Verifying Code Generation Tools for the B-Method Using Tests: a Case Study

Anamaria M. Moreira — Federal University of Rio de Janeiro

David Déharbe — Federal University of Rio Grande do Norte

Ernesto C. B. de Matos

João B. Souza Neto

Cleverton Hentz — Federal Institute of Education, Science and Technology of Rio Grande do Norte

Valério de Medeiros Jr.

Introduction

- Verification of compilers and code generators is a complex task
- Here we present a **case study where two code generation tools were verified using tests** (C4B and b2llvm):
  - Overview of our **testing strategy**
  - The **tools we used**
  - The **results obtained**

Related Work

- Most of the work on verifying code generators falls into one of the three categories:
  - **Formal verification:** focuses on techniques that prove a code generator to be correct for every input model
  - **Test case generation based on grammars:** produces test inputs for a code generator based on a grammar specification
  - **Translation validation:** shows the correct translation of individual inputs, checking for correctness in each output of the code generator individually

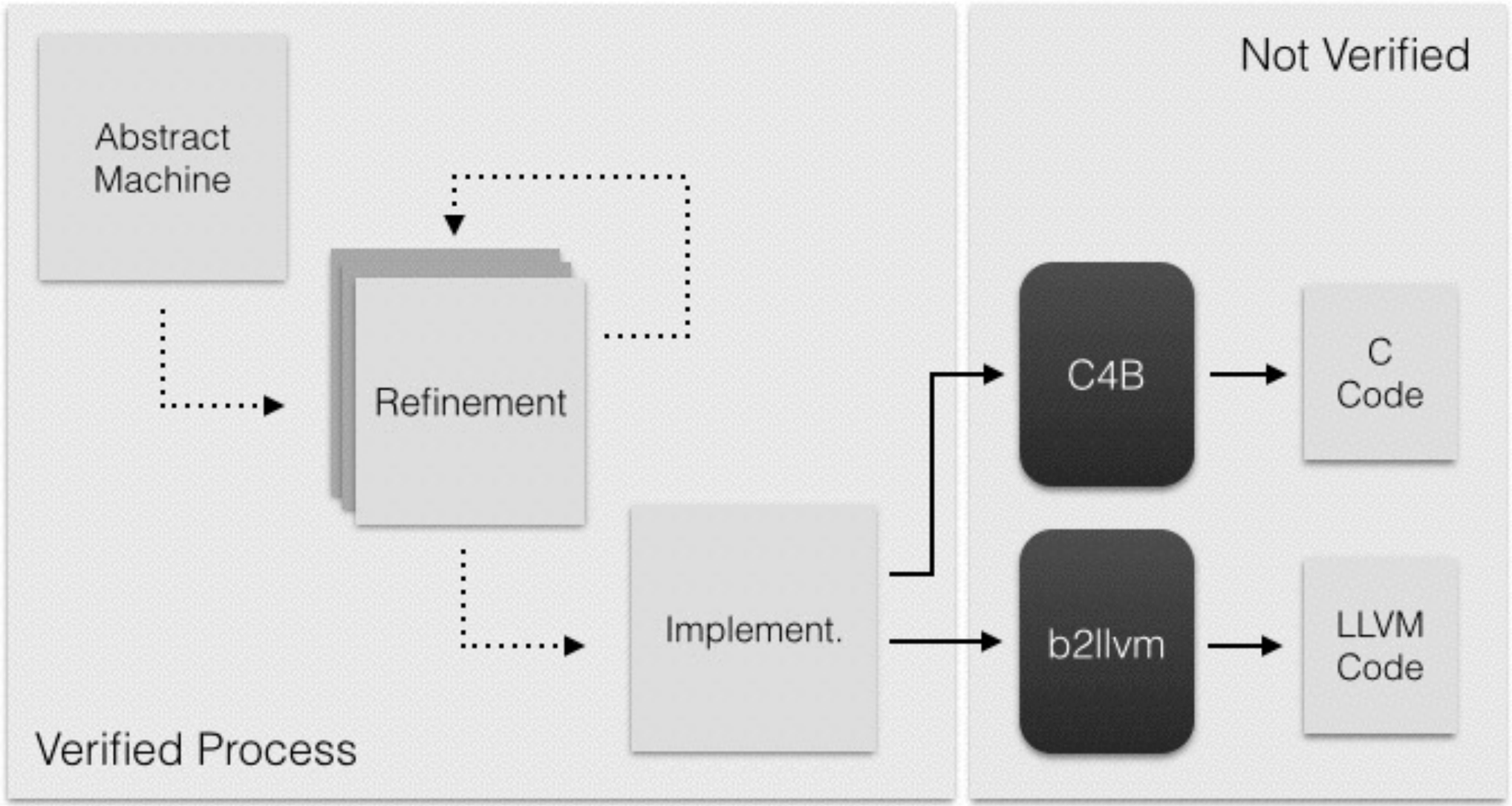
Related Work

- Most of the work on verifying code generators falls into one of the three categories:
  - **Formal verification:** focuses on techniques that prove a code generator to be correct for every input model
  - **Test case generation based on grammars:** produces test inputs for a code generator based on a grammar specification
  - **Translation validation:** shows the correct translation of individual inputs, checking for correctness in each output of the code generator individually

Background

- The **B-Method is a formal method**
- It uses concepts of **first order logic**, **set theory** and **integer arithmetics** to specify **abstract state machines** that represent software behaviour
- The model can be **verified using proof obligations** to ensure its consistence
- It provides a **refinement mechanism**

Background


Abstract
Machine
Refinement
Implement.
Not Verified
C4B
C
Code
b2llvm
LLVM
Code
Verified Process

Background

- Tools verified in the case study:
  - **C4B**
    - Code generator **distributed** and integrated **with the Atelier B IDE**
    - AtelierB is a **consolidated tool that is used in many projects** both in the academia and in the industry
    - C4B **automatically produces C code from B implementations**

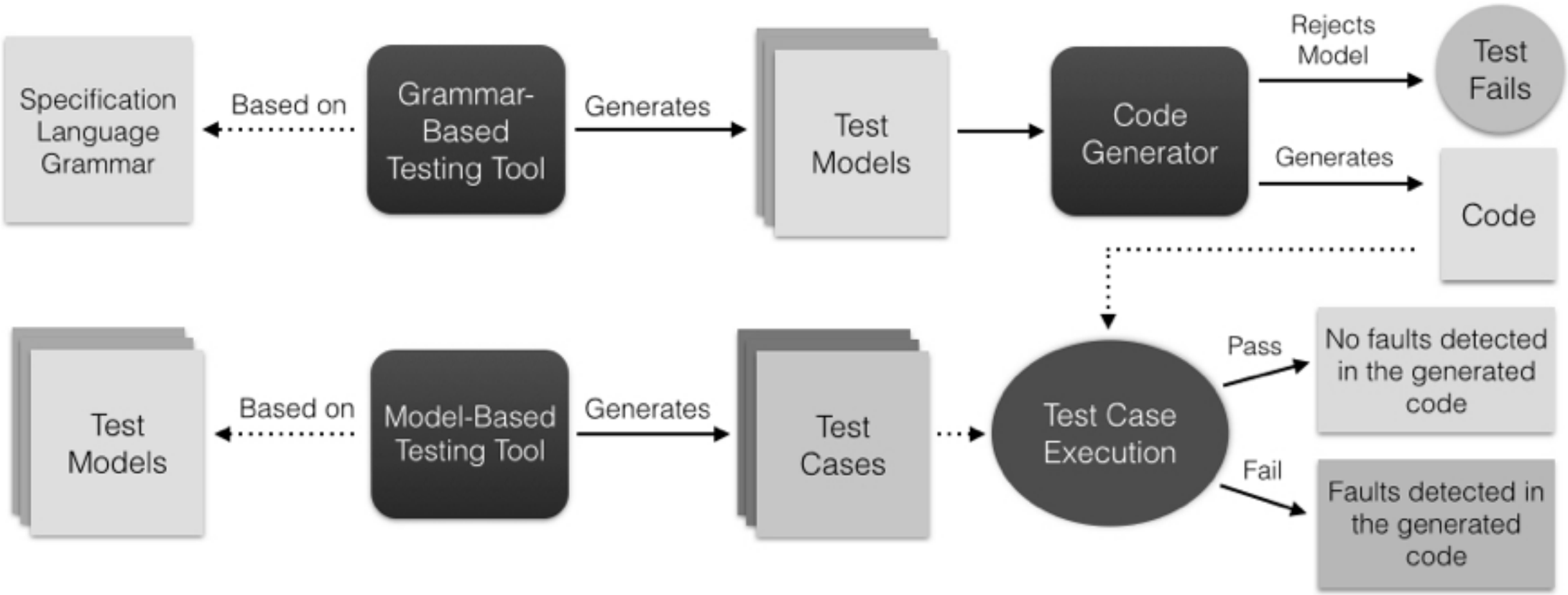
Background

- Tools verified in the case study:
  - **b2llvm**
    - A **compiler for B** implementations **that generates LLVM code**
    - It is **currently under development**
    - **Supports part of the B notation**

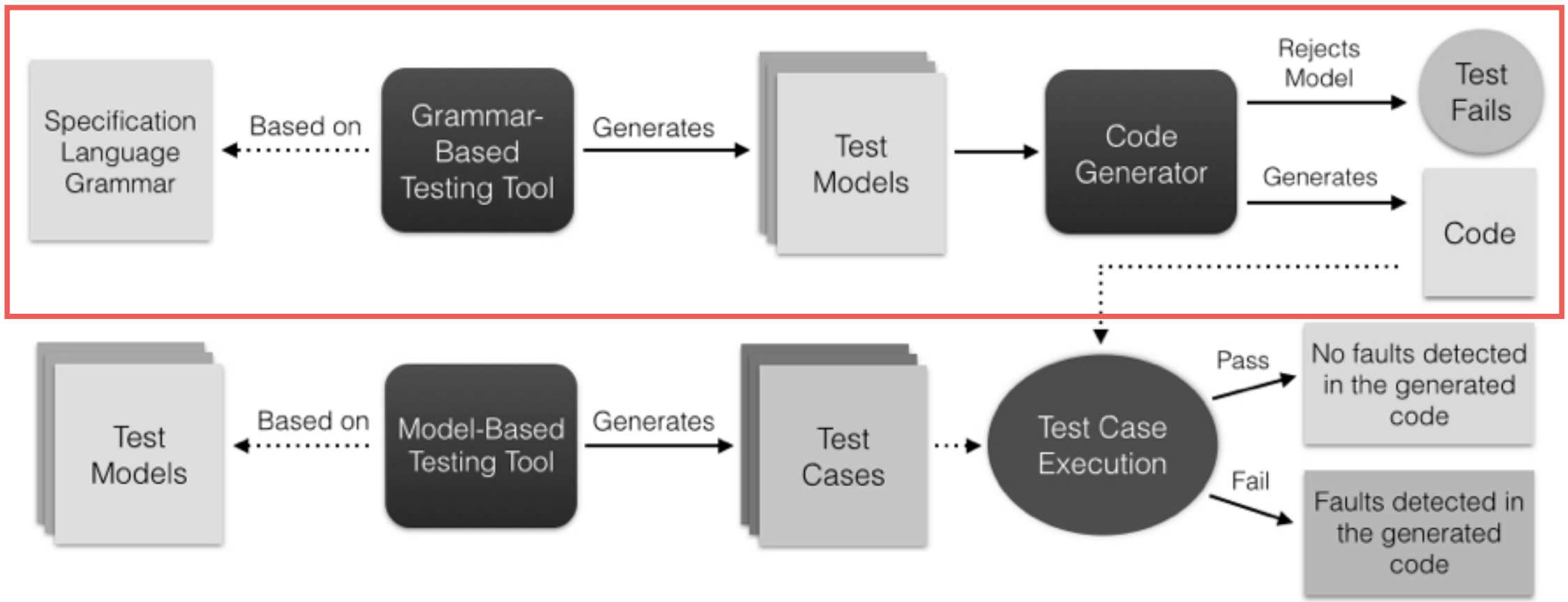
Testing Strategy

- The two main questions that we want to address are:
  - **Is the tool capable of generating code for the wide range of inputs it can receive?**
  - **Does the code generated by the code generation tool comply with the input model?**
- To answer the first question we used the **Grammar-Based Testing**
- To answer the second question we used **Model-Based Testing**

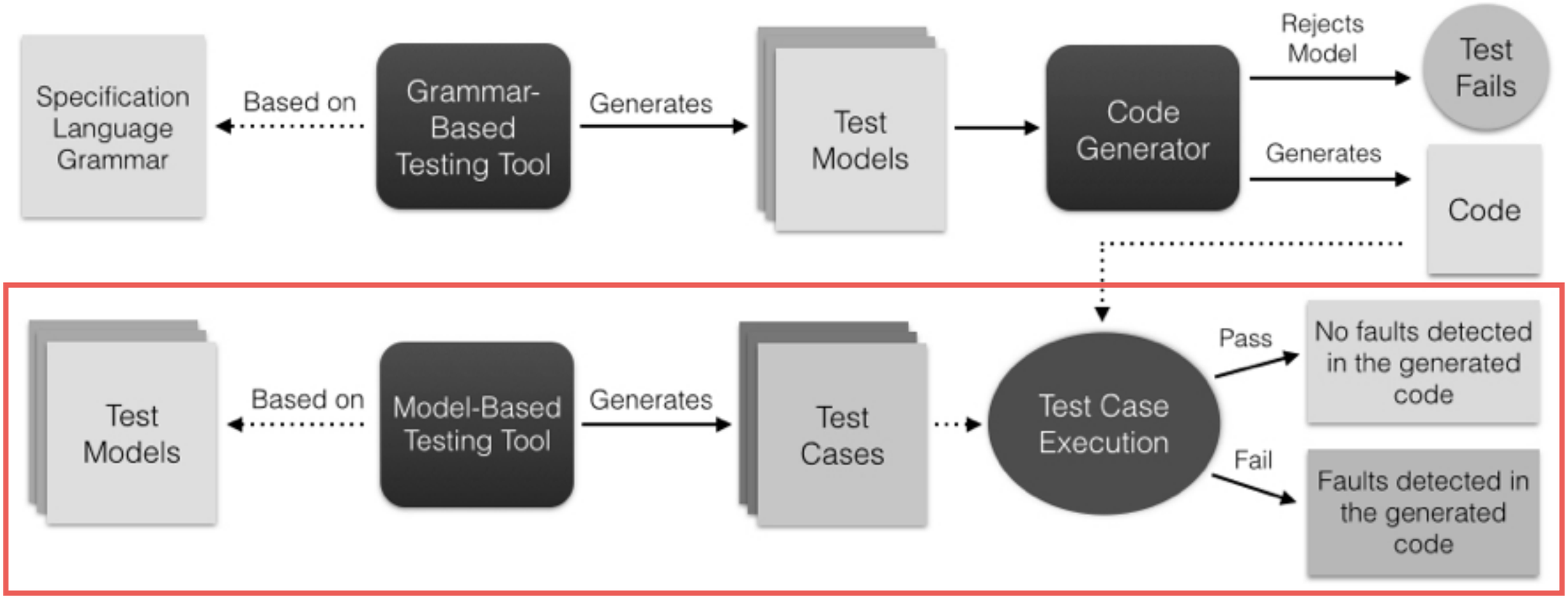
Testing Strategy



Testing Strategy



Testing Strategy



Grammar-based Testing

- The **tests are generated based on grammar descriptions**
- The grammar **describes the input language accepted by the code generator**
- **To restrict the number of test inputs** generated **we use grammar-based coverage criteria**, such as: **Terminal Coverage**, **Production Coverage**, and **Context-Dependent Branch Coverage**

LGen

- A sentence generator based on syntax description
- Receives as input a grammar described using the EBNF (Extended BNF) notation
- **Generates a set of sentences** of the language **corresponding to the input grammar**
- **Uses coverage criteria to restrict the set of sentences**

Model-Based Testing

- We generate **unit tests from the same input models used to generate code**

- The **generated tests are executed** on the generated code **to find discrepancies between the input model and the implementation** (they check if they have the same behaviour for a given test input)

- The **criteria** used to generate the test cases are: **Equivalent Classes**, **Boundary Value Analysis**, **Active Clause Coverage** and **Combinatorial Clause Coverage**

BETA

- A tool supported approach to generate unit tests from B specifications

- Receives as input an abstract B machine and generates test **cases for the implementation** of the model

- **Supports Input Space Partitioning and Logical Coverage** testing criteria to generate test cases

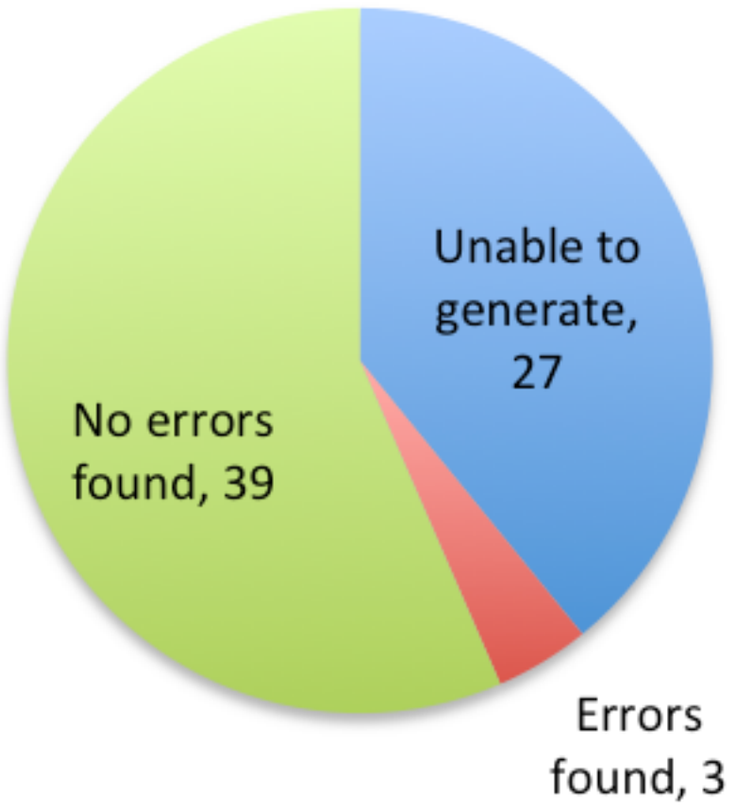
- Generates **test case specifications** and **partial executable test scripts**

Results

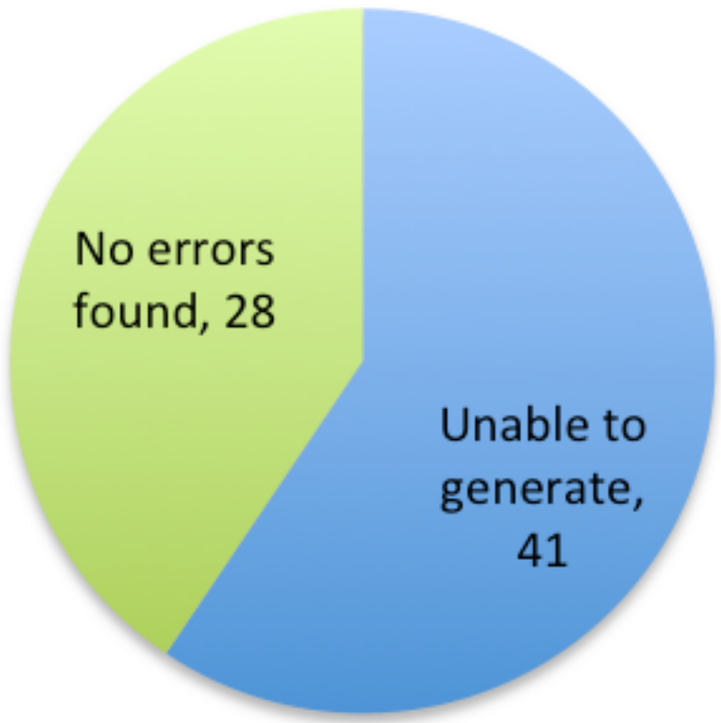
Grammar-based Testing

- LGen generated 69 test models based on the B grammar definition using production coverage.

C4B



b2llvm



Results

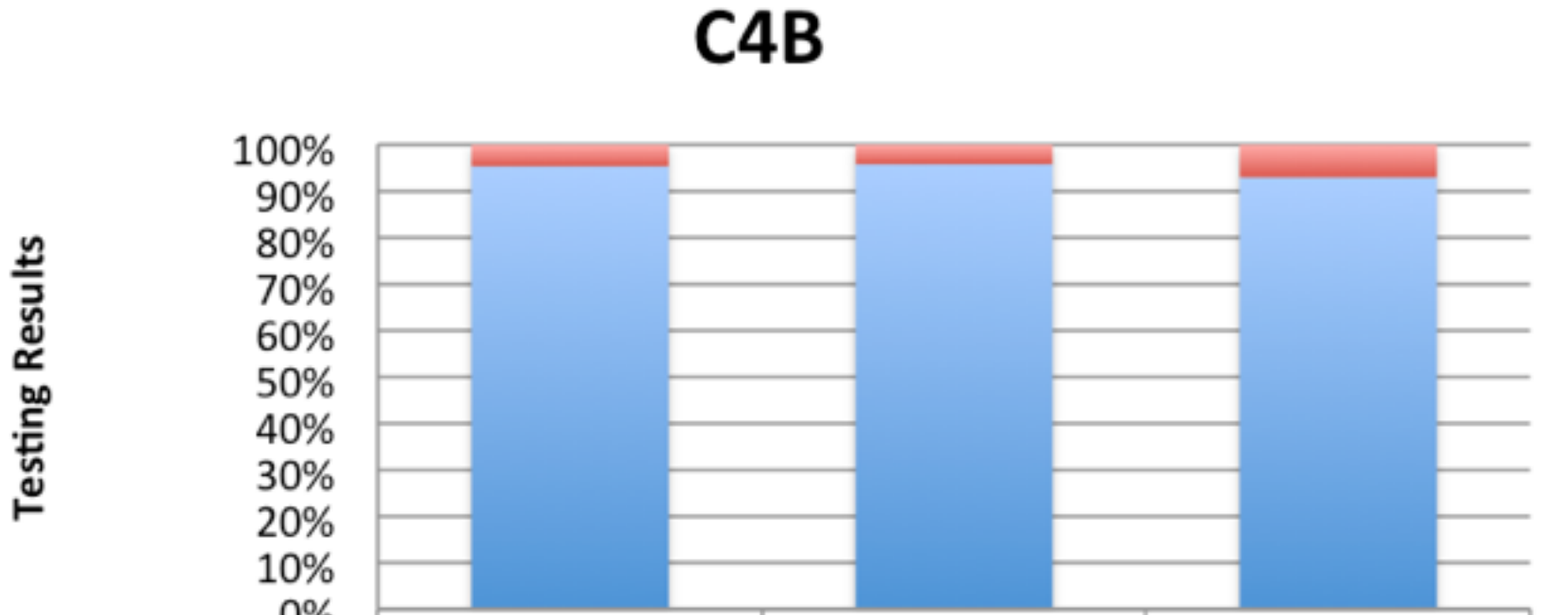
Grammar-based Testing

- C4B rejected 27 test models because it didn't support some of the syntactic constructions used

- b2llvm rejected 7 test models for the same reason and 34 due to bugs in its code

Results

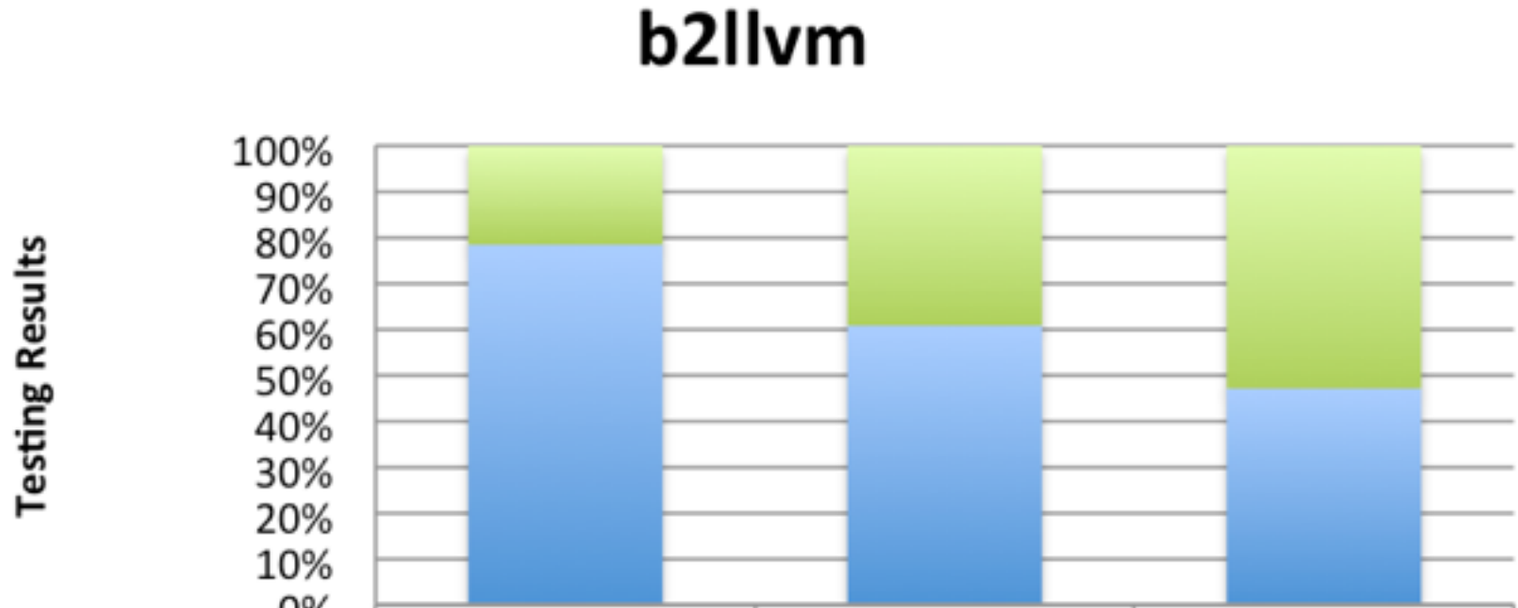
Model-based Testing



	EC/BV	ACC	CoC
Not performed	0	0	0
Failed	3	5	5
Passed	62	115	67

Results

Model-based Testing



	EC/BV	ACC	CoC
Not performed	14	47	38
Failed	0	0	0
Passed	51	73	34

Results
Model-based Testing

- The tests that failed for C4B were related with modularisation of the code. The generated code did not import the necessary modules

- Many tests for b2llvm were not performed because of the lack of support for some syntactic constructs

  - In this case, the tests generated were used to guide the development of missing features in b2llvm

Conclusions

- We presented a case study where we verified two code generations tools for the B-Method using tests (a combination of grammar-based testing and model-based testing)

- We gave an overview of our testing strategy and the tools used to support it

- With moderate effort, we were able to find important problems and missing features on both code generation tools

- The problems encountered during the case study were reported to the tool developers and will contribute to improve the reliability of C4B and b2llvm.

- We believe that the testing strategy could be used to test other code generation tools (as long as you have tools to support it)

Questions?

anamaria@dcc.ufrj.br

david@dimap.ufrn.br

{chentz, ernestocid, jbsneto, valerio}@ppgsc.ufrn.br