Automated Debugging

Software Engineering Chair November 6, 2008

Project 1: Simplifying Input

In this project, you write a simple tool that simplifies failure-inducing input. The idea is to use delta debugging to find a minimal input that causes an XML parser to fail.

The program

Download XMLProc from `http://www.st.cs.uni-sb.de/whyprogramsfail/code/xmlproc-0.70a.zip`. The file `xmlproc/xpcmd.py` is a command-line interface to the XML parser. XMLProc has a small defect. The input file `xmlproc/demo/urls.xml` causes the parser to fail:

```
$ python xmlproc/xpcmd.py xmlproc/demo/urls.xml
xmlproc version 0.70

Parsing 'xmlproc/demo/urls.xml'
Traceback (most recent call last):
  File "xmlproc/xpcmd.py", line 112, in <module>
    p.parse_resource(sysid) [...]
UnboundLocalError: [...]
$ _
```

Your task

Use Delta Debugging to simplify the failure-inducing input. Proceed in eight steps:

1. Write a testing function. Start with a Python program that invokes the parser, as described above, and assesses the result. To avoid future complications, you may wish to create an empty `__init__.py` file in the xmlproc directory and work from its parent directory. Invoke the parser directly from Python. Take a look at the `xpcmd.py` source to see how this works. The essential lines are

   ```
   from xmlproc.xml.parsers.xmlproc import xmlproc
   p = xmlproc.XMLProcessor()
   p.parse_resource(file)
   ```

   In order to catch the exception thrown by the parser, be sure to read and understand exception handling in Python. Take care to differentiate three outcomes:

- The parser completely parses the file (PASS), without errors or warnings
- The parser fails as in the original failure (FAIL)
- The parser has another outcome, in particular syntax errors (UNRESOLVED)

2. Write a splitting function. You can start with the `split()` function as described in the "Why Programs Fail" book (example 5.5). In the long term, you may want to split the input along token delimiters, or even use syntactic simplification (section 5.8.3).

3. Attach Delta Debugging. To complete Delta Debugging, you need an implementation of the `listminus()` function as described in the book (example 5.6). The listsets module[1] gives you exactly this. Finally, you need the ddmin() function from example 5.4. There is a ddmin module[2] that even comes with a self-contained test function.

4. Choose a representation. How do you represent the configuration that is to be minimized? If you want to simplify input, each delta might be a single character—thus, the string `"defect"` becomes the list `['d', 'e', 'f', 'e', 'c', 't']`. However, this will cause trouble as you cannot distinguish between the same character at different locations. For instance, `listminus(['d', 'e', 'f', 'e', 'c', 't'], ['e']) = ['d', 'f', 'c', 't']`—which is not what you want. A better solution is to store characters with their indices, as in `[('d', 1), ('e', 2), ('f', 3), ('e', 4), ('c', 5), ('t', 6)]`—this way, each circumstance can be uniquely identified. For a more general solution, you may wish to set up Circumstance or Delta classes and to store lists of these objects.

5. Run it. What is the simplified failure-inducing input your tool extracted from `xmlproc/demo/urls.xml`? Record the number of tests that Delta Debugging takes.

6. Document it. Be sure to have docstrings for every function which describe its purpose.

7. (Optional) Improve efficiency by adding caching and/or syntactic simplification:

   - Delta debugging sometimes executes some tests more than once. In order to avoid the parser being called with the same input more than once, you can cache executed tests and their results

[1] http://www.st.cs.uni-sb.de/whyprogramsfail/code/dd/listsets.py
[2] http://www.st.cs.uni-sb.de/whyprogramsfail/code/dd/ddmin.py

and, whenever possible, use the cache instead of invoking the parser.

- In the book, Section 5.8.3 sketches how to implement syntactic simplification. Using a Python XML parser such as Expat (xml.parsers.expat), split the input along the XML structure to narrow down the failure cause more rapidly.

In both cases, validate the success by counting the number of tests. Take a look at the failure-inducing input found by Delta Debugging when using syntactic simplification and compare it to the failure-inducing input found when using single characters as deltas.

8. Provide an appropriate interface to your tool. It should accept input and give output exactly like this:

```
$ python yourtool.py xmlproc/demo/urls.xml
Relevant input: ...
Tests performed: ...
```

If you also implement one (or both) of the optional parts, there are additional input configurations your tool must handle:

```
$ python yourtool.py -cache xmlproc/demo/urls.xml
...
$ python yourtool.py -xml xmlproc/demo/urls.xml
...
```

Grading

Grading can be summarized as follows:

- Base solution only (no caching, no syntactic simplification): 60%
- Syntactic simplification: +30%
- Caching: +10%

Submission

Submit your solution as a .zip file to `wasylkowski@st.cs.uni-sb.de` with `[Project 1]` in the subject. Provide your full name and immatriculation number in the body of the email.

The deadline is **2008-11-19 23:59**.