BTestBox
-
Design

January 2017

Contents

1 Introduction

This document is has the architecture for the tool BTestBox. It describes how each of the components of the tool work in order to test the translation and also to test the coverage.

Main principles behind each component will be explained and how to maintain the tool.

2 Tool Architecture

2.1 btestbox.py

btestbox.py is the 'main' file of the tool, it is the first step when the executable is called and responsible for receiving the arguments. Also, it has the function "getImportedMachine" that get the BXML of the files in the clauses imports/sees/extends/includes. Furthermore, it call the function compatible with the coverage argument (*coverageprocess.py*) and after that call the file that create the HTML report (*HTMLgen.py*).

2.2 coverageprocess.py

It is the second file in the execution order called by *btestbox.py*. It has one function of each different coverage and tries to follow the same step for all of them, what changes is the manner to calculate the covered percentage and to save the non covered goals.

Following the execution, for each operation inside the implementation *coverageprocess.py* is responsible for calling the *buildgraph.py*, *buildpaths.py* and *makecoverage.py*, that are compelled to created the graph, manage the paths and to develop the coverage respectively.

After doing the necessary for each operation, *coverageprocess.py* call *createBTest.py* and *runTest.py*. Finally, finish and return to *btestbox.py*.

This file also has the functions "getInputs" and "checkIfIsLocal". The former is responsible for getting the inputs of an operations and the latter to check is an operation is local.

2.3 graphgen.py

graphgen.py is the file that creates the graph for an operation. *coverageprocess.py* calls that file with the "mapOperations" function.

The file has 5 global dictionaries that are also read by the other components of BTestBox. The main function is to surf through the BXML tree and for each tag in the operation tree the node is created and added to the graph.

"mapOperations" is a function that will make the mapping, through "makeMap", or the graph, and also add some information on the last node of the graph.

"makeMap" will call the instruction to create the node depending of the tag in the BXML tree.

"mapAssig", "mapIf", "mapNary", "mapSkip", "mapWhile", "mapOperationCall" and "mapCase" create the node to the Assignment, If, Nary_Exp, Skip, While, Operation_Call and Case tags, respectively.

"mapStart" will generate the first node of the graph, therefore it calls "solveFirstNodeData".

"solveFirstNodeData" get the clauses ASSERTIONS, INVARIANT, PROPERTIES, SETS, VALUES and CONSTRAINTS and then insert as data for

the first node. This will help in the predicate generation. This function uses "SolveFirstNodeImportedAndSees" and "solveSets" as auxiliaries.

"getImpWithImportedMch" is a function that with the name of a machine and the directory will look in the folder for the BXML tree of the implementation.

"clearGraphs" as the name suggests it will clear the global graphs of the component.

graphgen.py will also use other two components of BTestBox code, *instgen.py* and *nodescreator.py*.

2.4 instgen.py

The *instgen.py* goal is to translate the BXML tree in a string. This is important in order to ask ProB to evaluate a predicate and when BTestBox creates the copy files in *createBTestSet.py*.

Right before asking ProB to evaluate a predicate, the BXML is translated in a predicate, a string that ProB can handle.

The translation of each tag is written in this document. It is possible that new tags or an operation inside a tag that need a different approach from the translation raise an error. This could mainly be a problem during the translation from B to the target language or when asking ProB to evaluate.

2.5 nodescreator.py

This is a basic component and it function is to create the nodes of the graph or the predicate tree.

nodescreator.py is mainly used by *graphgen.py* and *buildpredicate.py*.

2.6 buildpaths.py

Component to create the guides.

Inputs: The graph dictionaries. Outputs: The guides, a dict for the branches, a dict for the node status (visited or not) and another dict for the branches status (visited or not).

The dictionaries that are created they are seen by other components and are important during the phase of building the predicate XML tree.

2.7 makecoverage.py

Intermediary component between the *coverageprocess.py* and *buildpredicate.py*. For each coverage it has a different approach due the different goals.

Basically, it takes one guide and run through it calling *buildpredicate.py* to apply the instruction of the node. Finished running the guide, it call the function "checkPredicate" that ask ProB to evaluate the predicate.

Inputs: The BXML of the operation (machine and implementation), the guides, the variables in the input parameters clause, operation name, the goals

(depend of the coverage), the imported machine list, the sees machine list, implementation name, bpd directory, Atelier-B directory, proB path, the directory for copy.

Outputs: It return if the operation was covered or not, all the inputs for test, all the outputs for test, a list with the variables and a list with variables types.

2.8 buildpredicate.py

This component is the heart of BTestBox and is responsible for generating and handling the predicate XML. Also, it conduct how to get the outputs for given inputs, manage the sets of the implementation.

When the instruction of a node is an operation call, *buildpredicate.py* calls *solveroc.py* to maneuver it. To check the predicate it call ProB through the component *callprob.py*

2.9 callprob.py

callprob.py is a component that will call ProB and manage it output.

Inputs: The predicate, the variables of the operation, ProB path. Output: A normalized ProB output, it get only the result of the significant variables.

2.10 createBTestSet.py

Component responsible for creating the copy of the user files and add the functions get and set. Also it create the following files based on the tests:

- TestSet_**COVERAGE_MACHINE_NAME**.mch;
- TestSet_**COVERAGE_IMPLEMENTATION_NAME**.imp;
- runTest_**COVERAGE_MACHINE_NAME**.mch;
- runTest_**COVERAGE_IMPLEMENTATION_NAME**.imp.

2.11 debugFile.py

File used to debug, with him was faster to change the inputs of BTestBox than using the debug of the tool.

2.12 HTMLgen.py

Component to generate HTML reports.

Inputs: The copy directory path, the percentage of coverage of each operation, the tests for each coverage (input and it corespondent output). Outputs: The HTML files.

1. **COVERAGE_IMPLEMENTATION_NAME**_FilesLib.html

2. reportText_**COVERAGE_IMPLEMENTATION_NAME**.html

3. report_**COVERAGE_IMPLEMENTATION_NAME**.html

2.13 solveroc.py

Component responsible for solving the operation calls, it call the pog to create an IBXML file in the bpd of the project and then call Substitution_Calculus_Pred.exe on Atelier-B binary folder to do the substitution in the predicate XML.

Inputs: The substitution, the predicate XML, the first node of the XML, the implementation BXML, the list of imported machine, operation name, implementation name, the possible mutable variables, inputs, fixed names (sets, constants), the outputs, variables that were assign during the execution, directory of bpd and the Atelier-B directory. Output: The predicate XML with the applied substitution, the variables that changed during the execution.

2.14 testTranslation.py

This component is responsible for the translation of the created files, generation of the main (not generated by Atelier-B translator), compiling and running it.

Inputs: Files for translation, compiler.