BTestBox

A Tool for Testing B Translators and Coverage of B Models.

Authors:

- Diego Oliveira (U. Sherbrooke-🇨🇦)
- Valério Medeiros Jr. (IFRN-🇧🇷)
- David Déharbe (ClearSy-🇫🇷)
- Martin Musicante (UFRN-🇧🇷)

Outline

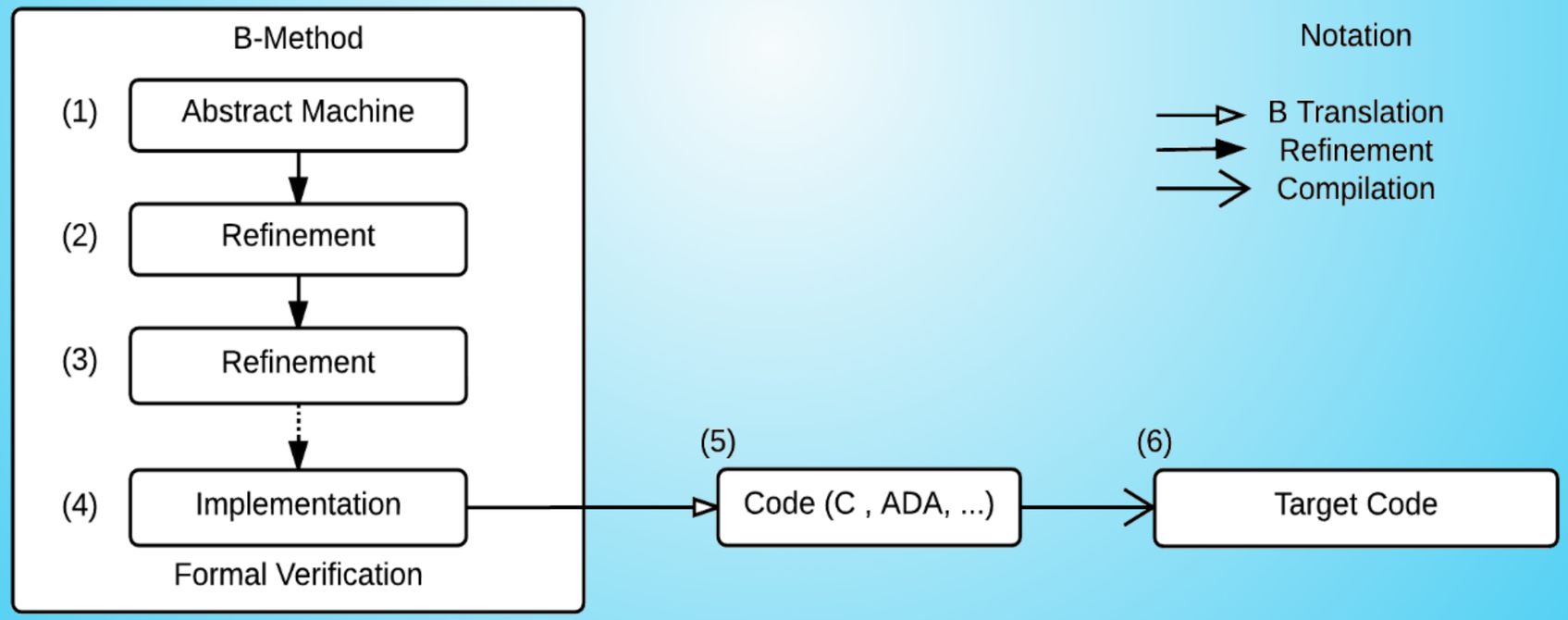
1. Introduction
2. Methodology of BTestBox
3. Example
4. Main results
5. Conclusion and future work

Introduction (1/2)

- An open source tool developed to support the validation of code generated.
  - Tests the translated code and assures dependability of the translation according to the coverage criteria.
    - Statement Coverage **(ST)**; Branch Coverage **(BC)**; Path Coverage **(PC)**; and others.
  - Extension developed for B Method.

Introduction - B Method (2/2)

- The B Method is a consolidated formal method that has been used for several years in critical systems.
- It is based on the abstract machine notation (AMN) and on the generalized substitution theory.
  - Supports modular modeling; each module specifies a software component in a different abstraction level.

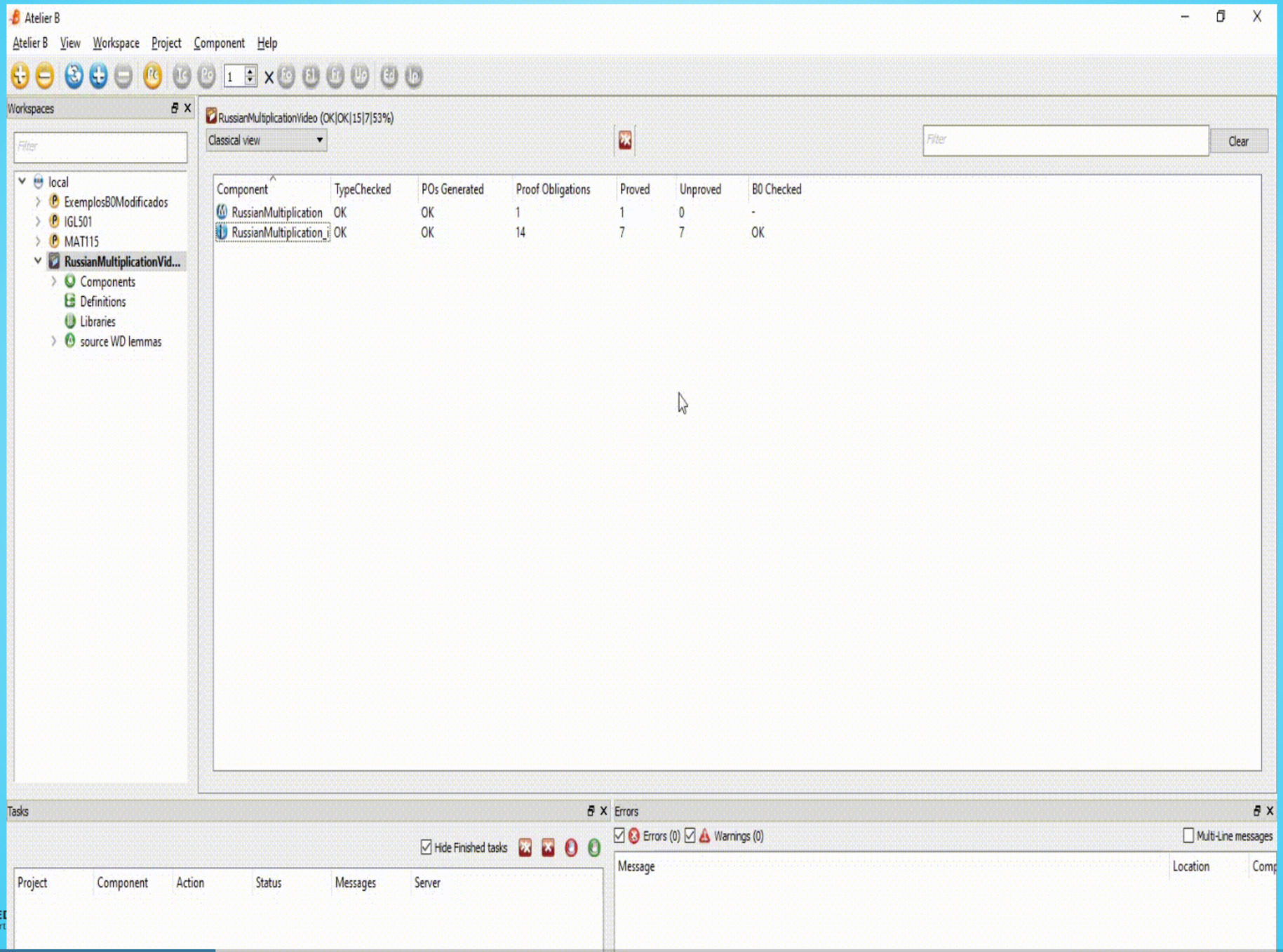


General motivation

- Errors in compilers silently introduce bugs [Leroy 2009].
- Eleven C compilers were identified with more than 325 errors [Yang et al. 2011].
- Code generators used in small communities have higher inherent technological risks than tools used in large scale [Stuermer et al. 2007].

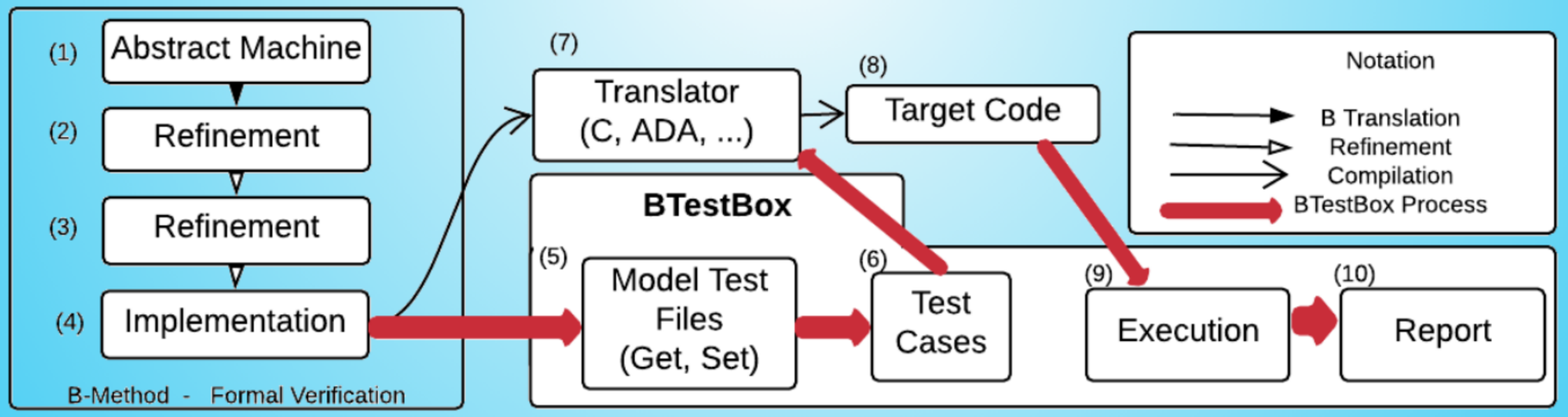
Motivation in B Method

- B code generators demand additional safety criteria.
  - They are used in critical applications.
- Testing techniques can be used as an instrument for an in-depth validation of the system.
  - A lower-cost complement of formal proofs.
  - Some certifications require the use of software testing techniques.


Atelier B
Atelier B View Workspace Project Component Help
Workspaces
Filter
local
ExemplosB0Modificados
IGL501
MAT115
RussianMultiplicationVid...
Components
Definitions
Libraries
source WD lemmas
RussianMultiplicationVideo (OK|OK|15|7|53%)
Classical view
Filter
Clear
Component
TypeChecked
POs Generated
Proof Obligations
Proved
Unproved
B0 Checked
RussianMultiplication
OK
OK
1
1
0
-
RussianMultiplication_i
OK
OK
14
7
7
OK
Tasks
Hide Finished tasks
Project
Component
Action
Status
Messages
Server
Errors
Errors (0)
Warnings (0)
Multi-Line messages
Message
Location

Creating the test cases

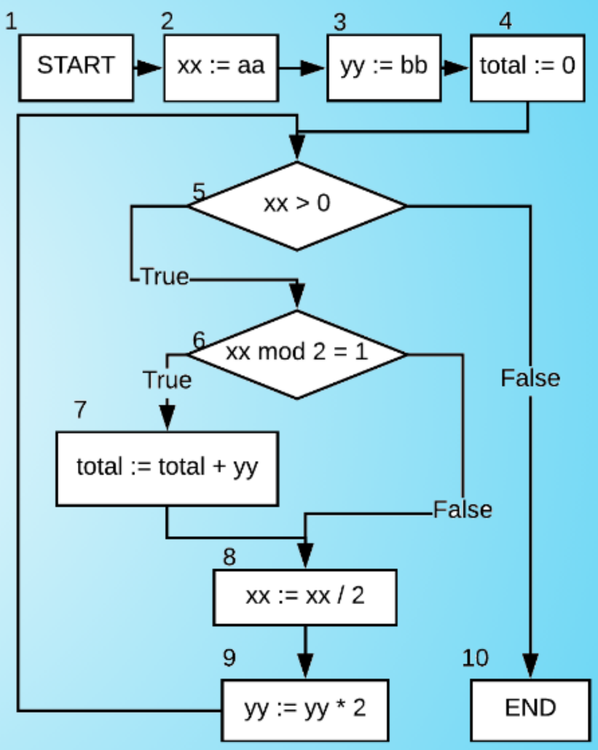
- BTestBox generates a control flow graph and uses Hoare logic to generate a predicate characterizing the possible values for the execution paths, according to the chosen criterion.
- The predicates are solved and the values of the input and output parameters are stored for each valid solution.
- The test components are translated and executed, and the metrics are reported.



Example - Control flow graph

RussMult

```
IMPLEMENTATION RussianMult_i
REFINES RussianMult
CONCRETE_VARIABLES xx,yy,total
INVARIANT xx ∈ N ∧ yy ∈ N ∧ total ∈ N
INITIALISATION
        xx,yy,total := 0,0,0
OPERATIONS
 RussMult(aa,bb) =
  xx:=aa; yy:=bb; total:=0;
  WHILE xx > 0 DO
        IF xx mod 2 = 1 THEN
                total := total + yy
        END;
        xx := xx / 2;
        yy := yy * 2
  INVARIANT xx ∈ N ∧
        total+xx*yy = aa * bb
  VARIANT xx
  END
 END
END
// The variant and invariant are necessary to prove
// the correctness of the operation, including its termin
```



- 1, 2, 3, 4, 5, 6, 7, 8, 9, 5, 10;
- 1, 2, 3, 4, 5, 6, 8, 9, 5, 10;
- 1, 2, 3, 4, 5, 10.

Generating Predicates

Based on Hoare triples ($\{P\}C\{Q\}$)

P the precondition

C the command

Q is the post-condition

- The predicates for each path generated by BTestBox are used to create test cases.
- Each criterion requires one different condition to be satisfied.

Branch coverage criterion - Test case 1

$$\exists(xx, yy, total).\ (xx > 0 \land xx \bmod 2 = 1 \land xx \in \mathbb{N} \land total + xx * yy = aa * bb)$$
$$\land xx : \mathbb{N} \land yy : \mathbb{N} \land total : \mathbb{N} \land aa : \mathbb{N} \land bb : \mathbb{N}$$

In the next step, it is necessary to find if the predicate is verified by at least one solution. Our tool uses the solver to find a suitable variable interpretation (if it exists).

A possible solution is (aa = 1, bb = 0, xx = 0, yy = 0, total = 0).

Branch coverage criterion - Test case 2

$$\exists(xx, yy, total).\ (xx \leq 0 \land xx \in N \land total + xx * yy = aa * bb)$$
$$\land xx : N \land yy : N \land total : N \land aa : N \land bb : N$$

A possible solution is (aa = 0, bb = 0, xx = 0, yy = 0, total = 0).

Branch coverage criterion - Test case 2

$$\exists(xx, yy, total).(xx \leq 0 \wedge xx \in N \wedge total + xx * yy = aa * bb)$$
$$\wedge xx : N \wedge yy : N \wedge total : N \wedge aa : N \wedge bb : N$$

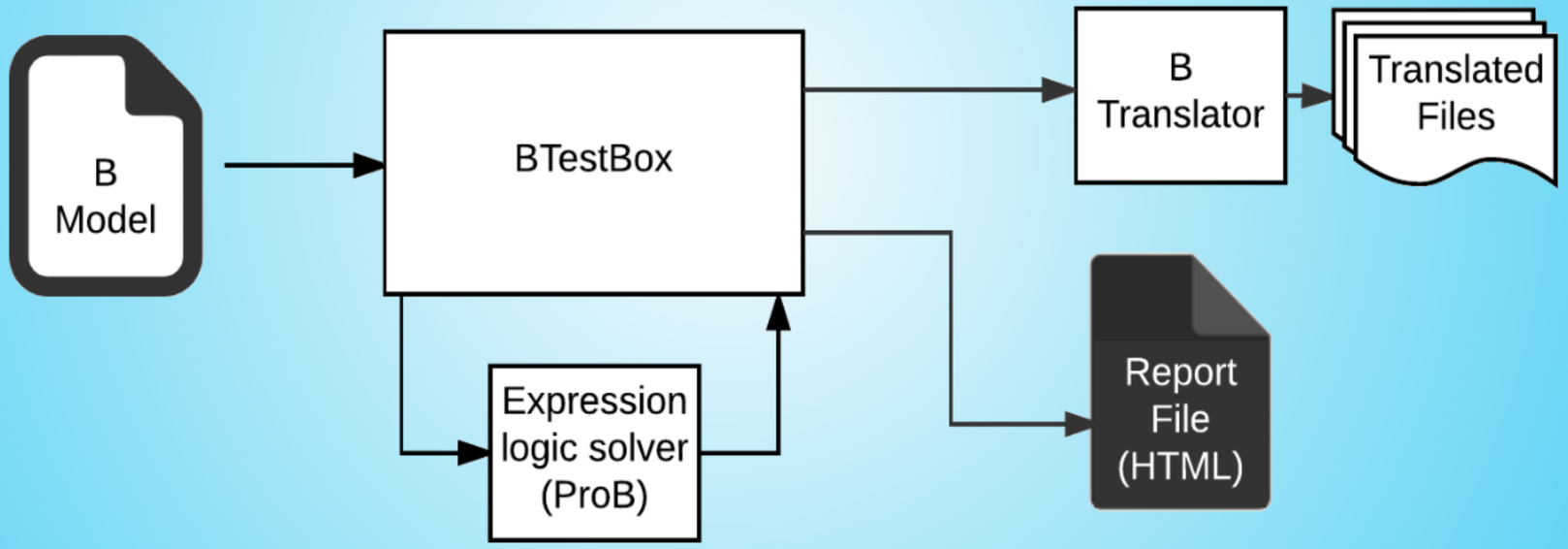
Other possible solution is (aa = 2, bb = 0, xx = 0, yy = 0, total = 0).

B Method with a verified implementation ensures that the loop ends.

This test case is not relevant!

Creating test case files

- Its improvements will provide interesting information and metrics about the code.



- Compatibility to any B Translators.
- Possible to use other solvers.

Experiments

BTestBox was tested for a variety of different B syntactic features, with the goal of assuring the correctness of the process and the functionality of the tool. Our tool ran more than 120 proved implementations.

- The Clauses group, with 14 examples.
- The Operation Call group, with 26 examples in different contexts.
- The Depth blocks group, with 89 examples.
- The Industrial project group, with one big example, is responsible for different contributions to the BTestBox.

Experiments with parallelization

- All of the examples shown can be easily executed in a leased cloud computer (with 20 dedicated vCPUs for one hour) costing less than one US dollar with similar results.
- The time is reduced approximately proportionally when we use a computer with more processing cores.

Component	Quantity of operations	Execution time with 4 cores	Execution time with 20 cores	Reduced time for evaluation
COMP_1seq1	39	587 s	38 s	93%
COMP_2seq1	199	3102 s	176 s	94%
COMP_3seq1	999	18214 s	2286 s	87%

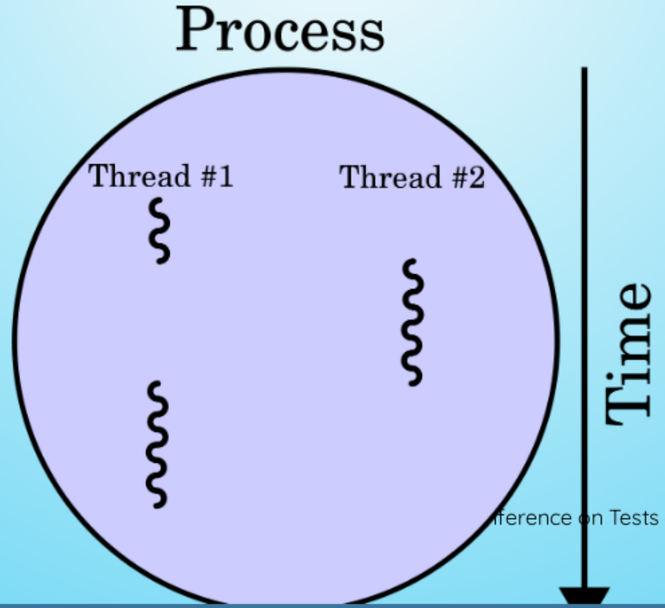
Related work

- Tools developed in our research group:
  - BETA relies on input space partitioning and logical coverage criteria to generate unit tests from abstract B machines. The tool automates all steps of the test generation process.
  - BTestBox is completely automatic in all steps of the test generation process based directly on implementation modules which are a closer representation of the actual software. Another difference is that BETA is focused on unit testing. BTestBox tests is focused on the entire module and its functions.

Conclusions and final considerations

The main differences between the previous and new versions are:

- (i) inclusion of several code coverage criteria, the previous version supported only Statement Coverage;
- (ii) new HTML interface, including reports generated for each coverage criterion;
- (iii) better support to new case studies both academic and industrial. This support is based on the use of parallelization techniques, in order to improve scalability.



Conclusions and final considerations

The parallelisation techniques applied reduced the process time and extended the power to generate tests for larger samples. We also configured the tool to run it on a remote computer with more processor cores. This improvement required changes that made our tool compatible with the most common operating systems (Windows, OS X and Linux).

All recent advances presented here are encouraging, showing performance numbers that are one order of magnitude better than for the previous version of the tool.

Questions?

Doubts and comments.

Thank you for your attention!

Presentation avaliable at: valeriomedeiros.github.io/2019_TAP

Acknowledgement

The work is partly supported by IFRN, UFRN, ClearSy and High Performance Computing Center at UFRN (NPAD). This study was financed in part by the Brazil (CAPES).

Extras

Export to PDF (Click here and CRTL+P)

Export to PDF with notes (Click here and CRTL+P)

Extras

Export to PDF (Click here and CRTL+P)

Export to PDF with notes (Click here and CRTL+P)

Extras

Export to PDF (Click here and CRTL+P)

Export to PDF with notes (Click here and CRTL+P)

Extras

Export to PDF (Click here and CRTL+P)

Export to PDF with notes (Click here and CRTL+P)