Chapter 13

Test Generation from Formal Specifications

The purpose of this chapter is to introduce techniques for the generation of tests from formal specifications written in the Z (Zed) notation.

13.1. Introduction

Requirements serve as the basis of test generation. Requirements for a program under test can be known to the test generator in a variety of forms. A commonly used form is a plain English language description. The tester generates test cases from these requirements.

While requirements written in plain English allow quick understanding and dissemination to a wide audience, they suffer from several shortcomings. A serious shortcoming arises from the inherent ambiguity of expression in almost any natural language. Any ambiguity in the requirements specification raises the possibility of generating an incorrect test case. Such a test case, if passed by the program under test, might lead to a program that is correct as per the tester's interpretation but unacceptable to the customer.

To avoid, or at least reduce the chances of misunderstanding, one often takes refuge in some mathematical notation. Requirements written in plain natural language are rewritten using precise mathematical notation. Such a description is known as *formal specification*, or simply *specification*. The hope in generating a formal specification is that the process of developing one from requirements in a natural language will allow the discovery of any ambiguities and lead to a precise form of requirements.

Given that mathematical specifications themselves are not necessarily always correct, specifications are also subject to some form of testing. Nevertheless, once an acceptable set of specifications is obtained, it serves as input to a variety of processing tasks, one of which is

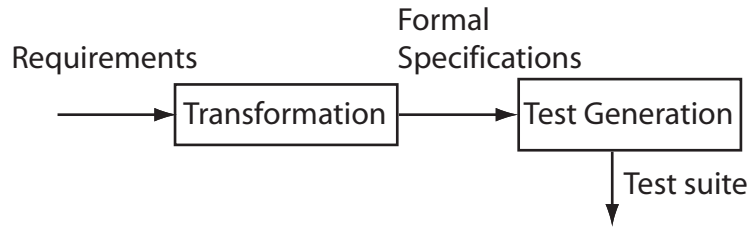

Figure 13.1: A generic view of the specification based test generation process.

test generation.

A variety of techniques exist for the generation of tests from a formal specifications. Figure 13.1 illustrates a generic process. First, requirements available in a natural language are transformed to a formal specification. The formal specification is fed to a test generation procedure that outputs a set of test cases. The test generation procedure might be automated, semiautomated, or manual.

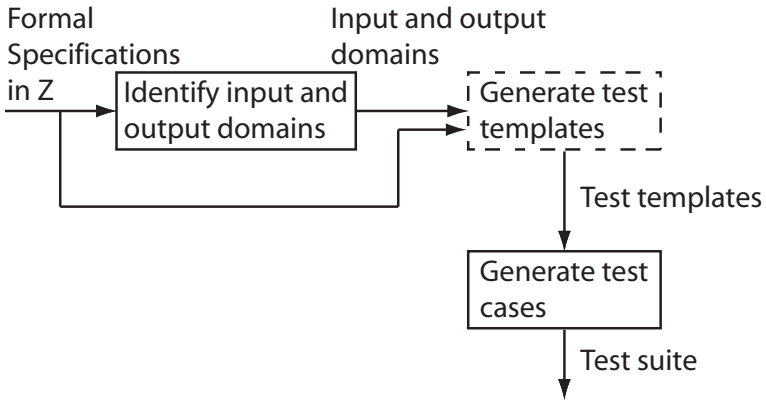

Figure 13.2: A procedure for generating tests from Z specifications.

Test generation procedures depend on the notation used in the formal specifications. In this chapter we describe the generation of tests when the formal specification uses Z, a common notation for the formal specifications of programs. Specifications written using Z are known as Z-specifications. For those who pronounce the letter "Z" as *zee*, it is worth noting that the Z in "Z notation" is pronounced as "Zed."

Figure 13.2 shows the three steps in generating tests when specifications are written using Z. The first step is to identify input domain and the output domain (the range) of the program under test. A set of test templates is generated from the input domain, the range, and additional information available from the specification. A test template is an abstract specification of a class of test cases.

The templates are input to a test case generation procedure that generates the test cases. The template generation step is inside a dashed box indicating that some methods for generating tests from a Z specification do not generate test templates and, instead, generate directly from the input domain, the range, and additional information in the specifications. The steps in

Figure 13.2 are described in some detail in this chapter.

13.2. The Z notation in brief

We begin with an informal overview of the Z notation. Several books on Z are available to help you acquire a more detailed knowledge of the notation. Here we focus on a few elements of Z that are used in the illustrative examples throughout the chapter.

The Z notation is based on typed set theory and first order predicate logic. The examples given in this section are deliberately kept simple for ease of understanding. Note that our focus is on test generation from formal specifications of software systems; this section provides the necessary background information. The Z notation can also be used to specify hardware as well as mixed software-hardware systems.

13.2.1. A Z specification

A useful software system consists of one or more operations that transform some input data to output data. In performing such a transformation, the system moves from one state to another. The Z notation is intended to serve as a precise specification of such a software system. A Z specification is also known as a *model-based* specification. The specification serves as a mathematical model of the software system. The implementation must obey the constraints implied by the model.

A Z specification consists of a sequence of *schemas* interspersed with informal commentary. The commentary could be in any natural language to explain what is specified precisely using the mathematical notation. Schemas serve as the basic building block of the model of the software system specified.

A schema describes both the static and dynamic aspects of a system. The static aspects include the states that the system can occupy and the relationships that remain invariant when the system moves from one state to another. The dynamic aspects include the specification of operations in terms of their input and output relationships. Usually, one or more schemas specify one operation of the software system. Let us now take a brief look at various elements of Z.

13.2.2. Sets

Z provides a basic glossary of sets often used in formal specifications and computer programs. These include the set of natural numbers, i.e. integers 0 and higher, denoted by $\mathbb{N}$, the set of integers denoted by $\mathbb{Z}$, and the set of reals denoted by $\mathbb{R}$. The set of strictly positive integers is denoted by $\mathbb{N}_1$.

You can define your own sets in a several ways. First, you could simply display the elements of a set as follows.

$\{$*hydrogen*, *helium*, *oxygen*, *nitrogen*, *florine*$\}$

You can name the sets you define and use the name at a later point in your specification such as the following.

GAS== {*hydrogen*, *helium*, *oxygen*, *nitrogen*, *florine*}

The symbol "==", as used above, can be read as *is defined as*. Z allows the construction of new sets from existing ones through their cartesian products.

Let us assume that *SYMBOL* denotes the set of symbols in the periodic table such as H for hydrogen, Na for sodium, and so on. We can now define a set of pairs of symbols and their corresponding atomic numbers as follows.

$$\textit{SYMBOL} \times \mathbb{N}_1$$

The above expression defines a set of pairs in which the first element of each pair is a symbol and the second element a strictly positive integer that denotes the element's atomic number. Examples of such pairs are $(H, 1)$ and $(O, 8)$. The following cartesian product defines a set of triples.

$$\textit{SYMBOL} \times \mathbb{N}_1 \times \mathbb{R}$$

The expression above denotes a set of triples where the first component of each triple is a symbol, and the second and third components can represent, respectively, its atomic number and the average atomic mass. Examples of such triples include, $(H, 1, 1.00794)$ and $(O, 8, 15.9994)$.

Note that while triple $(H, 2, 4.002602)$ belongs to the set of triples defined above, it is not a valid triple in the periodic table known to humans. A cartesian product merely defines a set of tuples that could be interpreted in several ways. In this example we interpreted a triple as an element in the periodic table. We will learn more about how to specify constraints on tuples.

We can define a new set from another by suitably constraining the selection of its elements as in the following example.

$$\textit{CEL} == \textit{SYMBOL} \times \mathbb{N}_1 \times \mathbb{R}$$

$$\textit{HVEL} == \{e : \mathit{CEL} \mid \mathit{third}(e) > \mathit{minMass}\}$$

First we have defined the set of chemical elements, *CEL*, to be a set of triples. Next, we defined the set of heavy elements, *HVEL*, to be the set of only those triples whose third component is greater than *minMass*. Here the third component of a triple is extracted from the triple by the function *third*.

In the example above we have defined the set named *HVEL* using a technique known as *set comprehension*. In its general form, a set definition using set comprehension looks like the following.

$$\{\mathit{declaration} \mid \mathit{predicate} \bullet \mathit{expression}\}$$

In our example defining the set $HVEL$, $e : CEL$ corresponds to the declaration part and $\mathit{third}(e) > \mathit{minMass}$ to the predicate part; the expression part is empty. One could also define a set in Z by specifying the range of values it contains. For example, the range $1 \ldots 118$ specifies the set containing integers 1 through 118.

13.2.3. Types

A type is a set of objects with constraints. Each of the sets defined in the previous section is a type; we did not impose any explicit constraint. Z has a rich collection of ways to define types and declare variables of specific types. For starters, we can define a type by listing its name inside brackets as follows.

[*SYMBOL*]

Once listed as above, *SYMBOL* serves as a basic type and can be used anywhere in a Z specification. The following additional examples show how variables can be assigned to types via declarations and how new types are defined.

$size : \mathbb{N}$	*size* is a natural number.
$gs : GAS$	gs is a GAS object.
$g : \mathbb{P}\, GAS$	g is a set of *GAS* objects.
$el : \mathbb{P}(SYMBOL \times \mathbb{N}_1 \times \mathbb{R})$	*el* is a set of triples as defined earlier.
$el : CEL$	Same as the definition given earlier.

The third and fourth declarations above need some explanation. The symbol $\mathbb{P}$ stands for *power set*. The power set of $\mathbb{Z}$, denoted $\mathbb{P}\,\mathbb{Z}$, is the set of all subsets of the set of integers. Some members of $\mathbb{P}\,\mathbb{Z}$ are $\varnothing$ (the empty set), {0, 1, 2}, and {-2, 14}. The expression $\mathbb{P}$ *GAS* denotes the set of all subsets of the set *GAS*. Some elements of $\mathbb{P}$ *GAS* are {*hydrogen*}, {*hydrogen*, *florine*}, and {*nitrogen*, *oxygen*}. The set $\mathbb{P}(SYMBOL \times \mathbb{N}_1 \times \mathbb{R})$ is a set of triples explained earlier.

An *axiomatic* description in Z declares variables and specifies an optional set of constraints on the values of the variables declared. Once so defined, these variables become *global* and can be used from the place of definition until the end of the specification. Let us examine the following axiomatic description.

$$
\begin{array}{|l}
maxNum, maxSize : \mathbb{N}_1 \\
maxMass : \mathbb{R} \\
\hline
count \leq maxSize \\
maxNum \leq 118 \\
maxMass \leq 262 \\
maxSize = 150
\end{array}
$$

The description above is separated into two parts by a horizontal dividing line. The top part contains declarations of variables *maxNum*, *maxSize*, and *maxMass* and the bottom part the constraints on these variables.

13.2.4. Expressions and types

The arithmetic operators $+$, $-$, $*$, div, and mod can be used to create arithmetic expressions with integers. Set expressions are formed using the traditional set operators $\cup$, $\cap$, and $\setminus$.