SPTG: Symbolic Path-Guided Test Case Generator

Table of content

SPTG overview

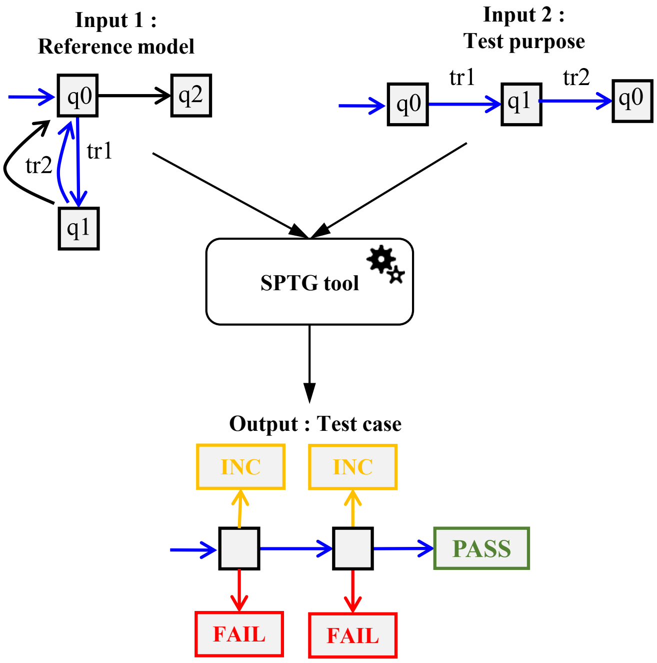
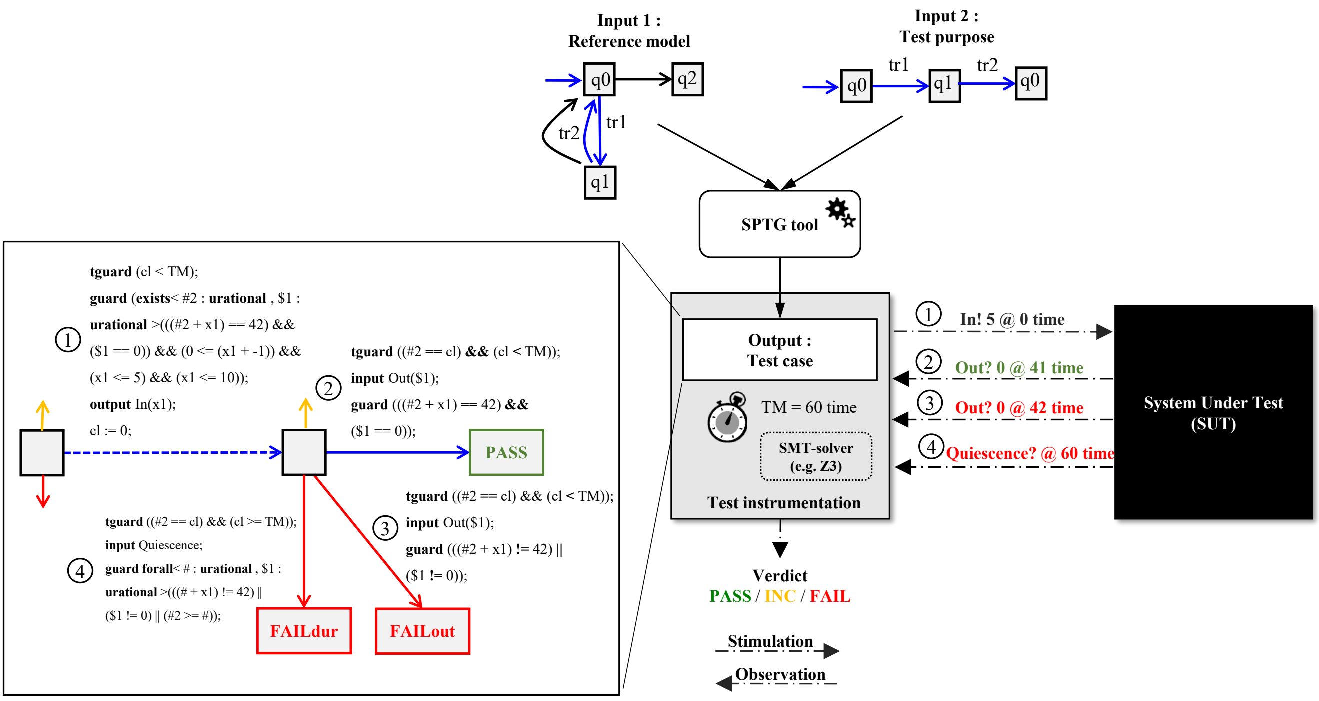


Figure 1: Schematic view of SPTG showing the model automaton with a selected test purpose (blue path) and the generated test case automaton with terminal verdict states.

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.



```
① tguard (cl < TM);
guard (exists< #2 : urational , $1 :
urational >(((#2 + x1) == 42) &&
($1 == 0)) && (0 <= (x1 + -1)) &&
(x1 <= 5) && (x1 <= 10));
output In(x1);
cl := 0;

② tguard ((#2 == cl) && (cl < TM));
input Out($1);
guard (((#2 + x1) == 42) &&
($1 == 0));

③ tguard ((#2 == cl) && (cl < TM));
input Out($1);
guard (((#2 + x1) != 42) ||
($1 != 0));

④ tguard ((#2 == cl) && (cl >= TM));
input Quiescence;
guard forall< # : urational , $1 :
urational >(((# + x1) != 42) ||
($1 != 0) || (#2 >= #));
```

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 >= 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** developped 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**.

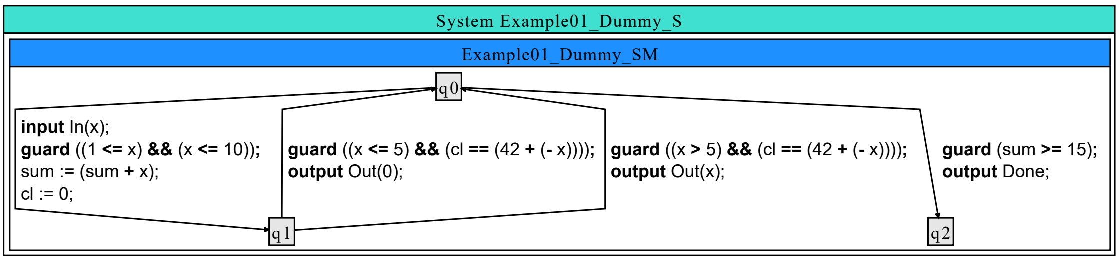
> **Note:**
> The input reference model automaton is encoded in the **XLIA language**, the input language of the **Diversity** symbolic execution platform. **SPTG** extends Diversity with dedicated functionality for symbolic path-guided test generation.See model_specification for more details.

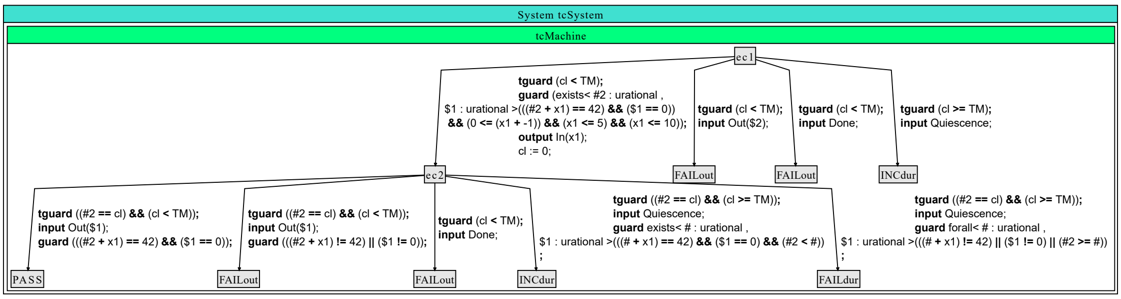
SPTG generates the resulting **test case automaton** in the follwoing formats:

- specification langauge **XLIA** the same langauge 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*	 System Example01_Dummy_S Example01_Dummy_SM q0 q1 q2 **input** In(x); **guard** ((1 <= x) && (x <= 10)); sum := (sum + x); cl := 0; **guard** ((x <= 5) && (cl == (42 + (- x)))); **output** Out(0); **guard** ((x > 5) && (cl == (42 + (- x)))); **output** Out(x); **guard** (sum >= 15); **output** Done;
Input 2: *Sequence of transitions (path) : Test purpose*	`tr1; tr2`

Description	Content
Output: *Deterministic timed symbolic automaton : Generated test case*	

System tcSystem
tcMachine
ec1
tguard (cl < TM);
guard (exists< #2 : urational ,
$1 : urational >(((#2 + x1) == 42) && ($1 == 0))
&& (0 <= (x1 + -1)) && (x1 <= 5) && (x1 <= 10));
output In(x1);
cl := 0;
tguard (cl < TM);
input Out($2);
tguard (cl < TM);
input Done;
tguard (cl >= TM);
input Quiescence;
FAILout
FAILout
INCdur
ec2
tguard ((#2 == cl) && (cl < TM));
input Out($1);
guard (((#2 + x1) == 42) && ($1 == 0));
tguard ((#2 == cl) && (cl < TM));
input Out($1);
guard (((#2 + x1) != 42) || ($1 != 0));
tguard (cl < TM);
input Done;
tguard ((#2 == cl) && (cl >= TM));
input Quiescence;
guard exists< # : urational ,
$1 : urational >(((# + x1) == 42) && ($1 == 0) && (#2 < #))
;
tguard ((#2 == cl) && (cl >= TM));
input Quiescence;
guard forall< # : urational ,
$1 : urational >(((# + x1) != 42) || ($1 != 0) || (#2 >= #))
;
PASS
FAILout
FAILout
INCdur
FAILdur

Compilation Instructions

To compile SPTG, navigate to the `Release` directory of the `org.eclipse.efm.symbex` module:

```
cd PATH_TO_SPTG/org.eclipse.efm.symbex/Release/
```

Then build the project:

```
make all -j4
```

During compilation, the process automatically overwrites the existing `sptg.exe` in the `bin` directory using:

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

If you wish to preserve the existing executable, rename it before compilation as follows:

```
mv ../../bin/sptg.exe ../../bin/sptg_old.exe
```

PlantUML: PUML to SVG Conversion Guide

A quick reference for converting `.puml` files to `.svg` images via the command line.

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
`java -jar plantuml.jar -tsvg yourfile.puml`	Converts the input file (`.puml`) to an SVG output (`.svg`).

Example

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