar testing. In: Proceedings of the 8th International Conference on Computer and Information Science (ICIS'09). (2009) 1013–1019

[20] Dong, Y.: Recursive functions of context free languages (I)-the definitions of CFPRF and CFRF. Science in China (Series F) **45**(1) (2002) 25–39

[21] Dong, Y.: Recursive functions of context free languages (II)-validity of CFPRF and CFRF definitions. Science in China (Series F) **45**(2) (2002) 1–21

[22] Chen, H., Dong, Y.: Towards practical computable functions on context-free languages. In: Proceedings of the 4th Annual Conference on Theory and Applications of Models of Computation(TAMC'06). Volume 3959. (2006) 555–565

[23] Castagna, G., Xu, Z.: Set-theoretic foundation of parametric polymorphism and subtyping. In: Proceedings of the 16th ACM SIGPLAN International Conference on Functional programming (ICFP'11). (2011) to appear.

[24] Richard A. DeMillo, Richard J. Lipton, F.G.S.: Hints on test data selection: Help for the practicing programmer. **11**(4) (1978) 34–41

[25] Offutt, A.J., Lee, A., Rothermel, G., Untch, R.H., Zapf, C.: An experimental determination of sufficient mutant operators. ACM Transactions on Software Engineering Methodology **5** (1996) 99–118

[26] Bilska, A.O., Leider, K.H., Procopiuc, M., Procopiuc, O., Rodger, S.H., Salemme, J.R., Tsang, E.: A collection of tools for making automata theory and formal languages come alive. In: ACM SIGCSE Bulletin. (1997) 15–19

[27] Grunschlag, Z.: JavaCFG. `http://www.cs.columbia.edu/~zeph/software/JavaCFG/` (2004)

[28] Møler, A.: dk.brics.grammar. `http://www.brics.dk/grammar/` (2008) Latest release.

[29] Almeida, A., Alves, J., Moreira, N., Reis, R.: CGM: A context-free grammar manipulator. In: Proceedings of Compiler, Related Technologies and Applications (CoRTA'08). (2008)