CS302 Spring 1999 – Regression Testing

Regression testing is a common approach to testing compilers. The general idea is to maintain a test suite and the compiler outputs—the 'generated code' and the diagnostics—for each program in the suite. In the early development stages, the 'generated code' may not be code at all, but some other output, e.g., a list of variables and types or intermediate code. These outputs are called the *baseline*.

Each time the compiler is changed, it is tested with the programs in the test suite and the results are compared with the baseline. For many changes, e.g., algorithm improvements, the output is identical to the baseline. Unexpected differences suggest errors. For other changes, e.g., generating better code or supporting additional source-language features, the output will differ from the baseline in presumably expected ways. Once the changes are accepted as correct, the latest outputs are established as the new baseline.

A small test suite for PCAT is in `/u/cs302acc/pcat`. (You are encouraged to mail additional programs to `apt` for inclusion in the suite.) To test some compiler phases (e.g., typechecking) it is suitable to include invalid programs as well as valid ones in the test suite.

Details of the testing scheme depend on the compiler stage being tested. When the output of the compiler is some intermediate file (e.g., an ast), its contents can be compared with expected contents. When the 'generated code' is actually executable (either directly or via an interpreter), testing should also execute it, and compare the outputs with the expected outputs.

This testing scheme can be automated using a shell script. For example, you can test your interpreter with the help of the script `/u/cs302acc/2/dotest`, which compiles each test program to an `.ast` file, type-checks it, interprets it leaving the outputs in the files described above, and compares the outputs with the baseline. If the baseline files don't exist, `dotest` creates them from the current outputs. Here is `dotest`:

```
#!/bin/sh
cd tst
for i
do
        d=`dirname $i`
        f=`basename $i .pcat`
        echo $f:
        /u/cs302acc/bin/frontend $d/$f.pcat > $f.ast 2> $f.err
        if [ -r $d/$f.in ]; then
          ../interp $f.ast < $d/$f.in > $f.out 2>> $f.err;
        else
          ../interp $f.ast > $f.out 2>> $f.err;
        fi;
        if [ -r $f.out.bak ]; then diff $f.out.bak $f.out;
          else cp $f.out $f.out.bak; fi
        if [ -r $f.err.bak ]; then diff $f.err.bak $f.err;
          else cp $f.err $f.err.bak; fi
done
exit 0
```

The compiler outputs and the baseline go in a private directory `./tst`. For the test program `/u/cs302acc/pcat/fibb.pcat`, the outputs are in `tst/fibb.out` and `tst/fibb.err`, and `tst/fibb.out.bak` and `tst/fibb.err.bak` are the baseline. The input is in `/u/cs302acc/pcat/fibb.in`. The other test programs are handled similarly; for those that do not read input, there is no `.in` file.

Note that the baseline is not part of the test suite; you must construct it. You may wish to construct the initial baseline by using my executable; appropriate files for the existing test suite are in `/u/cs302acc/2/tst`.

Once the outputs are deemed acceptable, you can use `/u/cs302acc/2/newbaseline` to establish a new baseline:

```
#!/bin/sh
cd tst
for j in *.out *.err ; do
        if [ -r $j.bak ]; then
                if cmp -s $j.bak $j; then rm $j
                else echo tst/$j 1>&2; mv $j $j.bak; fi
        else
                echo tst/$j 1>&2; mv $j $j.bak
        fi
done
exit 0
```

Things can be further automated by adding a `makefile` entry to execute `dotest` on a specified test suite. The `makefile` entry is something like this:

```
test:   interp /u/cs302acc/pcat/*.pcat tst/*.pcat makefile
        dotest /u/cs302acc/tst/*.pcat tst/*.pcat
```

This entry also permits an optional private test suite containing your own PCAT programs in `tst/*.pcat`. The command `make test` builds the interpreter (`interp`) and executes `dotest` on the test suites.