Test Cases Generation for URL Parsing

Jinghao Sun, Tianyu Li
School of Electronic Engineering and Computer Science
Peking University

Abstract

For url parsers, the correctness of the parser is crucial. In this project, we aim to generate test cases for the url parser. We first analyze the structure of the url and then write some test cases based on the structure. Then we write a URL Parser Tester to generate test cases automatically. We test performance of the wirtten URL parser with the tester, collect data and arrange it.

1 Introduction

URL

A uniform resource locator (URL), colloquially known as an address on the Web, is a reference to a resource that specifies its location on a computer network and a mechanism for retrieving it.

Every HTTP URL conforms to the syntax of a generic URI. The URI generic syntax consists of five components organized hierarchically in order of decreasing significance from left to right:

`URI = scheme ":" ["//" authority] path ["?" query] ["#" fragment]`

For URLs, the `path` is divided into `hostname` and `path`. Below is an example URL:

`https://example.com:8080/path/to/resource?search=query#section`

URL parser

URL Parser parses URL strings into seperate parts. For the URL above, a parser should seperate it into:

- Scheme: ‘https’
- Hostname: ‘example.com’
- Port: ‘8080’
- Path: ‘/path/to/resource’
- Query: ‘search=query’
- Fragment: ‘section’

The parser we write is simple. It uses simple Regular Expressions to parse the URL.

```
    R"(([^:/?#]+):\/\/([^:/?#]*)(?::(\d+))?([^?#]*)(?:\?([^#]*))?(?:#(.*))?)"
```

The parser is not perfect and may have bugs. We need to write test cases to test the parser.

URL parser tester

URL Parser Tester is a tool to generate test cases for URL Parser. It generates test cases based on the structure of the URL. The parser tester we write is written in C++, which generates both legal and illegal test cases. The tester also has built-in performance test, which is manually written. After the parser parses the URL, the tester will check the correctness of the parser.

2 URL generation

URL generation is generally divided into two parts: segment creation and concatenation.

For equivalence classification, there are legal and illegal urls. For legal url, it can be divided into normal and strong legal urls.

URL	
Legal URL	Illegal URL
Normal Legal URL	
Strong Legal URL	

2.1 Segment creation

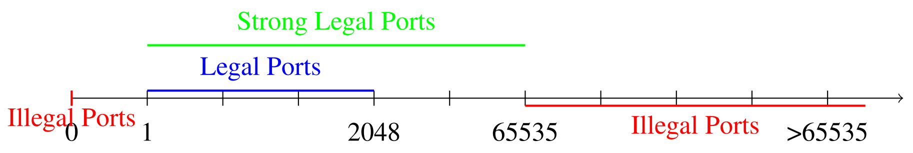
Scheme For conventional schemes, we have these below:

```
"file","http","https","FTP","gopher","mailto","MMS",
"ed2k","Flashget","thunder","tel","sms",...
```

When generating legal schemes, we choose from the above list. For strong legal schemes, we generate random strings with legal length, but closer to the boundary. For illegal schemes, we generate random strings with illegal length.

Hostname For legal hostnames, we generate random strings with legal length. The hostname varies from normal domain names to IP addresses, with both IPV4 and IPV6. For strong legal hostnames, we generate random strings with legal length, but closer to the boundary.

Port For legal ports, we generate random integers from 1 to 2048. For strong legal ports, we generate random integers from 1 to 65535. For illegal ports, we generate 0 or intergers larger than 65535.



Path For legal paths, we generate random strings with legal length, seperated by ‘/’. For strong legal paths, we generate random strings with legal length, but closer to the boundary. Meanwhile, there are empty paths and path with a single ‘/’. For illegal paths, we generate random strings with illegal characters.

Query For legal queries, we generate random query pairs with legal length, seperated by ‘&’. For strong legal queries, we just make them longer. For illegal queries, we generate some pairs with empty value.

Fragment For legal fragments, we generate random strings with legal length(0-5 characters). For strong legal fragments, we generate random strings with legal length, but relatively longer(0-30 characters). For illegal fragments, we generate random strings and insert illegal characters(blank for example).

2.2 concatenation

After generating segments, we concatenate them to form a URL. The URL is generated by concatenating the segments with the following format:

```
scheme://hostname:port/path?query#fragment
```

For illegal URLs, we adapt another method: After the scheme, we choose strings like ‘:/’, ‘:////’, or empty strings to make the URL illegal.

3 Performance

In main function, we have basic legal test, basic illegal test, random legal test, and random illegal test. For basic legal test, 8/12 test cases are passed. For basic illegal test, 16/17 test cases are passed. For random legal and illegal test, we generate 10000 urls for each, and run 10 times. Result is shown as below:

Performance Test			
Legal URL		Illegal URL	
Test	Pass	Test	Pass
20000	2616	10000	5731
20000	2515	10000	5825
20000	2529	10000	5716
20000	2587	10000	5789
20000	2503	10000	5872
20000	2522	10000	5743
20000	2511	10000	5781
20000	2540	10000	5686
20000	2578	10000	5763
20000	2527	10000	5830

We can see that the result is rather stable, and the parser does have some bugs. According to the output, parser designers can find the bugs and fix them.

4 Conclusion

In this project, we write a URL parser tester to generate test cases for URL parser. It successfully generates large amount of test cases and find a lot of bugs in the parser. Also, we gain some experience in creating a C++ project and collaborating to finish a programming project. The tester is not perfect and it is not comprehensive enough, but it is useful to make basic tests. Some software testing technologies is applied, like equivalence classification, boundary value analysis and random testing.

Considering praticality, a shell script is convenient to act as interface. For general parsers, we can write a shell script and connect the parser and the tester together. The tester generates urls and send them to the parser, then the tester will read the parser's output and analyze.

See the github repository for more information.