Assignment 1 - Invariant verification - Submission deadline: December 13

Due: Tuesday, 13 December 2022, 11:59 PM

In this assignment you will implement the symbolic algorithm for invariant verification, using BDDs as data structure to represent and manipulate regions.

The attached **inv_mc.smv** file contains a python script that uses the **pynusmv** library to verify invariants of SMV programs.

Using the **inv_mc.smv** script as a starting point, implement a function **check_explain_inv_spec (spec)** that respects the following specifications:

- the function checks if **spec** is an invariant of the loaded SMV model or not, that is, whether all the reachable states of the model satisfy **spec** or not.
- the function must return an explanation for why the model does not satisfy **spec**, if it is the case;
- the return value is a tuple where the first element is **True** and the second element is **None** if the invariant is true. When the invariant is not verified, the first element is **False** and the second element is an execution of the SMV model that violates **spec**;
- the execution is a tuple of alternating states and inputs, starting and ending with a state. States and inputs are represented by dictionaries where keys are state and inputs variable of the loaded SMV model, and values are their value.

The pynusmv library

pynusmv is a python wrapper to NuSMV model checking algorithms.The library consists of several modules. To implement the project you can use only the following modules:

- **init**
- **glob**
- **fsm**, except for method **reachable_states**
- **prop**
- **dd**
- the helper function **spec_to_bdd (model, spec)** included in **inv_mc.py** to convert a property to an equivalent BDD.

You can find more information about the pynusmv library at the websites http://lvl.info.ucl.ac.be/Tools/PyNuSMV and https://pynusmv.readthedocs.io.

Binary files to install pynusmv on recent Python versions are available at https://github.com/davidebreso/pynusmv/releases/latest

Examples

The archive **examples.zip** contains some SMV programs to test your implementation.

How to submit the assigment

- You can do the assigment either on your own or in a group of up to three people.
- Create the group before submitting the assigment using the "Assigments: Groups self-selection" activity on moodle.
- You have to create a group even if you do the assigment on your own.

Your submission should include:

- The complete code of the implementation
- A short report describing the details and implementation choices. The report should describe how your code generate the counterexample, and should justify the correctness of the algorithm.

You can either submit code and report as separate files, or submit a python notebook with code and report.

examples.zip	20 October 2020, 1:17 PM
inv_mc.py	20 October 2020, 1:17 PM

Submission status

Group	3
Submission status	Nothing has been submitted for this assignment
Grading status	Not marked
Time remaining	20 days 11 hours
Grading criteria	**Basic correctness** Is the True/False answer correct for all cases? **Maximum score** 6 **Symbolic implementation** Is the basic reachability algorithm implemented with a symbolic approach? **Maximum score** 4 **Correctness of the counterexamples** Are the counterexample real executions of the system? Are they returned in the correct form? **Maximum score** 6 **Symbolic counterexample search** Is the search for counterexample implemented with a symbolic approach? **Maximum score** 4 **Report: justification of correctness** Does the report prove that the algorithm and the counterexample search are correct? **Maximum score** 4 **Report: other aspects** Does the report describe the solution clearly and with enough details? **Maximum score** 4 **Implementation** Is the code clear and understandable? **Maximum score** 4
Last modified	-
Submission comments	Comments (0)

Add submission

You have not made a submission yet.