Learning DOTAs via mutation testing

Overview

This project contains the implementation of an equivalence testing approach for DOTAs learning, called mutation-based equivalence testing. The basic idea/framework comes from Learning from Faults: Mutation Testing in Active Automata Learning. In order to apply the basic framework to the Deterministic One-Clock Timed Automata setting, we further propose several improvements, including two special mutation operators, a heuristic test case generation algorithm, and a score-based test case selection method. By setting reasonable parameters, the newly developed mutation-based equivalence testing is a viable technique for implementing equivalence oracle to find counterexamples in a DOTAs learning setup.

Installation

The project was developed using Python3, and you only need to download the project, but there are a few prerequisites before running:

- Python3.7.* (or high)
- graphviz (used for drawing)

Usage

1. Run experiment

1) Run case study

To run the cases in experiments by Mutation_new:

```
$python3 mutation_new.py
```

To run the cases in experiments by Mutant_checking:

```
$python3 mutant_checking.py
```

To run the cases in experiments by Heuristic_random_testing:

```
$python3 heuristic_random_testing.py
```

2) Run experiments of improvements

To run the cases in experiments by pure random generation method:

```
$python3 generation_random.py
```

To run the cases in experiments by A&T's generation method:

```
$python3 generation_A&T.py
```

To run the cases in experiments only with timed mutation operator:

```
$python3 operator_timed.py
```

To run the cases in experiments only with split operator:

```
$python3 operator_split.py
```

To run the cases in experiments by greedy selection method:

```
$python3 selection_greedy.py
```

3) Run your own model:

> `model.json` is a JSON file about the structure of the model. Although this is a black box learning tool, in the prototype stage, users should provide model structure files to model DOTAs to be learned.

model.json

```
{
    "states": ["1", "2"],
    "inputs": ["a", "b", "c"],
    "trans": {
        "0": ["1", "a", "[3,9)", "r", "2"],
        "1": ["1", "b", "[1,5]", "r", "2"],
        "2": ["1", "c", "[0,3)", "n", "1"],
        "3": ["2", "a", "(5,+)", "n", "1"],
        "4": ["2", "b", "(7,8]", "n", "1"],
        "5": ["2", "c", "(4,+)", "r", "1"]
    },
    "initState": "1",
    "acceptStates": ["2"]
}
```

Explanation:

- "states": the set of the name of locations;
- "inputs": the input alphabet;
- "trans": the set of transitions in the form `id : [name of the source location, input action, guards, reset, name of the target location];`
  - "+" in a guard means INFTY;
  - "r" means resetting the clock, "n" otherwise
- "initState": the name of initial location;
- "acceptStates": the set of the name of accepting locations.

In the process of use, you must ensure that the naming is correct and the content follows the prescribed format.

2. Parameter Settings

For mutation-based equivalence testing, we have set the generally applicable parameters in advance, but users can also customize the relevant parameters in the files `./test/random_testing.py` and `./test/mutation_testing.py`.

Output

If we learn the target DOTA successfully, the final COTA will be drawn and displayed as a PDF file. And all results will be stored in a folder named `results` and a file named `result.json`.