TestCoCa: Test-Suite Coverage Calculator

Martin Ergang, Marek Trtík

Introduction

- **TestCoCa** is a tool calculating **coverage** of particular location(s) in a C program. Locations are either
  - **Program branchings** (Computation of branch coverage).
  - **The entry block of a function** (Reachability of a specific error).
- The coverage is computed from a **given test-suite**.
  - TestCoCa executes all tests in the suite and accumulates the coverage of locations visited during executions of tests.
  - The tests are expected to be in the XML-based Test-Comp's format.
- In terms of Test-Comp, TestCoCa is a **validator**.
  - A validator measures a performance of competing tools.

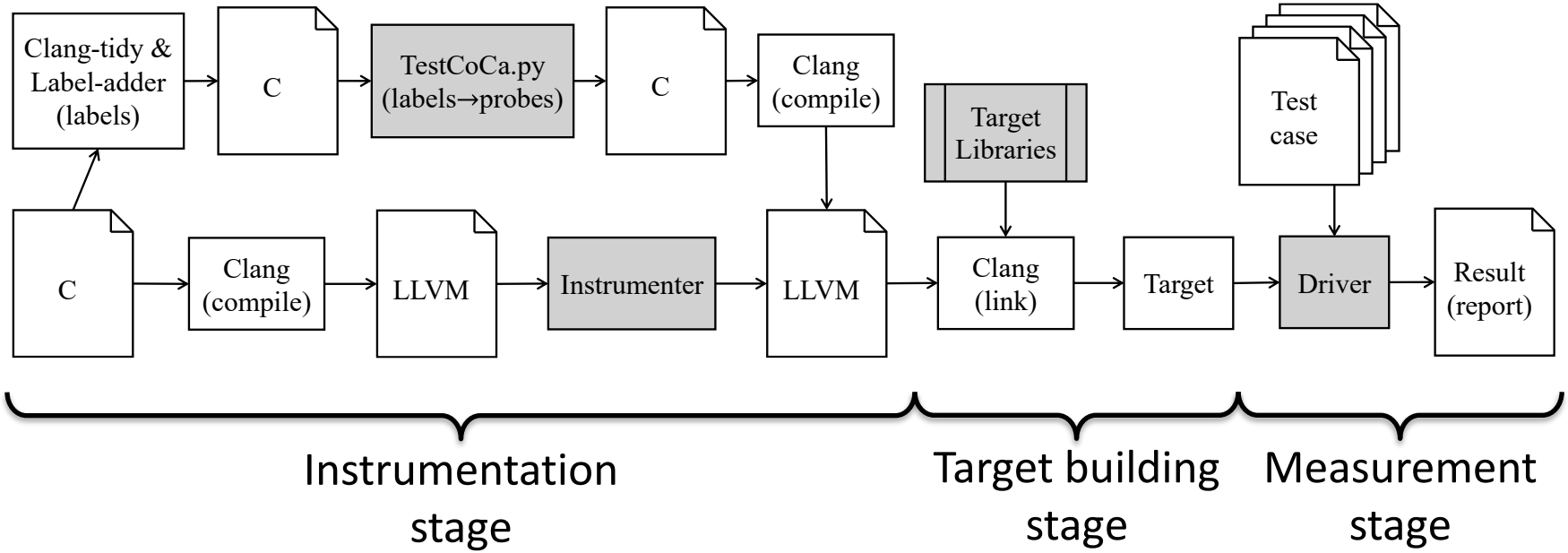
Software Architecture

- TestCoCa consists of **three main components**:
  - Instrumenter
  - Target Libraries
  - Driver
- The components are used by a Python script **TestCoCa.py** implementing the actual **analysis pipeline** of the tool.
- The pipeline consists of **three stages**:
  - Instrumentation stage
  - Target building stage
  - Measurement stage

Pipeline Stages

Observe there are **two paths** in the **instrumentation stage**.

Components in **gray** are from **TestCoCa's repository**.



Instrumentation Stage

- The instrumentation is performed by the **Instrumenter** component.
- The Instrumenter can operate in **two modes** based on the goal specified in the property file:
  - **Coverage**
    - Every basic block in LLVM bitcode, which is a target of some **conditional jump**, is instrumented.
    - This mode implements the **short-circuit evaluation** of branching conditions as specified by the **semantics of the C language**.
  - **Error function**
    - The tool instruments the **entry basic block** of the given "error" function, whose reachability should be checked.

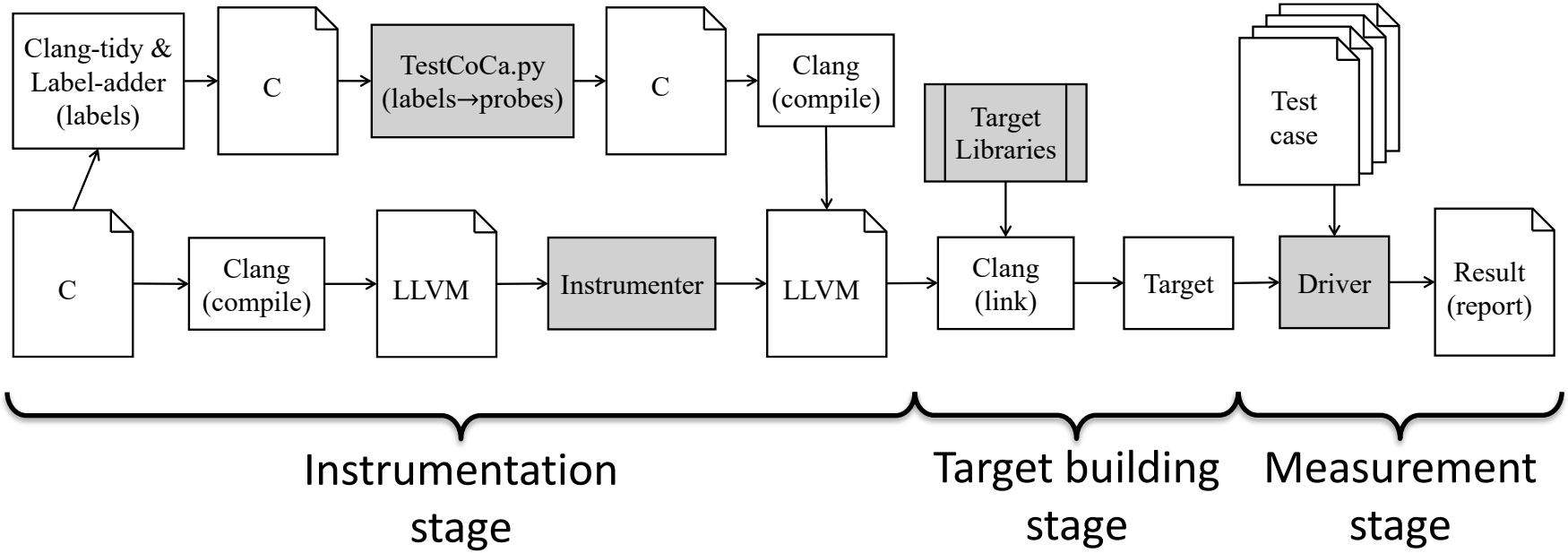
Instrumentation Stage

- TestCov treats all conditions (including conjunctions and disjunctions) atomically, i.e., **short-circuit evaluation is not applied**.
- In order to synchronize the instrumentation with TestCov, TestCoCa provides an **alternative instrumentation** path for this goal.
  - It is enabled by options --format and --testcomp.

Data-Flow

Observe there are **two paths** in the **instrumentation stage**.

Components in **gray** are from **TestCoCa's repository**.



Coverage Computation Stage

- In the last stage, the Driver executes the Target program for each test case in the test-suite passed to the tool.
  - **Shared memory** is created between the process of the Driver and the Target.
  - The Driver first writes the input data from the test case into the shared memory.
  - Then the Target reads the input data via calls to __VERIFIER_nondet_* functions during its execution.
  - Executed probes save their information to the shared memory.
  - Once the execution of the Target ends the Driver reads the saved information from the shared memory and updates overall statistics of either branch coverage or the reachability state of the “error” function.

Strengths

- TestCoCa's strengths are these **two abilities**:
  - **Ability to survive crash of the Target**.
    - The Driver can still read and use the information written into the shared memory by probes before the crash.
  - **Ability to deal with long execution times** of some test cases.
    - Test cases are executed repeatedly in waves.
    - Each wave defines a "wave timeout" (same for each test case in the wave).
    - The Driver force-terminates longer executions.
    - The timeout of the first wave is the shortest and it increases with further waves.
    - The first wave comprises all test cases.
    - If the Target's execution for a test case ends before the wave timeout, then the test case does not appear in future waves.

Weakness

- TestCoCa's weakness is that it **does not guarantee the correctness** of computed results (coverage).
  - Target's execution may accidentally overwrite some parts of the shared memory, e.g., due to wrong pointer arithmetic.
  - The Driver thus checks for consistency of information read from the shared memory and discards all entries which must be corrupted.
    - Also prints warnings.
    - But not all possible corruptions are detected.

Tool Availability

- TestCoCa is available in a binary form at Zenodo:

  https://doi.org/10.5281/zenodo.17819536
- And the source code is available at GitHub:

  https://github.com/staticafi/TestCoCa
  - The source code is released under the open-source zlib license.

- *Acknowledgement*: We would like to thank Prof. Dr. Dirk Beyer for aiding the development of the tool by performing evaluations on the Test-Comp computing infrastructure.