Grammar-based Test Case Generation

Aalok Thakkar
CIS 700: Software Analysis and Testing

In the previous class, we learned about property based test case generation and using the Korat tool for generating all binary trees with fewer than three nodes upto isomorphism. Continuing on the same theme, this article will introduces grammar-based test case generation and discusses its limitations.

1 Generating Binary Trees

Recall that a binary tree is a data structure in which each node has at most two children, which are referred to as the left child and the right child. Here is how you would define the data structure in Haskell:

```
data BinaryTree a = Empty | Node (BinaryTree a) a (BinaryTree a)
```

If you are unfamiliar with the Haskell syntax or would like to learn more about trees in Haskell, please refer to Learn You a Haskell for Great Good! In class, we tried to generate binary trees (up to isomorphism) of bounded size. We learned that naively generating linked structures and checking if they are trees is not a good way to go about it. The naive approach generates 16,284 possibilities but Korat can minimize the number of cases to a way smaller number. The table below summarizes the number of cases for binary tree generation:

Number of Nodes	**Number of Binary Trees**	**Candidates Considered by Korat**	**Candidates Considered by Naive Approach**
8	1430	54418	2^{53}
10	16796	815100	2^{72}
12	208012	12284830	2^{92}

Table 1: Generating Binary Trees

2 Grammar-based Generation

However, one can easily give a context-free grammar to generate a binary tree. The definition itself, gives us a way to do it.

$$\mathsf{BinT} \rightarrow \epsilon \mid [\mathsf{BinT}]\, a\, [\mathsf{BinT}]$$

Where ϵ denotes an empty tree. Using this grammar, we can generate binary trees of size N. Let BinT_n denote all trees of size exactly n. We define BinT_n inductively as:

$$\mathsf{BinT}_0 \rightarrow \epsilon$$
$$\mathsf{BinT}_n \rightarrow [\mathsf{BinT}_p]\, a\, [\mathsf{BinT}_q]$$

Where $p + q = n - 1$. Then, all we need to do is compute the language generated by BinT_N. A straightforward implementation would be:

```
allTrees 0 = [Empty]
allTrees n = [Node tl 0 tr | p <- [1 .. (n-1)], q <- [1 .. (n-1)], (p + q + 1 == n)
                             tl <- allTrees p, tr <- allTrees q]
```

As we are only interested in the structures of the tree (up to isomorphism) and not the values stored in their nodes, the above code just assigns a default value (0) to every node. The question of interest is that if we can generate trees with just two simple lines of code, why do we need a tool like Korat? Indeed for certain data structures, we do not need Korat. All we need is the grammar which characterizes the possible inputs to our program. This is the fundamental idea underlying the grammar-based approach. Please find more about Grammar-based Test Case Generation at this blog and this survey.

But does the set of inputs to a program always admit a context-free structure? We know that the set of all inputs to a (terminating) program is characterized as a recursive set, which contains but is not limited to languages generated by context-free grammars. Let us look at a concrete example in the context of automated test case generation.

3 Red-Black Trees

A Red-Black Tree is a binary tree such that:

1. Each node is either red or black.
2. The root is black.
3. All leaves are black.
4. If a node is red, then both its children are black.
5. Every path from a given node to any of its descendant leaves contains the same number of black nodes.

We use the following way to express Red-Black trees:

```
data Color = Red | Black
data RBTree a = Empty | Node Color (RBTree a) a (RBTree a)
```

You can find more about Red-Black trees and their applications at this Wikipedia article and in this book. We wish to show that the set of all Red-Black trees cannot be generated by a context-free grammar.

In order to prove this, we first translate Red-Black trees into a set of strings on a binary alphabet, and then prove that this set is not context-free. Please refer to this, this, or this book to recall topics in automata theory in general, and context-free languages in particular.

Proposition 1. Let $\mathbb{T}$ be the set of all Red-Black trees.

$$\phi : \mathbb{T} \to \{a+b\}^*$$

$$\phi : \mathsf{Empty} \to \epsilon$$

$$\phi : \mathsf{Node\ Red\ (LTree)\ \_\ (RTree)} \to \phi(\mathsf{LTree})\phi(\mathsf{RTree})$$

$$\phi : \mathsf{Node\ Black\ (LTree)\ \_\ (RTree)} \to a\phi(\mathsf{LTree})\phi(\mathsf{RTree})b$$

If $\mathbb{T}$ is context free, then $\phi(\mathbb{T})$ is context free.

The proof of this proposition is left as an exercise. Think about how the grammar of $\mathbb{T}$ can be transformed to a grammar of $\phi(\mathbb{T})$.

Proposition 2. $\phi(\mathbb{T})$ is not context-free.

Proof. Let R be the regular language $\{a^n b^{n'} a^{m'} b^m \ n, n', m, m' \in \mathbb{N}\}$. Let $L = \phi(\mathbb{T}) \cap R$. Observe that:

$$L = \{a^n b^{n'} a^{n'} b^n : n' < n; n, n' \in N\}.$$

For sake of contradiction, if $\phi(\mathbb{T})$ were context-free, then L would be context free. From first course in automata theory, we know that L is not context-free. Therefore, $\phi(\mathbb{T})$ is not context-free. □

4 Conclusion

From the two propositions, we can conclude that $\mathbb{T}$ is not context free, that is, the set of all Red-Black trees cannot be characterized by a grammar, and our grammar-based test-case generation approach does not work here. In such cases, we need tools like Korat. But what about the data structures in Assignment 3? Can you find grammars to generate those structures, or prove that there are no grammars characterizing them?

सम्प्राप्ते रक्तश्यामवृक्षविषये
नहि नहि रक्षति डुकृङ्करणे