Sprint 2: QA Language, By Dmitry Vasin and Ben Ni

Intermediate Representation

Syntax tree represented in XML format.

Backend Translator

In order to translate from XML representation to Python we chose a language that is built specifically for reading XML, namely XQuery. The XQuery program reads the XML file and then calls the appropriate function depending on which XML element it encounters and then prints the corresponding python code. For example in the simple example of a program with 1 step, the XQuery program would print out all the starter code like loading the python libraries, then run the step checking function for every step. It would then check which instruction and which result is contained in the step and output the corresponding python code. Since we only have 1 step it would then quit. The output is our target language. In order to make running the compiler on different files simpler we created a python script which passes the filename parameter to XQuery, as it is apparently unable to do so through the command line. This script is part of the batch file. The compiler itself is located in the codegen.sq file.

Language structures not parsed

All of the language parts are parsed except non-primitive data types. Every other operation is implemented according to the spec from sprint 1. We did not parse non-primitive data types because the syntax for them is not yet set in stone.

Example inputs provided

The test case are written so that they separately test an independent function of the language and whether it is being parsed correctly.

Test case 0: Simplest test case, one action regarding object with one descriptor

Test case 1: Step with results, result and should statement checking

Test case 2: Combining action on object with result checking

Test case 3: Adding together numbers from multiple sources

Test case 4: Actions requiring arguments, like editing a textbox

Test case 5: Multi step programs

Test case 6: If statement

Test case 7: If statement with an otherwise clause

Test case 8: Go to statements

Test case 9: Loops

Test case 10: Exit statement

Test case 11: References to earlier steps and multi step result checking

Test case 12: Testing contain statement in should clause

Test case 13: Functions

Test case 14: Concatenating strings acquired from objects

Test case 15: Big program that combines everything

Running the script

In order for the XQuery compiler to work BaseX needs to be installed. Download it from here: http://basex.org/products/download/all-downloads/

And make sure that the bin folder of the installation is in your path variable.

Run the compile.bat file to output the code for testcase 15

If you want to run it for another file then replace the "output15.html.xml" with whatever file you want to test, 15 have been provided in the outputs folder.

If you want to actually run the output of our compiler to make sure it's working python and the Chrome engine need to be installed.

In order to install the Chrome engine follow these steps

1. locate the scripts folder under your python installation path (ex: C:\Program Files\Python36\Scripts)

2. download the chromedriver from https://sites.google.com/a/chromium.org/chromedriver/downloads

3. locate the scripts folder under your python installation path(ex: C:\Program Files\Python36\Scripts)

4. put the chromedriver.exe in the script folder

If you want to recompile the testcases to IR follow the directions in Sprint 1. The supplied IR representations are in the output folder. Use the qaCompiler folder in Sprint2.

Verifying output files

In order to test whether the code is correctly compiled, you can simply run the python program that is produced by the compiler. It's difficult to find websites which have all the features we are testing, therefore we have also provided a test html page on which these script can be ran. Simply launch the script and watch the actions it performs. The compiled program should output that it finished successfully, unless any step failed in which case it should output the number of the step at which it failed.

Correctness

We have manually transcribed what test case 15 should look like, which covers all the features of our language. The output of the compiler is completely the same, meaning it is correctly representing the input.

Completeness

Test case 15 includes all the features that were in the language specification, and it works correctly signifying that it is complete.

Verifiability

Any example can be verified by simply running the script created by the compiler. All the steps taken should be apparent and match the described steps in our language. And once again test case 15 is a complete test of our program as we showed during the specification phase.