


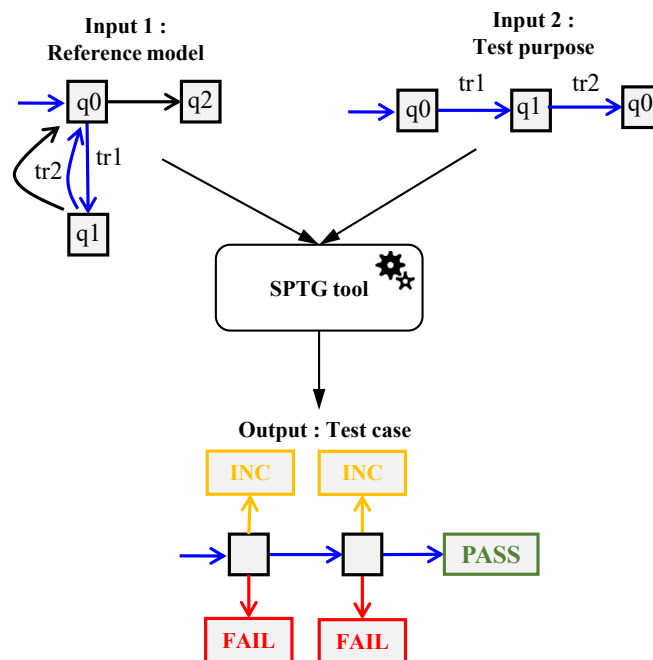


SPTG: Symbolic Path-Guided Test Case Generator

Table of content

1. [SPTG overview](#)
2. [Quick start with SPTG](#)
3. SPTG tutorials
 -  [Tutorial on model specification](#)
 -  [Tutorial on test case generation](#)
 -  [Tutorial on test purpose selection](#)

SPTG overview



****Figure 1:**** Schematic view of SPTG showing the model automaton with a selected test purpose (blue path) and the generated test case automaton with terminal verdict states.

SPTG is a model-based test generation tool that automatically produces **conformance deterministic test cases** from system models combining both **data** and **timing constraints**. As shown in **Figure 1**, SPTG takes an **automaton model** and a **test purpose**, i.e., a path of the model, and generates the corresponding **test case automaton** with **verdict states** PASS, FAIL, INC (for inconclusive).

It relies on **path-guided symbolic execution**, which explores the input path and builds **symbolic constraints** over inputs and timing. SPTG embeds the **Z3 SMT solver**, which is used to check the **satisfiability of path conditions** along the main test purpose path and its **immediate divergent paths**, as well as to ensure determinism. Infeasible branches, inconsistent with the test purpose, are pruned early during symbolic exploration, avoiding dead paths that correspond to excluded behaviors.

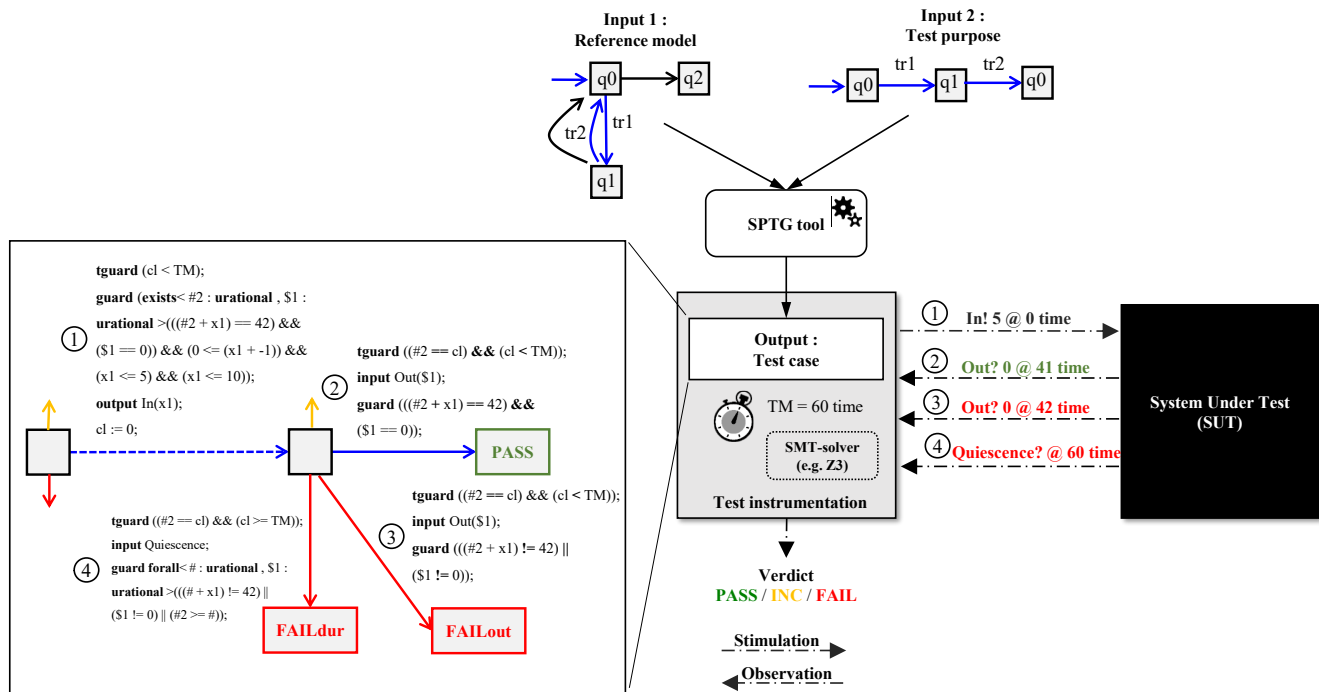


Figure 2: Execution of a generated test case against the System Under Test (SUT) with verdicts determined at runtime.

Figure 2 illustrates the execution phase, where the generated test case interacts with the **System Under Test (SUT)**. During execution, **Z3** is used to solve the **stimulation conditions (guards)**, determining the inputs and timings to apply. Test case transitions are controlled by a clock cl , which satisfies $cl < TM$, where TM is the maximal waiting time before either applying a stimulation or observing an output. Quiescence, i.e., the expected absence of output, is detected when $cl \geq TM$, indicating that the system remains silent as anticipated. This timing mechanism, combined with quiescence detection, ensures the test case is implementable in a real-time setting. Additionally, **Z3** checks that the **observed outputs** and their **timings** satisfy the corresponding observation conditions, after which verdicts are assigned.

Applications

- **Model-Based Testing (MBT)** of systems with combined data and timing behaviors.
- **Offline generation** of efficient and deterministic test cases from formal models.
- **Teaching and demonstration** of symbolic execution and model-based test generation principles.

References

SPTG implements the **symbolic path-guided test generation approach** developed in: <https://doi.org/10.1016/j.scico.2025.103285> (Open Access)

Quick start with SPTG

SPTG directory Structure:

- [examples](#)
- [tutorials](#)
- [src](#)
- [third-party](#)

- Release

```
cd PATH_TO_SPTG/examples/example02_dummy/  
run-sptg.sh
```

This workflow instructs SPTG to generate a **test case** from the **reference system model** (`example02_dummy.xlia`) using the **sequence of transitions** `tr1`; `tr2` that define the **test purpose**.

Note:
The input reference model automaton is encoded in the **XLIA language**, the input language of the **Diversity** symbolic execution platform. **SPTG** extends Diversity with dedicated functionality for symbolic path-guided test generation. See [model_specification](#) for more details.

SPTG generates the resulting **test case automaton** in the following formats:

- specification language **XLIA** the same language used to express the reference model (`PATH_TO_SPTG/examples/example02_dummy/output/testcase.xlia`)
- in graphical format **PlantUML** (`PATH_TO_SPTG/examples/example02_dummy/output/testcase.puml`).
- In addition, SPTG generates the test case automaton in JSON format with guards expressed in SMT-LIB format (`PATH_TO_SPTG/examples/example02_dummy/output/testcase_smt.json`).

You can visualize `.puml` files using [PlantUML](#) or the online tool [PlantText](#).

You can convert a file `.puml` to a file `.svg` (see the [PlantUML Conversion Guide](#)).

Description	Content
Input 1: Timed symbolic automaton : Reference system model	<pre>stateDiagram-v2 [*] --> q0 q0 --> q1 : input ln(x); guard ((1 <= x) && (x <= 10)); sum := (sum + x); cl := 0; q1 --> q0 : guard ((x <= 5) && (cl == (42 + (- x)))); output Out(0); q0 --> q2 : guard ((x > 5) && (cl == (42 + (- x)))); output Out(x); q2 --> q0 : guard (sum >= 15); output Done;</pre>
Input 2: Sequence of transitions (path) : Test purpose	<code>tr1; tr2</code>

Compilation Instructions

```
cd PATH TO SPTG/org.eclipse.efm.symbex/Release/
```

```
make all -j4
```

```
cp -f sptg.exe ../../bin/sptg.exe
```

```
mv ../../bin/sptg.exe ../../bin/sptg old.exe
```

PlantUML: PUML to SVG Conversion Guide

Prerequisites

1. **Java Runtime Environment (JRE):** Required to execute PlantUML.
2. **PlantUML JAR File:** The standalone application.

1. Download PlantUML

Get the latest stable release of `plantuml.jar` from the official github site:

👉 <https://github.com/plantuml/plantuml/releases>

2. Conversion Command

Navigate to the folder containing both `plantuml.jar` and your `.puml` file.

Use the `-tsvg` flag to generate an SVG image:

Command	Action
<code>java -jar plantuml.jar -tsvg yourfile.puml</code>	Converts the input file (<code>.puml</code>) to an SVG output (<code>.svg</code>).

Example

```
# Generates 'MyDiagram.svg'  
java -jar plantuml.jar -tsvg MyDiagram.puml
```