Automated Test-case Generator for CPS Model-checkers

A Major Project Report

submitted in partial fulfilment of the requirements for the award of degree

of

Bachelor of Technology

in

Computer Science & Engineering

by

Evanstar Field War ($B15CS037$)
Abhishek Das ($B15CS002$)

Department of Computer Science and Engineering
National Institute of Technology Meghalaya, Shillong
Meghalaya, India

CERTIFICATE

We hereby certify that the work which is being presented in the B.Tech. Major Project Report entitled **Automated Test-case Generator for CPS Model-checkers**, in partial fulfillment of the requirements for the award of the **Bachelor of Technology in Computer Science & Engineering** and submitted to the Department of Computer Science & Engineering of National Institute of Technology Meghalaya, India is an authentic record of our own work carried out during a period from July 2018 to May 2019 under the supervision of **Dr.Rajarshi Ray Assistant Professor.**The matter presented in this report has not been submitted by us for the award of any other degree elsewhere.

Evanstar Field War
B15CS037

Abhishek Das
B15CS002

This is to certify that the above statement made by the candidate(s) is correct to the best of my knowledge.

Date :

Dr. Rajarshi Ray
Assistant Professor

Dr. Diptendu Sinha Roy
Head of Department
Department of Computer Science & Engineering,
National Institute of Technology Meghalaya, India

DECLARATION OF ORIGINALITY

We hereby declare that this project work titled **Automated Test-case Generator for CPS Model-checkers** represents our original work carried out as students of the Department of Computer Science & Engineering of National Institute of Technology Meghalaya, India and to the best of our knowledge it contains no material previously published or written by another person unless cited. Any contribution made to this project work by others, with whom we have worked at National Institute of Technology Meghalaya or elsewhere, is explicitly acknowledged.

Evanstar Field War
B15CS037

Abhishek Das
B15CS002

Place: Shillong
Date:

ACKNOWLEDGEMENTS

We would like to express our deep gratitude and sincere thanks to our project guide Dr. Rajarshi Ray, of Computer Science and Engineering Department, National Institute of Technology Meghalaya for his guidance and active support during the progress of our project. Without his support and encouragement this project would have been trivial.

We also extend our heartiest thanks to our project coordinator Shyambabu Pandey, Master of Technology of Computer Science and Engineering Department for his cooperation and guidance.

We would also like to mention that it would not have been possible without the timely help and support of Computer Science and Engineering Department Laboratories.

On a more personal note, we would like to express our heartiest thanks to our parents, all our seniors and friends, who directly or indirectly, contributed to the successful completion of our project.

ABSTRACT

The term Cyber-Physical System (CPS) refers to a new generation of systems with integrated computational and physical capabilities that can interact with the physical world via sensors and actuators in a feedback loop.The ability to interact with, and expand the capabilities of the physical world through computation, communication, and control is a key enabler for future technology developments. Nowadays, CPS are everywhere, from medical devices to smart buildings.CPS are expected to play a major role in the design and development of future engineering systems with new capabilities that far exceed today's levels of autonomy, functionality, usability, reliability, and cyber security. SpaceEx is a platform for developing reachability and safety verification algorithms for hybrid automata models of CPS. Many model checking algorithms have already been implemented in SpaceEx and it accepts model in a XML file format.

A CPS model checker is software which can analyse mathematical models of CPS for property verification. Interesting properties can be in regards to safety, stability, etc, of CPS. Model-checkers are themselves software which needs to be sufficiently tested. In this project, the aim is to build an automated test-case generator for CPS model-checkers. The generator should attempt to create test-cases which can make existing model-checkers to fail.

Contents

1 Introduction **1**

1.1 Hybrid automata 1

1.2 Need for CPS research 1

1.3 Challenges and opportunities: Industry-Academia . . . 3

1.4 Syntax 4

2 Example of hybrid automaton - water tank system **5**

3 Literature survey **6**

3.1 SpaceEx model-checker 6

3.2 HYST 8

4 Why we need automated test-gase generator? **9**

5 Automated test-case generator for CPS model-checkers **9**

6 Implementation of automated test-case generator **10**

6.1 Number of locations of hybrid automata 10

6.2 Transitions of hybrid automata 11

6.3 Type of flow, invariant, guard and assignment 13

6.4 Graphical user interface (GUI) for CPS model-checkers 13

6.5 K-number of random test-cases 14

6.6 The reachable states of a hybrid automata region computed by SpaceEx 15

6.7 C++ script for automated running of the generated models 17

7 Result **17**

7.1 Running status of test-case CPS models on SpaceEx . 17

7.2 Translation status of test-case CPS model from SpaceEx to Xspeed 22

8 Conclusion **27**

List of Figures

1 Hybrid automaton of water tank system 5

2 SpaceEx model-checker [1] 7

3 Model transformation passes [2] 8

4 Automated test-case generator. 10

5 Locations of the generated hybrid automata. 11

6 Transitions generated by automated test-case generator 12

7 Concrete hybrid automata generated by automated test-case generator 12

8 Graphical User Interface of the automated test-case generator. 14

9 Test cases generated in the project directory 15

10 A sample hybrid automata whose reachable states are computed 16
11 Reachable states of Hybrid Automaton region. 16
12 Input and output of C++ script. 17

List of Tables

1 Type of guard, flow, assignment, and invariant 13
2 Status of tested models in SpaceEx 18
3 Showing translation status of the generated models. . . 22

1 Introduction

In this section, we intoduce hybrid automata and syntax of hybrid automata.

1.1 Hybrid automata

A CPS is a collection of computing devices that communicate with each other and interacts with the physical world via sensor and actuators in a feedback loop. The mathematical formalism of the CPS model is called hybrid automata. A Hybrid automaton is a formal model that consists of both discrete locations that are known as finite state and continuous regions [3]. A hybrid automaton shows two kinds of state changes discrete jump transitions and when time elapses, continuous flow transition occurs. Hybrid systems are a safety critical system. Hybrid automata define the behavior of hybrid system.

1.2 Need for CPS research

CPS research is still in its infancy. Professional and institutional barrier have resulted in narrowly defined, discipline-specific research and education venues in academia for the science and engineering disciplines. Research is partitioned into isolated subdisciplines such as sensors, communications and networking, control theory, mathematics, software engineering, and computer science. For example, systems are designed and analyzed using a variety of modeling formalisms and tools. Each representation highlights certain features and disregards others to make analysis tractable. Typically, a particular formalism represents either the cyber or the physical process well, but not both. Whereas differential equations are used for modeling physical processes, frameworks such as automata are used to represent discrete behavior and control flows. Workforce expertise is similarly partitioned, to the detriment of productivity, safety, and efficiency. Although this approach to modeling and formalisms may suffice to support a component-based "divide and conquer" approach to CPS devel-

opment, it poses a serious problem for verifying the overall correctness and safety of designs at the system level and component-to-component physical and behavioral interactions[4]. In the following paragraphs, research needs in CPS are briefly discussed.

Abstraction and architectures : Innovative approaches to abstraction and architectures that enable seamless integration of control, communication, and computation must be developed for rapid design and deployment of CPS. For example, in communication networks, interfaces have been standardized between different layers. Once these interfaces have been established, the modularity allows specialized developments in each layer. The overall design allows heterogeneous systems to be composed in plug-and-play fashion, opening opportunities for innovation and massive proliferation of technology and the development of the Internet. However, the existing science and engineering base does not support routine, efficient, robust, modular design and development of CPS. Standardized abstractions and architectures are urgently needed to fully support integration and interoperability and spur similar innovations in CPS[5].

Distributed computations and networked control : The design and implementation of networked control systems pose several challenges related to time and event-driven computing, software, variable time delays, failures, reconfiguration, and distributed decision support systems. Protocol design for real-time quality-of-service guarantees over wireless networks, trade-offs between control law design and real-time-implementation complexity, bridging the gap between continuous and discrete-time systems, and robustness of large systems are some of the challenges for CPS research. Frameworks, algorithms, methods, and tools are needed to satisfy the high reliability and security requirements for heterogeneous cooperating components that interact through a complex, coupled physical environment operating over many spatial and temporal scales.

1.3 Challenges and opportunities: Industry-Academia

Biomedical and Healthcare Systems : CPS research is revealing numerous opportunities and challenges in medicine and biomedical engineering. These include intelligent operating rooms and hospitals, image-guided surgery and therapy, fluid flow control for medicine and biological assays, and the development of physical and neural prostheses. Healthcare increasingly relies on medical devices and systems that are networked and need to match the needs of patients with special circumstances.Thus, medical devices and systems will be needed that are dynamically reconfigured, distributed, and can interact with patients and caregivers in complex environments. For example, devices such as infusion pumps for sedation, ventilators and oxygen delivery systems for respiration support, and a variety of sensors for monitoring patient condition are used in many operating rooms. Often, these devices must be assembled into a new system configuration to match specific patient or procedural needs. The challenge is to develop systems and control methodologies for designing and operating these systems that are certifiable, safe, secure, and reliable.

Next-generation air transportation systems : CPS research is likely to have an impact on the design of future aircraft and air traffic management systems, as well as on aviation safety. Specific research areas include (1) new functionality to achieve higher capacity, greater safety, and more efficiency, as well as the interplay and trade-offs among these performance goals; (2) integrated flight deck systems, moving from displays and concepts for pilots to future (semi)autonomous systems; (3) vehicle health monitoring and vehicle health management; and (4) safety research relative to aircraft control systems. One of the key technical challenges to realizing NextGen involves verification and validation of complex flight-critical systems with a focus on promoting reliable, secure, and safe use for NextGen operations. As the complexity of systems increases, costs related to verification and validation and safety assurance will likely increase the cost of designing and building next-generation vehicles. The broader aeronautics community has identified verification and validation methodologies and concepts as a critical research area.

1.4 Syntax

A hybrid automaton is a generalized finite-state automaton that is equipped with continuous variables. The discrete changes of the hybrid system are modelled by the edges of the hybrid automata, and the continous evolutions of the hybrid automata are modelled by differential equations that label locations of the hybrid automata. The syntax of hybrid automaton is described as follows.

- A hybrid automaton (HA) has eight tuples (Locations, Edges, Σ, X, Initial condition, Invariants, Flows, Jumps) where:
- Location is a finite set of locations $(loc1, loc2, .., locn)$ that denote the finite states of hybrid system.
- Σ denotes finite set of event names.
- Edges $\subseteq$ Locations$\star\Sigma\star$Locations denotes finite set of labeled edges which represent discrete evolution of control mode in the hybrid system.
- X$(x_1, x_2, ..., x_n)$ is finite set of real-valued variables. Dotted variables $\dot{X}$ $(\dot{x_1}, \dot{x_2}, ..., \dot{x_m})$ are used to represent first derivation of variables during continuous evolutions.
- Initial condition, Invariants, Flows are vertex labeling functions which assigns to each locations. Initial condition(loc) is a predicate whose free variables are from X and which states the possible valuations for those variables when the hybrid system starts from location loc. Variables of Invariants(loc) are from X and hybrid system will stays in a location till invariants holds. Variables of Flows(loc) are from X$\cup\dot{X}$ and which states the possible continuous evolutions when the control of the hybrid system is in location loc.
- Jumps is a function that assigns to each a labeled edge a predicate whose free variables are from X$\cup\dot{X}$. It states when the discrete change is possible and what possible updates of the variables when the hybrid system makes discrete changes.[3]

2 Example of hybrid automaton - water tank system

In this section, we give an example of Hybrid Automaton. The tank is

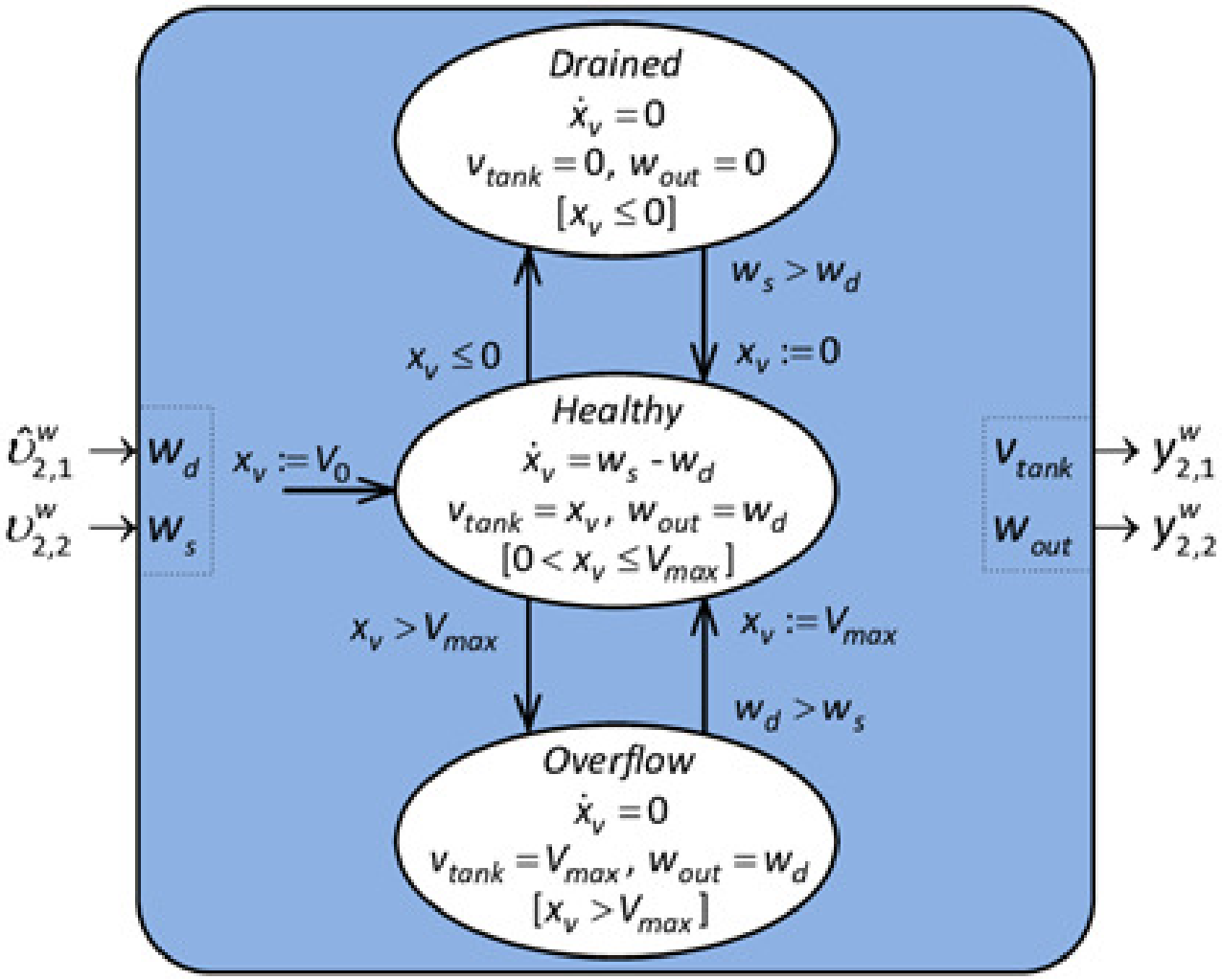


Figure 1: Hybrid automaton of water tank system

modeled with the open hybrid automaton shown in Figure 1. The continuous state x_v denotes the volume of the tank and changes according

to the following equation,

$$\dot{x} = w_s - w_d$$

where w_s denotes the input water supply rate from the pumping station, and w_d the output water demand rate. The volume x_v also determines the discrete state of the model. As shown in figure 1 the model will remain at the "Healthy" state while

$$0 < x_v < V_{max}$$

where V_{max} is the tank's maximum volume and it will transition to either "Drained" state if $x_v <= 0$ or the "Overflow" state if $x_v > V_max$.The model will return back to the "Healthy" state if, while in "Drained" state, the water supply becomes larger than the demand $w_s > w_d$, or if, while in the "Overflow" state, the water demand becomes larger than the water supply $w_d > w_s$. Finally the two outputs of the model, v_{tank} (that denotes the tank volume measurement), and w_{out} (that denotes the tank output water supply rate), take the proper values in each discrete state as shown in figure 1.

3 Literature survey

We have read some paper which are related to our project. First one is about SpaceEx model checker which is a faithful model checker. Second is about Hyst which is a translation tool that translate SpaceEx's format to other tool format.

3.1 SpaceEx model-checker

SpaceEx is a symbolic model checker that supports affine hybrid automata. It operates on symbolic states that contains a discrete location on the continuous region. A hybrid automata that works on affine function for flow, invariant and reset is called affine hybrid automata [1]. Affine function means a linear function + a constant. For example, a 1-dimension equation of affine function is : $y = ax + b$.

Reachability algorithm for SpaceEx is : $S_0 := post(init)$; $S_{(i+1)} := S_i \cup post(post(S_i))$;where init=initial state, post(s): continuous post operation and post(s):dynamic post operation.

Reachability is an important verification problem that demands to decide whether a set of points is reachable from a starting region.

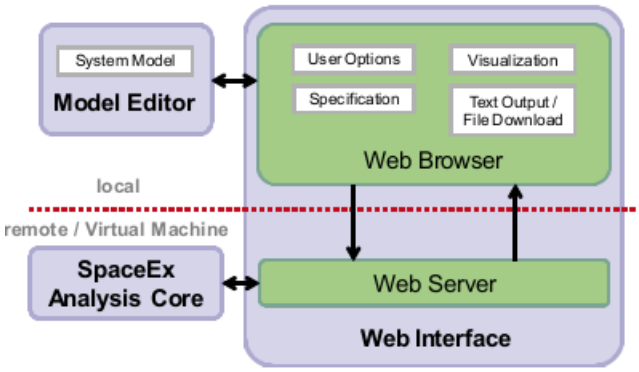


Figure 2: SpaceEx model-checker [1]

Explanation of Figure

- The model editor is a editor which is used as graphical editor for creating models of hybrid automata out of nested components. It produces model files in SpaceEx's sx format.
- Second component is command line program that is also known as the analysis core that takes a model file in sx format, a configuration(cfg) file that defines the initial states, other options the and scenario, analyzes the system and produces a series of output files.
- The third component of SpaceEx's is web interface, which is a graphical user interface with which one can comfortably specify other analysis parameters and initial states, visualize the output graphically, and run the analysis core. The web interface is browser based, and accesses the analysis core via a web server, which may be running locally or remotely on a virtual machine.

3.2 HYST

Direct translation from XML format used in SpaceEx to C++ format used in Xspeed is very difficult. To do this conversion we employ a Hyst tool which makes this conversion possible. Hyst works as an intermediate tool which takes input from SpaceEx in XML format and it gives output in corresponding tools format. The tool is capable of quickly converting a model to a number of hybrid system model checking tools. The tool supports generic model-to-model transformation passes that serve to both ease the translation and potentially improve reachability results for the supported tools. This tool can automatically translate benchmarks of several classes to the other tools input format. It gives a single platform for generating modified input, modification of models sources for the unmodified tools. It implements Java data structures for encoding the continuous variables, transitions, modes, invariants, differential equations of flow, and guards.

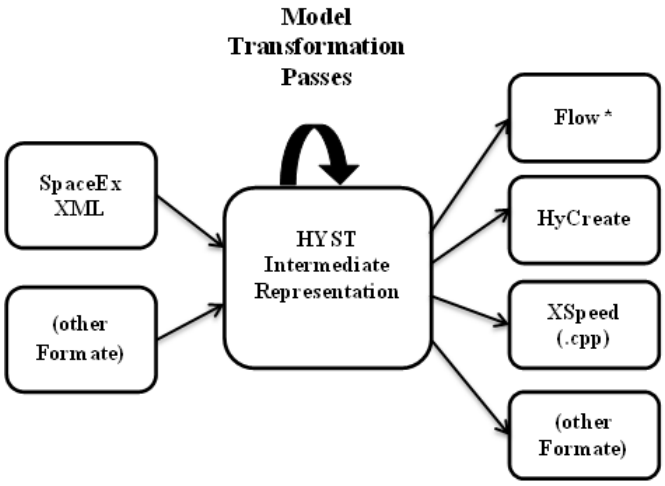


Figure 3: Model transformation passes [2]

Hyst is being used in three main ways. First, a user of verification tools can quickly generate a model file for a number of tools in order to find a tool that best fits the system under consideration. Second,

a developer of hybrid systems model checkers can use HYST to both compare the performance of newly developed algorithms with other up to date analysis tools, as well as to quickly check for correctness against a common set of models and as part of a regression test suite. Third, researches can write generic model transformation passes that modify the tools hybrid automaton intermediate representation.

4 Why we need automated test-gase generator?

- Manually generating many random models can be a very tedious process.
- There are not many random models from existing literature.
- The generated xml models can be test-cases for model checkers to check if they parse models that satisfy the given properties correctly.
- Model checkers should be able to run correctly and should not crash for different types of models which may have missing invariants, missing guards and also multiple edges between locations

5 Automated test-case generator for CPS model-checkers

Automated test-case generator will take tuples of Hybrid Automata (like number of locations, number of dimensions, number of transitions, type of flow, invariant, guard, assignment) as input and it will generate a concrete hybrid automaton model. The model file is in SPACEEX′S SX format, an XML based format for which there is a graphical model editor. Sx models are similar to the hybrid automaton known in literature, except that they provide a richer mechanism of hierarchy, templates and instantiations. When SpaceEx reads an sx file, it translates the components into either an automaton or into a network of automata in parallel composition.

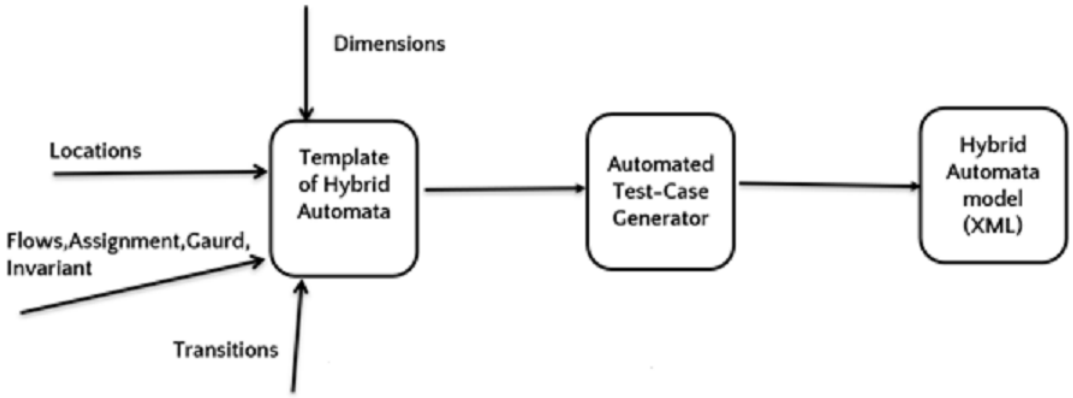


Figure 4: Automated test-case generator.

6 Implementation of automated test-case generator

The results from our implementation of the automated test-case generator are as follow :

6.1 Number of locations of hybrid automata

First it accepts the number of location of Hybrid Automaton from user and it generates locations. For example if a user gives number of location = n then it generates locations name as $v_0, v_1, ..., v_{n-1}$.The generated locations are illustrated in the following figure.

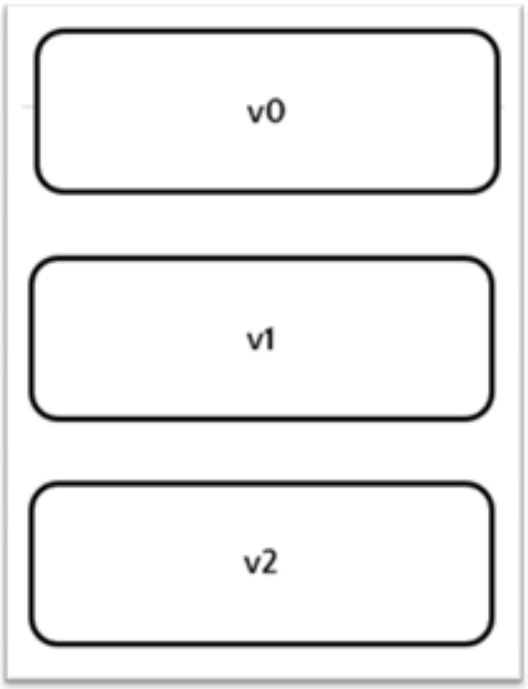


Figure 5: Locations of the generated hybrid automata.

6.2 Transitions of hybrid automata

Ît also accepts number of transitions from the user and randomly generates transitions between two locations. First it generates random locations and then put the transitions between the locations. For example if a user enter the number of transitions = n then it generates randomly $e_1, e_2, ..., e_{n-1}$. The generated transitions are illustrated in the following figure.

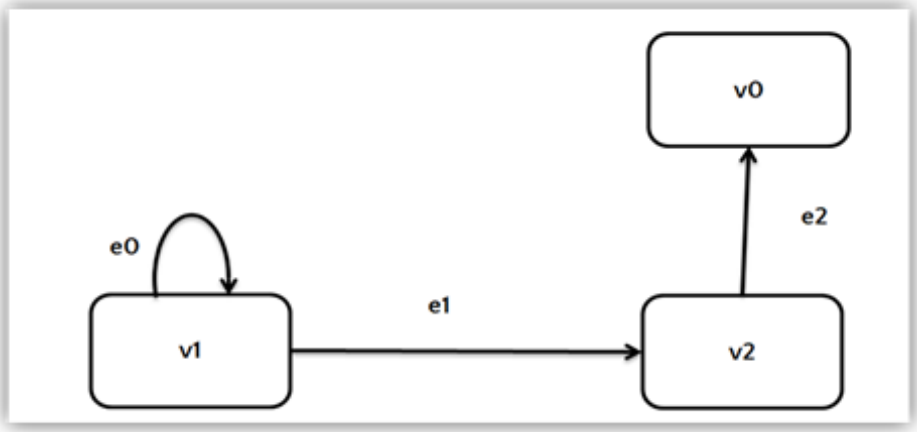


Figure 6: Transitions generated by automated test-case generator

After the steps of generating the locations and the transitions, we get to the final step of generating a concrete or complete hybrid automaton which has the type of flows, invariants, guards and assignment as specified in the template given in the test-case generator.

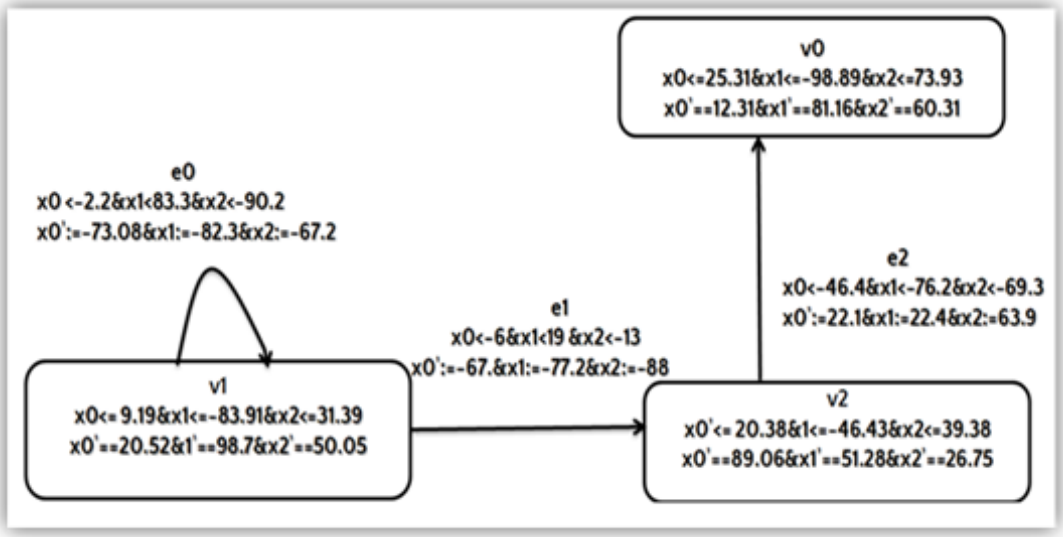


Figure 7: Concrete hybrid automata generated by automated test-case generator

6.3 Type of flow, invariant, guard and assignment

In the previous version, the Automated Model Generator supports only constant type of flow, invariant, guard, and assignment and now it extends to support to dierent types of dynamics for flow, invariant, guard, assignment. The automated test case generator supports the following types of dynamics for flow, invariant, and assignment:

1. Constant dynamics,
2. Affine dynamics, and
3. None (flow is zero)

In the figure, x0, x1 and x2 are dimensions of Hybrid Automata:

Table 1: Type of guard, flow, assignment, and invariant

Flow Type	**Guard Type**	**Invariant Type**	**Assignment Type**
constant ($\dot{y} = c$)	constant bound($y \leq c$)	constant bound ($y \leq c$)	constant bound ($y := c$)
affine ($\dot{y} = ay + b$)	polytope ($ax \leq c$)	polytope ($ax \leq c$)	linear ($x := ax + b$)

6.4 Graphical user interface (GUI) for CPS model-checkers

This interface uses icons, menus and other visual indicator (graphics) representations to display information and related user controls, unlike text-based interfaces, where data and commands are in text.
Eclipse Window Builder plug-in used:

WindowBuilder is built as a plug-in to Eclipse and the various Eclipse-based IDEs (RAD, RSA, MyEclipse, JBuilder, etc.).

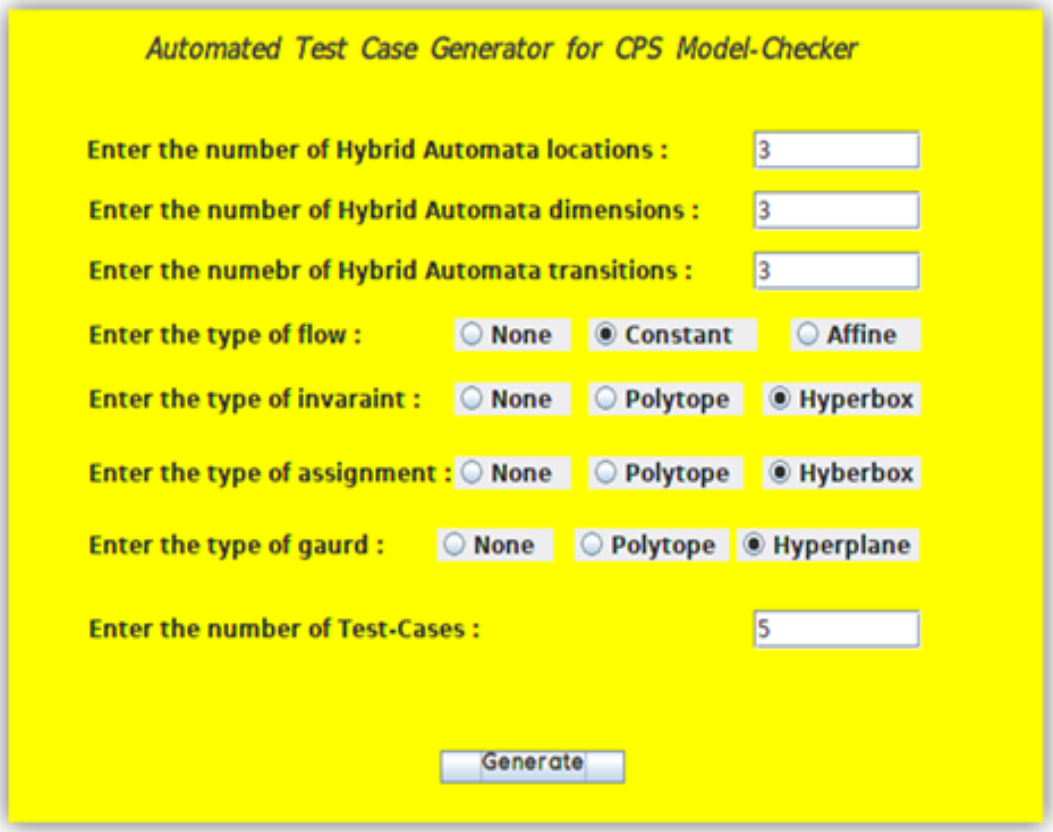


Figure 8: Graphical User Interface of the automated test-case generator.

6.5 K-number of random test-cases

The automated test-case generator can now generate k number of SpaceEx models of a given template for model checkers to test them. In the example given below we have entered the number of test cases as 4 and the following files are generated. For each XML file that is generated there is also a corresponding CFG file associated with it. A CFG file is a configuration file format which is used for storing settings of each XML model.

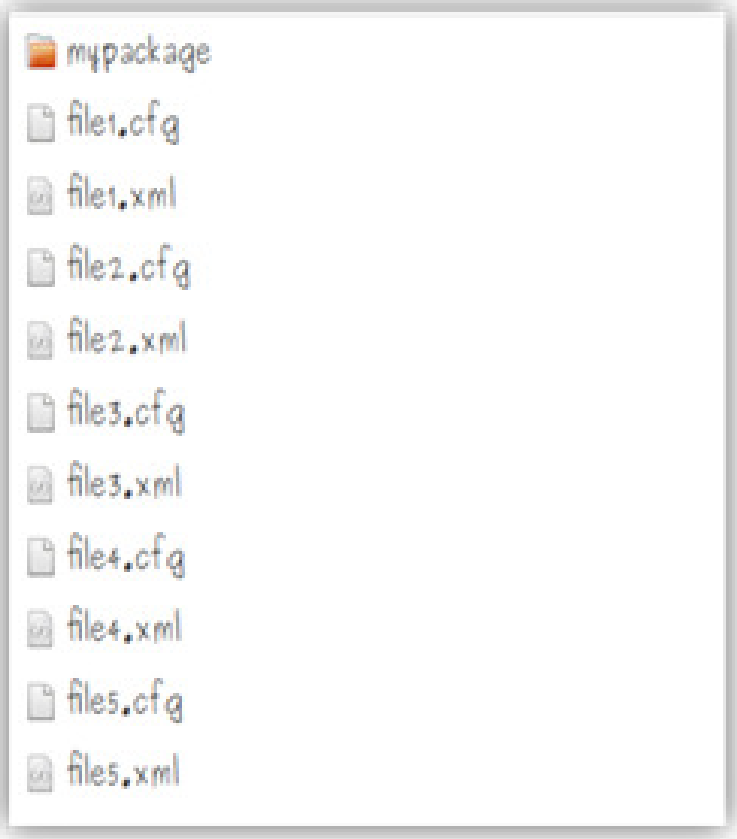


Figure 9: Test cases generated in the project directory

6.6 The reachable states of a hybrid automata region computed by SpaceEx

SpaceEx compute the reachable states based on two operators:
1. Computing the states reachable by time elapse ,and
2. Computing the image of a set of states that take a transition.[6]

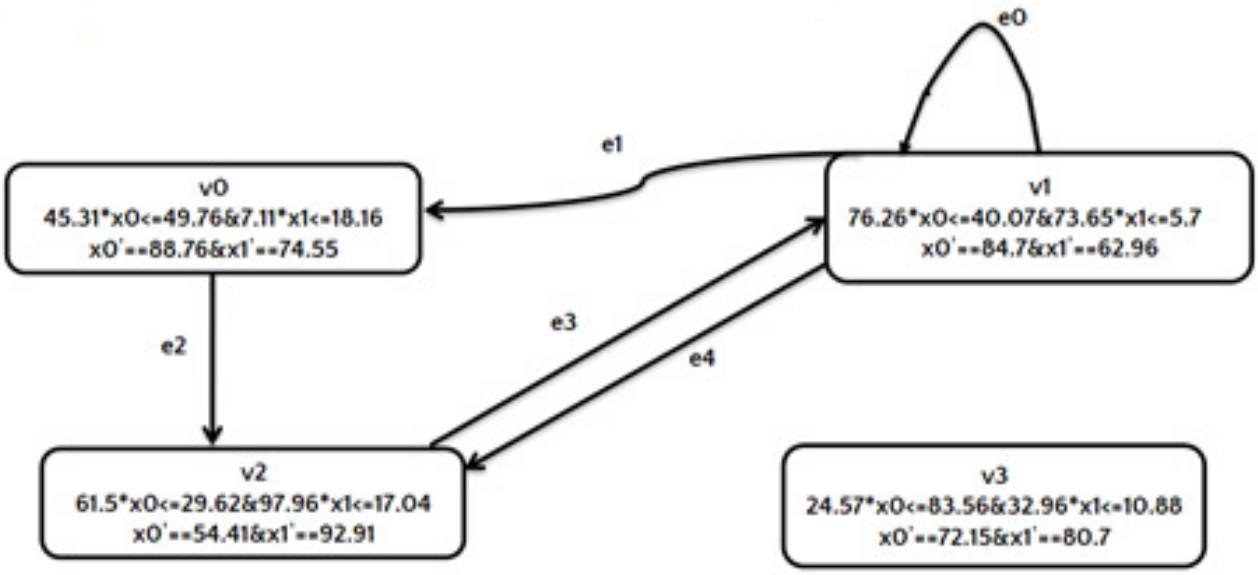


Figure 10: A sample hybrid automata whose reachable states are computed

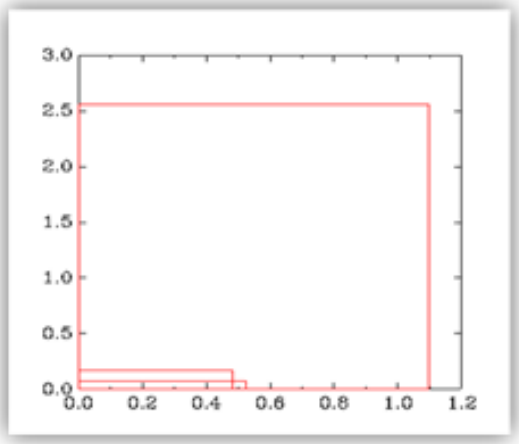
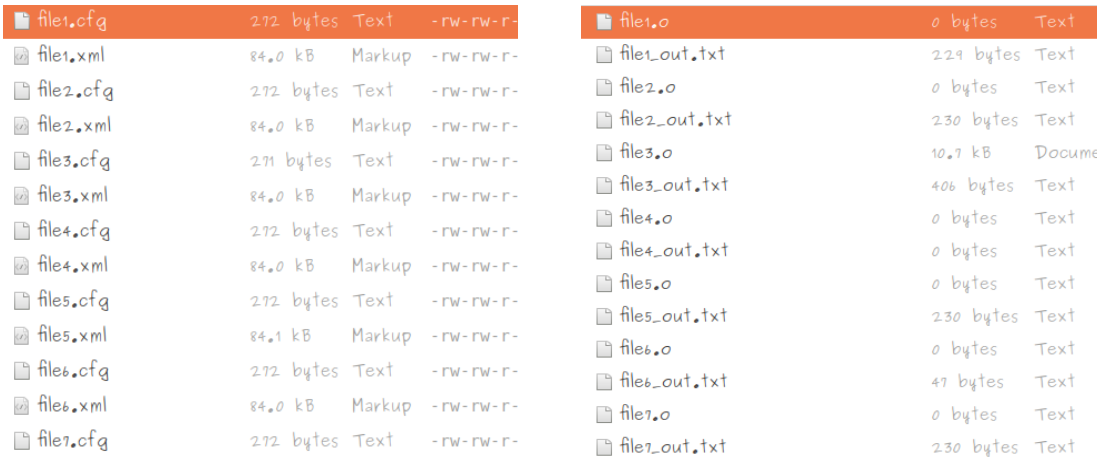


Figure 11: Reachable states of Hybrid Automaton region.

6.7 C++ script for automated running of the generated models

The checking of the generated test cases (xml and cfg files) in SpaceEx has been automated by a c++ script taking the n number of test cases at a time. For each test-case (xml and cfg file), the output will be generated in the specified output folder in the form of a .txt and .o files.



(a) Test-cases (b) Output files

Figure 12: Input and output of C++ script.

7 Result

7.1 Running status of test-case CPS models on SpaceEx

The SpaceEx tool platform is designed to facilitate the implementation of algorithms related to reachability and safety verification. The SpaceEx server can generate GIF and PDF images of two-dimensional graphs.

Some of the generated models are being run on SpaceEx and the running status messages are noted down in the table below.

Table 2: Status of tested models in SpaceEx

Test-cases(xml,cfg)	**Running status**	**Message**
test1	No	infinite loop at iteration 1
test2	No	infinite loop at iteration 1
test3	Yes	reachable states computed after 0.0150001s
test4	No	infinite loop at iteration 0
test5	No	infinite loop at iteration 1
test6	Yes	reachable states computed after 0.0150001s
test7	Yes	reachable states computed after 0.0150001s
test8	No	infinite loop at iteration 0
test9	No	infinite loop at iteration 0
test10	Yes	reachable states computed after 0.0150001s
test11	Yes	reachable states computed after 0.0150001s
test12	Yes	reachable states computed after 0.0150001s
test13	Yes	reachable states computed after 0.0140001s
test14	No	infinite loop at iteration 0
test15	No	no output content
test16	No	infinite loop at iteration 1
test17	No	infinite loop at iteration 1
test18	Yes	infinite loop at iteration 1
test19	Yes	reachable states computed after 0.0130001s
test20	Yes	reachable states computed after 0.0130001s
test21	Yes	reachable states computed after 0.0140001s
test22	Yes	reachable states computed after 0.0130001s
test23	No	infinite loop at iteration 1

Continuation of Table 2		
Test-cases(xml,cfg)	**Running status**	**Message**
test24	Yes	reachable states computed after 0.0130001s
test25	Yes	reachable states computed after 0.0130001s
test26	Yes	reachable states computed after 0.0130001s
test27	No	infinite loop at iteration 1
test28	No	infinite loop at iteration 1
test29	No	infinite loop at iteration 1
test30	No	infinite loop at iteration 1
test31	No	infinite loop at iteration 1
test32	No	infinite loop at iteration 1
test33	No	infinite loop at iteration 1
test34	Yes	reachable states computed after 0.0130001s
test35	No	infinite loop at iteration 1
test36	Yes	reachable states computed after 0.0140001s
test37	No	infinite loop at iteration 1
test38	Yes	reachable states computed after 0.0130001s
test39	No	infinite loop at iteration 1
test40	Yes	reachable states done after 0.0140001s
test41	No	infinite loop at iteration 1
test42	Yes	reachable states computed after 0.0140001s
test43	No	infinite loop at iteration 1
test44	Yes	reachable states computed after 0.0130001s
test45	No	infinite loop at iteration 1
test46	Yes	reachable states computed after 0s
test47	Yes	reachable states computed after 0s
test48	Yes	reachable states computed after 0s
test49	Yes	reachable states computed after 0s

Continuation of Table 2		
Test-cases(xml,cfg)	**Running status**	**Message**
test50	Yes	reachable states computed after 0s
test51	Yes	reachable states computed after 0.00100001s
test52	Yes	reachable states computed after 0s
test53	Yes	reachable states computed after 0s
test54	Yes	reachable states computed after 0s
test55	Yes	reachable states computed after 0s
test56	Yes	reachable states computed after 0s
test57	Yes	reachable states computed after 0s
test58	Yes	reachable states computed after 0s
test59	Yes	reachable states computed after 0s
test60	Yes	reachable states computed after 0s
test61	Yes	reachable states computed after 0s
test62	Yes	reachable states computed after 0s
test63	Yes	reachable states computed after 0s
test64	Yes	reachable states computed after 0s
test65	No	no output content
test66	No	infinite loop at iteration 1
test67	No	infinite loop at iteration 1
test68	Yes	reachable states computed after 0.00100001s
test69	Yes	reachable states computed after 0s
test70	Yes	reachable states computed after 0.00100001s
test71	Yes	reachable states computed after 0.00100001s
test72	Yes	reachable states computed after 0s
test73	Yes	reachable states computed after 0.00100001s
test74	Yes	reachable states computed after 0s
test75	Yes	reachable states computed after 0.00100001s
test76	Yes	reachable states computed after 0.00100001s

Continuation of Table 2		
Test-cases(xml,cfg)	**Running status**	**Message**
test77	Yes	reachable states computed after 0s
test78	Yes	reachable states computed after 0.00100001s
test79	Yes	reachable states computed after 0.00100001s
test80	Yes	reachable states computed after 0.00200001s
test81	Yes	reachable states computed after 0.00200001s
test82	Yes	reachable states computed after 0s
test83	Yes	reachable states computed after 0.00100001s
test84	Yes	reachable states computed after 0.00100001s
test85	Yes	reachable states computed after 0s
test86	Yes	reachable states computed after 0.00100001s
test87	Yes	reachable states computed after 0.00100001s
test88	Yes	reachable states computed after 0s
test89	Yes	reachable states computed after 0.00200001s
test90	Yes	reachable states computed after 0.00200001s
test91	Yes	reachable states computed after 0s
test92	Yes	reachable states computed after 0.00100001s
test93	Yes	reachable states computed after 0.00100001s
test94	Yes	reachable states computed after 0s
test95	Yes	reachable states computed after 0s
test96	No	infinite loop at 1
test97	Yes	infinite loop at 1
test98	Yes	reachable states computed after 0s

Continuation of Table 2		
Test-cases(xml,cfg)	**Running status**	**Message**
test99	Yes	reachable states computed after 0.00100001s
test100	Yes	reachable states computed after 0.00500001s
End of Table		

7.2 Translation status of test-case CPS model from SpaceEx to Xspeed

Direct translation from XML format used in SpaceEx to C++ format used in Xspeed is very difficult. To do this conversion we employ a Hyst tool which makes this conversion possible. Our evaluation demonstrates HYST is capable of automatically translating benchmarks in several classes (including affine and nonlinear hybrid automata) to the input formats of several tools. Additionally, we illustrate a general model transformation pass based on invariants implemented in HYST that illustrates the reachability improvement. Some of the generated models are being translated on Hyst model translator tool and the translating status and output messages are noted down in the table below.

Table 3: Showing translation status of the generated models.

Test-cases(xml,cfg)	**Running status**	**Message**
test1	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test2	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test3	Successful	Finished converting in 121 ms

Continuation of Table 3		
Test-cases(xml,cfg)	**Running status**	**Message**
test4	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test5	Successful	Finished converting in 121 ms
test6	Successful	Finished converting in 858 ms
test7	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test8	Successful	Finished converting in 360 ms
test9	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test10	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test11	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test12	Successful	Finished converting in 191 ms
test13	Successful	Finished converting in 170 ms
test14	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test15	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test16	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test17	Successful	Finished converting in 127 ms
test18	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test19	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test20	Successful	Finished converting in 163 ms
test21	Unsuccessful	Hyst error while exporting
test22	Unsuccessful	Hyst error while exporting
test23	Successful	Finished converting in 121 ms
test24	Successful	Finished converting in 110 ms
test25	Successful	Finished converting in 119 ms
test26	Unsuccessful	Hyst error while exporting
test27	Successful	Finished converting in 100 ms

Continuation of Table 3		
Test-cases(xml,cfg)	**Running status**	**Message**
test28	Successful	Finished converting in 102 ms
test29	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test30	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test31	Successful	Finished converting in 111 ms
test32	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test33	Unsuccessful	Hyst error while exporting
test34	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test35	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test36	Unsuccessful	Hyst error while exporting
test37	Successful	Finished converting in 117 ms
test38	Successful	Finished converting in 95 ms
test39	Successful	Finished converting in 109 ms
test40	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test41	Successful	Finished converting in 97 ms
test42	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test43	Unsuccessful	Hyst error while exporting
test44	Successful	Finished converting in 93 ms
test45	Unsuccessful	Hyst error while exporting
test46	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test47	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test48	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test49	Successful	Finished converting in 100 ms
test50	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass

Continuation of Table 3		
Test-cases(xml,cfg)	**Running status**	**Message**
test51	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test52	Unsuccessful	Hyst error while exporting
test53	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test54	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test55	Successful	Finished converting in 856 ms
test56	Successful	Finished converting in 858 ms
test57	Successful	Finished converting in 350 ms
test58	Successful	Finished converting in 278 ms
test59	Successful	Finished converting in 350 ms
test60	Successful	Finished converting in 350 ms
test61	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test62	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test63	Unsuccessful	Hyst error while exporting
test64	Unsuccessful	Hyst error while exporting
test65	Unsuccessful	Hyst error while exporting
test66	Successful	Finished converting in 410 ms
test67	Successful	Finished converting in 363 ms
test68	Unsuccessful	Hyst error while exporting
test69	Successful	Finished converting in 193 ms
test70	Successful	Finished converting in 181 ms
test71	Successful	Finished converting in 173 ms
test72	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test73	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test74	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test75	Successful	Finished converting in 173 ms

Continuation of Table 3		
Test-cases(xml,cfg)	**Running status**	**Message**
test76	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test77	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test78	Successful	Finished converting in 168 ms
test79	Successful	Finished converting in 560 ms
test80	Successful	Finished converting in 453 ms
test81	Unsuccessful	Hyst error while exporting
test82	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test83	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test84	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test85	Successful	Finished converting in 126 ms
test86	Successful	Finished converting in 265 ms
test87	Successful	Finished converting in 566 ms
test88	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test89	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test90	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test91	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test92	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test93	Unsuccessful	Hyst error while exporting
test94	Unsuccessful	Hyst error while exporting
test95	Unsuccessful	Hybrid Automaton IR structure was corrupted after running pass
test96	Successful	Finished converting in 233 ms
test97	Successful	Finished converting in 569 ms
test98	Successful	Finished converting in 745 ms

Continuation of Table 3		
Test-cases(xml,cfg)	**Running status**	**Message**
test99	Successful	Finished converting in 522 ms
test100	Successful	Finished converting in 423 ms
End of Table		

8 Conclusion

Through the automated test-case we have generated a number of SpaceEx xml models which have different combinations of types of flow, guard, invariant and assignment and run them on SpaceEx to compute the reachable states. Most of the models when tested run into infinite loop in the first iteration 1 or 0 whereas some of the models are successfully run on SpaceEx with warning messages that we have noted down. We also got the graph output of the SpaceEx models which had run successfully on SpaceEx as we have shown in the above graph for one of the generated models.

The generated models are also run on Hyst, a model translator to check if they can be translated to some other format for other model checker tools. For example, like XSpeed where the test-case format is in ".cpp" file. Here also we found out some models which are translated successfully and some which are not. We have also implemented a C++ script where the checking of the generated test cases (xml and cfg files) in SpaceEx has been automated by taking the n number of test cases at a time. This has helped a lot in running all the test-cases all at once in SpaceEx to generate the ".o" and ".txt" output files. Through this we observed that SpaceEx may still fail for some correct models and runs into infinite loop, while for the rest of the models it could run successfully with error messages.

References

[1] G. Frehse, "An introduction to spaceex v0. 8," 2010.

[2] S. Bak, S. Bogomolov, and T. T. Johnson, "Hyst: a source transformation and translation tool for hybrid automaton models," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. ACM, 2015, pp. 128–133.

[3] J.-F. Raskin, "An introduction to hybrid automata," *Handbook of networked and embedded control systems*, pp. 491–517, 2005.

[4] A. Rajhans, S.-W. Cheng, B. Schmerl, D. Garlan, B. H. Krogh, C. Agbi, and A. Bhave, "An architectural approach to the design and analysis of cyber-physical systems," *Electronic Communications of the EASST*, vol. 21, 2009.

[5] S. Graham, G. Baliga, and P. Kumar, "Abstractions, architecture, mechanisms, and a middleware for networked control," *IEEE Transactions on Automatic Control*, vol. 54, no. 7, pp. 1490–1503, 2009.

[6] G. Frehse, "An introduction to spaceex v0. 8," *December*, 2010.