HYST: A Source Transformation and Translation Tool for Hybrid Automaton Models

Stanley Bak
Air Force Research Laboratory, Rome, NY, USA

Sergiy Bogomolov
Albert-Ludwigs-Universität Freiburg, Germany

Taylor T. Johnson
University of Texas at Arlington, USA

ABSTRACT

A number of powerful and scalable hybrid systems model checkers have recently emerged. Although all of them honor roughly the same hybrid systems semantics, they have drastically different model description languages. This situation (a) makes it difficult to quickly evaluate a specific hybrid automaton model using the different tools, (b) obstructs comparisons of reachability approaches, and (c) impedes the widespread application of research results that perform model modification and could benefit many of the tools. In this paper, we present HYST, a Hybrid Source Transformer. HYST is a source-to-source translation tool, currently taking input in the SpaceEx model format, and translating to the formats of HyCreate, Flow*, or dReach. Internally, 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. Although these model transformation passes could be implemented within each tool, the HYST approach provides a single place for model modification, generating modified input sources for the unmodified target tools. 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 pseudo-invariants implemented in HYST that illustrates the reachability improvement.

1. INTRODUCTION

Hybrid systems are mathematical models that combine discrete and continuous dynamics. This formalism can capture the behavior of a large range of real-world systems. For example, embedded systems [23, 35] and systems from the biological domain [19, 21] can be modeled using the hybrid systems formalism. At the same time, the resulting system behavior requires a careful handling to ensure a precise yet time efficient analysis. A number of powerful and scalable model checkers have recently emerged [7, 16, 26, 28, 33]. They cover a number of hybrid system classes, e.g., affine vs. non-linear continuous dynamics and monolithic vs. automata networks. Furthermore, the analysis algorithms are built around different ideas and state representations, e.g., flow-pipe construction vs. decision procedures for differential equations. These design decisions make the tools particularly efficient in some settings, e.g., such as only for some classes of continuous dynamics.

In this paper, we present an automatic source-to-source model converter from the SpaceEx input format to the Flow*, HyCreate, and dReach formats. At present, direct comparisons between model checkers cannot be done out-of-the-box as the input languages are syntactically different. However, a manual comparison is possible because, although the input languages of the considered model checkers differ *syntactically*, they rely on the same behavioral *semantics*.

We envision HYST 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 tool's hybrid automaton intermediate representation. Then, all supported tools can immediately benefit from these research advances, rather than having to reimplement new approaches piecemeal for each tool.

Related Work. In the last decade, several research groups have worked on approaches to unify the syntax of hybrid model checkers. Sangiovanni-Vincentelli and his co-authors suggest the hybrid systems interchange format (HSIF) [11, 12, 39, 40]. A further attempt to provide a common input language focused on the model composition has been undertaken within the FP7 Multiform project [2, 44]. The project resulted in the Compositional Interchange Format (CIF). Earlier efforts for interchange formats were initiated for Charon [4]. The above outlined projects have in common an idea to collect all the features available in different hybrid model checkers and provide an input language which essentially subsumes the languages of every particular tool. Although having a common interchange language supported by all the tools would be a ultimate solution, this approach hinges on the willingness of the tool developers to support such an input format. Furthermore, the incorporation of a common format into an established tool by third party developers would be difficult due to the time overhead needed to get acquainted with the code of each particular hybrid model checker.

Alternative approaches include using other highly-used languages as standard input formats. Agrawal et al. [1] suggest an algorithm to translate Simulink and Stateflow models into the equivalent HSIF models. In a slightly different setting, Schrammel et al. [43] consider the translation problem for complex SSMs where involved treatment of zero-crossings is needed. Chen et al. [13] provide a translation from Stateflow to CSP [32]. Alur et al. [5] propose to use the symbolic analysis in order to improving the test coverage of SSMs. In this scope, a considered SSM is converted to a linear hybrid automaton. Assuming that a SSM can be converted to a hybrid system, our tool can principally use the resulting automaton as its input. Mathworks, unfortunately, does not provide any rigid operational semantics for its tools. This makes the model translation process error-prone and ambiguous, whereas we are mostly concerned with the formal verification of a given model. Other recent languages include the HYbrid systems with Discrete Interaction (HyDI) language, which is an extension of the SMV input language [17]. Recent converter initiatives include a converter from Ptolomy II to SpaceEx [42], and the HyLink converter from SSMs to hybrid automata [36].

2. HYBRID MODEL CHECKERS

HYST allows for the same model to be analyzed in several hybrid system model checkers. Each tool has unique characteristics and will have varying performance depending on the features of the model. In this section a brief comparison of the different tools supported by HYST is given, along with their different features. For syntax and semantics, we assume the hybrid automaton framework [3], particularly the same setup as defined for SpaceEx (modulo explicitly allowing nonlinear polynomials for expressions) [26].

2.1 SpaceEx

SpaceEx [26] is a symbolic model checker for affine hybrid systems. It operates on symbolic states comprising of a discrete location and a continuous region. Its reachability algorithm can be represented as a fixed point computation: $\mathcal{S}_0 := \text{post}_c(Init), \mathcal{S}_{i+1} := \mathcal{S}_i \cup \text{post}_c(\text{post}_d(\mathcal{S}_i))$, where $Init$ stands for the initial states of the considered hybrid system, $\text{post}_c(\mathcal{S})$ denotes the continuous post operator and $\text{post}_d(\mathcal{S})$ is the discrete post operator.

The continuous post operator in SpaceEx can be done in one of three scenarios: PHAVer [24], LGG [26], or STC [25]. The PHAVer scenario uses the constraint polyhedra representation. The reachability computation is exact for the class of hybrid systems with piecewise constant dynamics as the PHAVer scenario uses the exact arithmetic, although for affine dynamics the computed reachable sets of states is rather coarse because the tool internally abstracts the affine dynamics with piecewise constant ones. LGG performs better on systems with affine dynamics through computation of linear maps, Minkowski sums and convex hulls, using a support function representation. The STC scenario extends the ideas of the LGG scenario. It introduces the notion of a *flowpipe sampling*. A flowpipe sampling maps every time moment to a polyhedral enclosure of the states reachable at that moment. In other words, the algorithm attributes to every time moment the values of the support function on the predefined template direction set.

Note that the reachability problem for hybrid systems is undecidable. Therefore, the analysis termination is generally not guaranteed. The analysis still always terminates in practice as the analysis time is restricted by the maximum number of iterations, which essentially bounds the number of steps in the fixed point computation.

SpaceEx can also analyze network automata, i.e. multiple interacting automata, which enables compositional verification. The tool supports three reachability scenarios which use different underlying region representations.

PHAVer. This scenario essentially implements the algorithms from the tool PHAVer [24], the redecessor of SpaceEx. PHAVer scenario uses the contraint polyhedra representation and relies on the PPL library [6] for this purpose. The reachability computation is exact for the class of hybrid systems with piecewise constant dynamics as the PHAVer scenario uses the exact arithmetic. However, the reachable set is a rather coarse over-approximation in case of affine dynamics because the tool internally abstracts the affine dynamics with the piecewise constant ones. In contrast to other reachability scenarios, the PHAVer scenario computes continuous post operator for *unbounded time horizon.*

LGG [26]. The scenario is based on the support function representation [34] which allows for the efficient computation of linear maps, Minkowski sums and convex hulls. The support function $\rho_{\mathcal{R}}(\ell)$ of a region $\mathcal{R}$ with respect to the direction $\ell \in \mathbb{R}^n$ is defined as follows:

$$\rho_{\mathcal{R}}(\ell) = \max_{x \in \mathcal{R}} \ell \cdot x$$

Then, the region $\mathcal{R}$ can be over-approximated by a *template* polyhedron D induced by the finite set of template $D = \{\ell_1, \ldots, \ell_m\}$:

$$\mathcal{R}_D = \{x \in \mathbb{R}^n \mid \bigwedge_{\ell_i \in D} \ell_i \cdot x \leq \rho_{\mathcal{R}}(\ell_i)\}$$

The LGG scenario generally scales much better than the PHAVer scenario. On the opposite side, no exact arithmetic is used and thus numerical errors can influence the result. The continuous successors are computed only for the finite *time horizon*. The continuous post operator works by partitioning the time axis into small time intervals defined by the *sampling time*. For each of those time partitions, the post_c operator computes an over-approximation of the states reachable within this particular partition.

STC [25]. This scenario extends the ideas of the LGG scenario. In particular, the STC scenario introduces the notion of a flowpipe sampling. A flowpipe sampling maps every time moment to a polyhedral enclosure of the states reachable at that moment. In other words, the algorithm attributes to every time moment the values of the support function on the predefined template direction set. Furthermore, the support function value is represented by an interval $[r^-(t), r^+(t)]$ where $r^-(t)$ and $r^+(t)$ under- and over-approximate the support function value. By storing the interval values, the algorithm can always ensure that the introduced approximation error is below the user provided

Tool	Dynamics	Extra Inputs	Representation	Problem
SpaceEx (PHAVer)	Affine	n/a	Polyhedra	Unbounded reachability
SpaceEx (LGG)	Affine	Sampling time	Support functions	Unbounded reachability
SpaceEx (STC)	Affine	Sampling time	Support functions	Unbounded reachability
Flow*	Nonlinear	Order Cutoff, Error Interval, Time Step	Taylor models	Bounded reachability
HyCreate	Nonlinear	Min/max derivatives, Time Step	Hyper-rectangles	Bounded reachability
dReach	Nonlinear	Max time, bad states	SMT	Bounded model checking

Table 1: Summary comparison between tools supported by HYST.

threshold.

2.2 HyCreate

HyCreate [7] is a tool which computes reachable states in a way similar to mixed face-lifting [18], combined with support for pseudo-invariants [8]. This technique works by over-approximating the reachable set of states by moving the faces of a tracked polytope outward at the maximum derivative near each face. Neighborhoods around each face are constructed, and then the maximum derivative in the outward direction is considered along each face. Time is advanced in such as way that guarantees no trajectories leave the constructed neighborhoods. After time is advanced, a new polytope provides a bound on the reach state at some future time instant and the process repeats.

In HyCreate, states are tracked as N-dimensional rectangles (intervals aligned with the axes). This can lead to wrapping-effect error [37] for larger time bounds, which is somewhat controlled though splitting tracked boxes into smaller states. However, for large dimensional systems the boxes may need to be split into an exponential number of smaller boxes, such that the work needed to evaluate a certain passage of time keeps increasing. Thus, it works best for low-dimension systems. In HyCreate, a limit can be set to prevent splitting if too many boxes are being tracked, which has the effect of letting the computation proceed at the price of over-approximation error.

A time-step estimate is used in HyCreate that controls the width of each of the neighborhoods. This parameter is varied in our generator to obtain a desired runtime. One benefit of HyCreate is that it can handle nonlinear dynamics. The user inputs the derivative in each mode as a Java code function with provides the maximum and minimum derivative in an arbitrary box of the state space. This is used to determine the maximum outward derivative along each face since the neighborhoods constructed near each face will be N-dimensional rectangles. The invariants and resets are also entered as Java code, which is then compiled and run through the tool. The computation produces a visualization as it's progressing, and can output both the complete set of rectangles as well as a visualization of the reach set projected onto any two dimensions as a .png file of arbitrary size. In our converter, we used version 2.7 of HyCreate, which also supports simulation. Thus, we show visualizations both from the reachability computation, as well as the simulations.

2.3 Flow*

Flow* [15] is a tool which computes reach sets for nonlinear hybrid systems using Taylor models [38] as a state-space representation. The set of states is over-approximated at each point in time using a single Taylor model with an order that is configurable. A Taylor model is a polynomial with interval terms for each of the variables, along with an interval bloating term.

For example, the reachable set of states at a particular time instant, for a two-dimensional system, might be represented by the following Taylor Model, taken from the Flow* manual [14]:

$$\begin{aligned} x &= 1 + b^2 - c + [-0.02, 0.01] \\ y &= a^3 - b + [0, 0.1] \\ a &\in [-0.2, 0.2] \\ b &\in [-0.2, 0.2] \\ c &\in [-0.1, 0.1] \end{aligned}$$

The order of a Taylor model is the maximum number of variables multiplied together in each monomial. In the above Taylor model, the order is 3, due to the a^3 term.

Time is advanced in Flow* by repeated application of the Picard iteration. This requires symbolically integrating the Taylor model, which is not difficult since it is a polynomial (or a polynomial Taylor expansion for non-polynomials). The error interval is bloated to encompass the result after a certain number of iterations (the Taylor model is trimmed to have a maximum order, and the remaining terms are pushed into the remainder interval). Time can then be advanced again from the newly obtained Taylor model. Taylor models work well even for medium-dimension ($\sim$ 10) nonlinear systems, as long as the set of states being tracked remains relatively small. For high accuracy, a high order is desired. However, in high dimensions a lower order might be necessary because the number of combinations of variables grows quickly, so the number of terms in each polynomial can grow as well; it is not uncommon to get hundreds of terms being tracked for each variable.

Flow* allows the order of the Taylor model to be adaptive, depending on the amount of error. This can allow the computation to proceed quickly if the error remains low, while high-accuracy in cases where it is needed. In HYST, we used the adaptive order setting between order 3 and 8. There is also a time-step parameter in Flow* which determines the time step to use in the Picard iteration. This parameter is varied to obtain the desired runtime.

2.4 dReach and dReal

dReach is a tool for bounded reachability analysis (i.e., bounded model checking [BMC]) of hybrid systems [29] that uses the dReal satisfiability modulo theories (SMT) solver for δ-complete decidability queries over the reals [27]. In dReach, after conversion to the SMT representation, the continuous variables of a hybrid automaton are represented as nullary real-valued functions (i.e., symbolic real constants). The locations of a hybrid automaton are also represented as nullary real-valued functions, albeit their constraints ensure they correspond to bounded integers to create a one-to-one correspondence with the finite set of locations. The continuous dynamics are specified as ODEs, and invariants over continuous variables are supported. Sets of states are represented symbolically as formulas over these variables. The inputs to dReach are a hybrid system model, an integer bound $k \geq 0$, and a safety property ϕ, and dReach unrolls the transition and trajectory relations k times to check if ϕ is reachable in *exactly* k iterations.

2.5 Tool Comparison

A summary of the output tools and comparison between them appears in Table 1.

Verification Problem. Tools may be classified by the verification problem they address. All of the SpaceEx scenarios target unbounded reachability for enabling verification primarily of safety properties (invariants). Additionally, as unbounded reachability is a generalization of time-bounded reachability, SpaceEx may solve the time-bounded reachability problem. All the other tools focus on time-bounded verification.

Networks and Compositions of Hybrid Automata. SpaceEx supports composing networks of hybrid automata, whereas the other tools work with flat automata. SpaceEx does include preliminary support for flattening which can be used to interface with the other tools.

State Representation and Error Control. All tools maintain symbolic representations of reachable states, although the specific representation and its possible error control capabilities differ. For example, SpaceEx uses either polyhedra (in PHAVer scenario) or support functions (in STC and LGG scenarios). Flow* works with Taylor models, HyCreate uses sets of hyper-rectangles, and dReach utilizes symbolic formulas in the SMT-LIB standard (effectively nonlinear real arithmetic with transcendentals) [10].

Flows, Invariants, Guards, and Reset Mappings. SpaceEx supports affine functions for flows (as affine ODEs), invariants, and resets. The SpaceEx syntax, however, does not restrict from using nonlinear functions (although the tool itself will not compute reachability for such models). Therefore, we can define nonlinear functions in SpaceEx XML and translate them to the corresponding tool with HYST. Flow*, HyCreate, and dReach support nonlinear flows, invariants, and resets. All tools use may transition semantics, so urgency may be modeled using appropriate invariant and guard conditions.

Comparison Challenges. Given that HYST can translate the same model to multiple tools, a reasonable question is which tool is best for a particular model. A comparison between tools is not a straightforward operation. First, the problem the tools solve differs. SpaceEx typically solves an unbounded verification problem, and typically terminates based on a fixed-point check. Flow* and HyCreate solve time-bounded reachability, while dReach solves time-bounded verification problems in a bounded model checking manner. One way to force termination of SpaceEx to compare to time-bounded reachability tools is to add a time variable $\dot{t} = 1$ and add an invariant in every location that $t \leq T_{max}$. Typically this is achieved by composing the original SpaceEx hybrid automaton model of interest with a single-location timed automaton with $\dot{t} = 1$ and invariant $t \leq T_{max}$, which after composition ensures every location has the time variable with dynamics $\dot{t} = 1$ and invariant $t \leq T_{max}$. Adding such a variable, however, may affect tool performance.

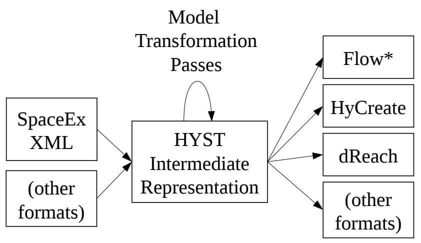


Figure 1: HYST conversion architecture. SpaceEx XML models specifying hybrid automata are translated to an intermediate representation (IR) inside HYST, where model transformation passes are performed. Then, the hybrid automaton is output in different syntax for input to other tools, including HyCreate, Flow*, and dReach. It is not shown, but the SpaceEx configuration file is also used, where, e.g., initial states and potentially bad states are specified.

Another issue is that the tools have various parameters which affect accuracy and performance. The initial model file produced by HYST may not have the optimal settings for the model being considered. For comparison, we believe tool developers need to detect reasonable parameters for models being considered. A general parameter most tools have is the notion of a sampling time or time step. One way to compare which we plan to investigate is to modify this sampling time until the runtimes of the tools are roughly equal, and then examining the resultant output. Finally, the outputs of the tools are typically in the internal representation of the tools, and therefore difficult to compare. A conversion to a common format, for example the bounding box at a point in time, needs to be done prior to comparison.

3. CONVERTER ARCHITECTURE

The conversion architecture used in HYST is shown in Figure 1. The tools takes as input a source file, parses it to an intermediate representation (IR), then prints the resulting output source format desired. Using the SpaceEx format has a number of practical advantages: a number of existing examples already exist in the format, there is a visual model editor for these files, SpaceEx can import from the CIF format [30] and output SpaceEx XML, and there is preliminary support in SpaceEx for hybrid automata flattening. As an input format, the grammar specifying flows, guards, invariants, and resets of the automata does not have any restrictions on being affine, so we use reuse the input format of SpaceEx for HYST and allow for nonlinear expressions. We note that in its current version however, that SpaceEx cannot analyze nonlinear examples as the algorithms it uses do require affine expressions. In terms of output formats, the supported tools are Flow*, HyCreate (both reachability and simulation), and dReach, and output to other tools is part of our planned future work.

Internally, the IR is currently a set of data structures in Java which encode the modes, transitions, continuous variables, flow differential equations, guards, and invariants. The intermediate representation may be modified prior to output for a specific tool's format through model transformation passes. Passes can be viewed as a model-to-model conversions. Some passes can aid in the exporting process. For example, dReach requires that identity resets be explicitly defined, whereas other tools do not. A model transformation pass is therefore run before writing a dReach source file which adds identity resets to transitions which did not define them. Another model transformation pass can be used to check and rename variables that are disallowed keywords for specific tools. Finally, model transformation passes can be used to modify the reachability computation itself. For example, the method of pseudo-invariants [8] has previously been shown to improve the accuracy and speed of the reachability computation for multiple tools. Implementing it as a model transformation pass allows all supported tools to be able to use the technique. An example using this pass is shown later in Section 4. Other candidate passes we plan on implementing including over-approximating abstraction techniques, such as hybridization. Once implemented in Hyst, these techniques would not need to be reimplemented for each tool to use the approach, saving implementation effort and reducing the likelihood of mistakes.

This architecture can effectively extend the allowed input formats for the various tools. For example, compositions of hybrid automata can be automatically flattened for tools which do not support such semantics. However, this may be infeasible for some systems due to the potentially exponential increase in the size of the flattened composition. This is possible because the input format, SpaceEx's XML files, supports a general notion of compositions of hybrid automata through the use of base components and network components, in which base components are instantiated and inputs/outputs are connected appropriately. For example, a typical feedback plant/controller architecture may be modeled in SpaceEx by an automaton for the plant, another for the controller, and then their composition is specified where the outputs of the plant are used as inputs by the controller (and vice-versa).

However, in other tools support for compositions is mixed (e.g., none of Flow*, HyCreate, or dReach support compositions, although other recent tools like Passel [33] do support it). One solution would be to add direct composition support for each of these tools, a large implementation effort. With Hyst, a composed hybrid automaton can immediately be flattened using an option in SpaceEx (particularly `-output-system-file`), then the flattened model can be output to tools which do not natively support such semantics. There are many performance downsides to this approach, although it enables the use of complex compositional models in these other tools as well through the use of our converter.

4. RESULTS

In the introduction, three classes of users were considered for Hyst: (1) Users of verification tools, (2) developers of the

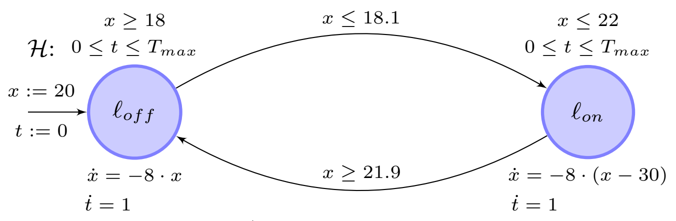
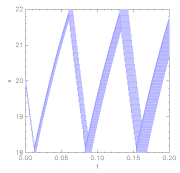
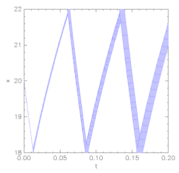
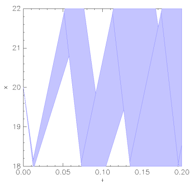
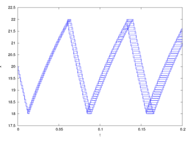


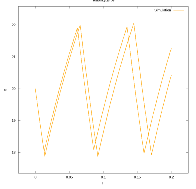
Figure 2: Thermostat/heater example hybrid automaton.

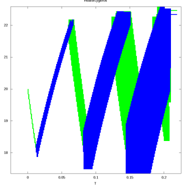

(a) SpaceEx LGG

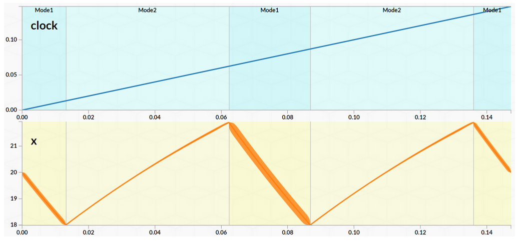

(b) SpaceEx STC


(c) SpaceEx PHAVer


(d) Flow*


(e) HyCreate2 Sim


(f) HyCreate2



(g) dReach

Figure 3: Default reach set visualizations for thermostat/heater hybrid automata example from Figure 2 for the various output model files and tools.

tools, and (3) researchers who develop general techniques which may be applicable to a wide variety of tools.

4.1 Users of Verification Tools

Users can use Hyst as a quick way to create workable model files for all of the supported tools. We used Hyst to convert numerous hybrid system models from different classes, including typical affine and nonlinear examples. For the purposes of illustration, we use the example of a non-deterministic heater system interacting with the temperature in a room measured by a thermostat, as shown in Figure 2 [31]. Although a simple system, it is illustrative of most of the features found in hybrid automata including invariants, guards, flows, and non-determinism. More complicated benchmarks have been converted using Hyst, however this system is sufficient for reach-set illustration.

This system was converted by Hyst into the input format of the various tools. Each tool was then run on the files, producing a reach set as the output. The graphical results for each of the tools are shown in Figure 3. For all tools except dReach, these figures correspond to reachable states. For dReach, the image corresponds to a witness counterexample execution that leads to a bad (goal) set of states. These models serve as initial starting points for users who want to analyze a model, as they can immediately see the rough performance of the various tools. As described earlier, the tools contain tool-specific parameters which, after generation using HYST, can be further tweaked by the user.

We currently have about a dozen benchmarks which can be run through the translator and executed by the various tools. These examples range from biological systems, neuron models, power converters, and typical systems for evaluations of nonlinear reachability methods. The full set of examples and their converted models are available on the HYST website[1]. The conversion process for each model takes less than a second and is negligible when compared with the reachability computation runtime.

4.2 Tool Developers

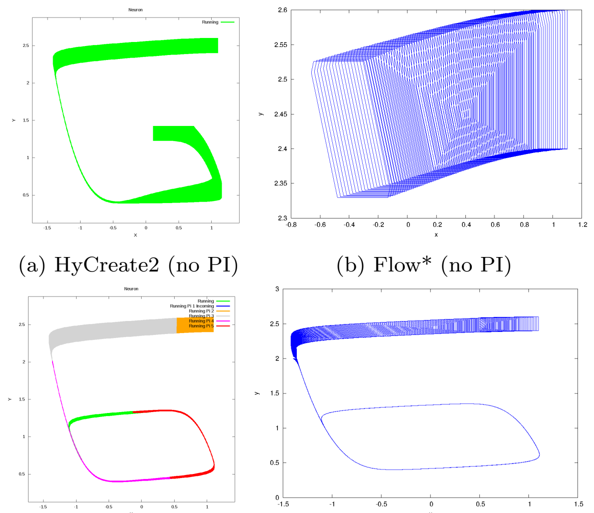
For tool developers such output can also be illuminating. One unexpected finding was that the visualization output for a buck-boost converter differed between HyCreate2 and Flow* for the same model, shown in Figure 4. If the produced reach sets do not intersect, this is indicative of a bug in one of the tools (or in the translation process). In this case, the issue was not caused by the reachability algorithm but rather by the visualization output of Flow* being strictly six digits after the decimal point, which is not sufficient for the time scales considered for this fast-switching system. This was confirmed by rescaling time in the model file (using milliseconds as the X-axis instead of seconds), which corrected the visualization of the reach set.

4.3 Researchers

Many research results are applicable to general hybrid automata models, and not only a specific tool. Reimplementing these results in each tool would require significant effort, and be an error prone process. Piecemeal implementation of such results also is problematic because it is not clear if a tool is superior to another one because of an optimization performed on the model, or the underlying algorithm, or due to subtle differences in the implementation of the technique. By implementing generic model transformations in HYST, effort and errors can be reduced, and a more fine-grained comparison of tools becomes possible.

For example, the time-scaling performed on the buck-boost system in Figure 4 was done using a time-scaling model transformation pass. Using a command-line flag to HYST, the user can select the pass to perform and time scale desired, and the output model will be modified accordingly.

Another model transformation pass implemented in HYST is the insertion of pseudo-invariants (PI) [8]. This method splits an individual mode of a hybrid automaton into several using a set of provided conditions (called pseudo-invariants). The modes after splitting have identical dynamics to the mode they came from, and the transformed automaton is bisimilar to the original automaton. However, these artificial discrete transitions allow accumulating the set of states being tracked for tools that use flow-pipe construction meth-



(a) HyCreate2 (no PI) (b) Flow* (no PI)

(c) HyCreate2 (with PI) (d) Flow* (with PI)

Figure 5: A single implementation of pseudo-invariants within HYST improves reachability computation for both Flow* and HyCreate2.

ods. This can, in certain cases, serve to increase computation accuracy and reduce computation time.

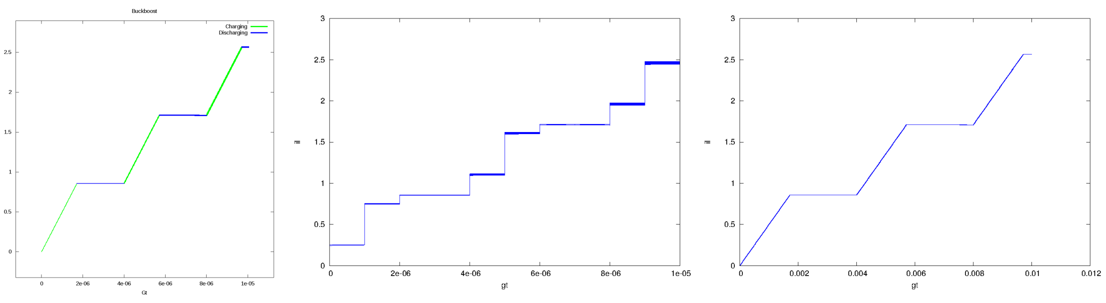
To demonstrate this pass, we used the 2-d nonlinear model of a FitzHugh-Nagumo Neuron, using the dynamics and initial states given by Dang et al. [20]. This system was converted to both Flow* and HyCreate2 (which support nonlinear dynamics). Without pseudo-invariants, neither tool can complete a single cycle within the state-space of the system due to accumulated error. By passing the appropriate flag and parameters to HYST, a modified model is produced, for which an improvement in computation is visible for both of the tools. The reachability plots with and without the PI pass are shown in Figure 5. Other model-transformations passes that we plan to implement are variants of hybridization, both as approximation and inclusions, as well as triangulation-based versions.

5. CONCLUSION

In this paper, we present a source-to-source conversion tool called HYST for hybrid automata. The tool is capable of quickly converting a model to a number of hybrid system model checking tools. Additionally, it supports model transformation passes, which serve to both ease conversion, and allow generic application of model-transformation research results.

As future work, we plan to extend HYST to more case studies and more tools, and we welcome contributions of tool authors interested to integrate in the HYST framework. In particular, we plan to extend the output formats of HYST to other recent hybrid systems analysis tools, such as C2E2 [22], the HyDI language [17] of HyCOMP/nuXmv, Ariadne [9], KeyMaera [41], and other recent tools. he limitations of the converter include a lack of compositions beyond using SpaceEx to compute the composition (which may in general blow up), and some limitations in automation for comparing different tools. Additionally, as the language of SpaceEx corresponds closely to the hybrid automata modeling frame-

[1]HYST and examples are available at: http://verivital.uta.edu/hyst/



(a) HyCreate2 (b) Flow* original (c) Flow* rescaled

Figure 4: The reach set computed by HyCreate2 and Flow* appears to differ. By rescaling time in the Flow* model, this is confirmed as a result of the number of decimal digits Flow* outputs, rather than a bug in the tool.

work, other tools that support more general language definitions may be better suited as the source input format, and we can envision extensions of HYST as a general source-to-source translation framework. Once composition is better supported, we can envision integrating other tools like Passel [33].

6. REFERENCES

[1] A. Agrawal, G. Simon, and G. Karsai. Semantic translation of simulink/stateflow models to hybrid automata using graph transformations. *Electronic Notes in Theoretical Computer Science*, 109:43–56, 2004.

[2] D. N. Agut, D. van Beek, and J. Rooda. Syntax and semantics of the compositional interchange format for hybrid systems. *The Journal of Logic and Algebraic Programming*, 82(1):1 – 52, 2013.

[3] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, pages 209–229, London, UK, 1993. Springer-Verlag.

[4] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specification of hybrid systems in Charon. In N. Lynch and B. Krogh, editors, *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*, pages 6–19. Springer Berlin Heidelberg, 2000.

[5] R. Alur, A. Kanade, S. Ramesh, and K. C. Shashidhar. Symbolic analysis for improving simulation coverage of simulink/stateflow models. In *Proceedings of the 8th ACM International Conference on Embedded Software*, EMSOFT '08, pages 89–98, New York, NY, USA, 2008. ACM.

[6] R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly not closed convex polyhedra and the parma polyhedra library. In *Static Analysis*, pages 213–229. Springer, 2002.

[7] S. Bak. HyCreate: A tool for overapproximating reachability of hybrid automata.

[8] S. Bak. Reducing the wrapping effect in flowpipe construction using pseudo-invariants. In *4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems (CyPhy 2014)*, pages 40–43, 2014.

[9] A. Balluchi, A. Casagrande, P. Collins, A. Ferrari, T. Villa, and A. L. Sangiovanni-Vincentelli. Ariadne: a framework for reachability analysis of hybrid automata. In *PROCEEDINGS OF THE INTERNATIONAL SYPOSIUM ON MATHEMATICAL THEORY OF NETWORKS AND SYSTEMS*, 2006.

[10] C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB standard: Version 2.0, 2010.

[11] L. Carloni, M. D. Di Benedetto, A. Pinto, and A. Sangiovanni-Vincentelli. Modeling techniques, programming languages, design toolsets and interchange formats for hybrid systems. Technical report, 2004.

[12] L. P. Carloni, R. Passerone, A. Pinto, and A. L. Sangiovanni-Vincentelli. Languages and tools for hybrid systems design. *Foundations and Trends in Electronic Design Automation*, 1, 2006.

[13] C. Chen, J. Sun, Y. Liu, J. Dong, and M. Zheng. Formal modeling and validation of stateflow diagrams. *International Journal on Software Tools for Technology Transfer*, 14(6):653–671, 2012.

[14] X. Chen. User's manual of Flow* v1.2.0. http://www-i2.informatik.rwth-aachen.de/i2/fileadmin/user_upload/documents/HybridSystemsGroup/chen/manual-1.2.0.pdf, 2013.

[15] X. Chen, E. Abraham, and S. Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. *2013 IEEE 34th Real-Time Systems Symposium*, 0:183–192, 2012.

[16] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, pages 258–263, 2013.

[17] A. Cimatti, S. Mover, and S. Tonetta. Hydi: A language for symbolic hybrid systems with discrete interaction. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, pages 275–278, Aug. 2011.

[18] T. Dang. *Verification et synthese des systemes hybrides*. PhD thesis, INPG, oct 2000.

[19] T. Dang, C. Le Guernic, and O. Maler. Computing reachable states for nonlinear biological models. In *Computational Methods in Systems Biology*, pages 126–141. Springer, 2009.

[20] T. Dang and R. Testylier. Reachability analysis for polynomial dynamical systems using the bernstein expansion, 2012.

[21] T. Dreossi and T. Dang. Parameter synthesis for polynomial biological models. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*, pages 233–242. ACM, 2014.

[22] P. S. Duggirala, S. Mitra, and M. Viswanathan. Verification of annotated models from executions. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*, EMSOFT '13, Piscataway, NJ, USA, 2013. IEEE Press.

[23] M. Egerstedt. Behavior-based robotics using hybrid automata. In *Hybrid Systems: Computation and Control*, pages 103–116, 2000.

[24] G. Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9-11, 2005, Proceedings*, pages 258–273, 2005.

[25] G. Frehse, R. Kateja, and C. Le Guernic. Flowpipe approximation and clustering in space-time. In *Proc. Hybrid Systems: Computation and Control (HSCC'13)*, pages 203–212. ACM, 2013.

[26] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In *Computer Aided Verification (CAV)*, LNCS. Springer, 2011.

[27] S. Gao, J. Avigad, and E. Clarke. Delta-decidability over the reals. In *Logic in Computer Science (LICS), 2012 27th Annual IEEE Symposium on*, pages 305–314, 2012.

[28] S. Gao, J. Avigad, and E. M. Clarke. δ-complete decision procedures for satisfiability over the reals. In *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, pages 286–300, 2012.

[29] S. Gao, S. Kong, and E. Clarke. Satisfiability modulo ODEs. In *International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, Oct. 2013.

[30] M. Goyal. Translation between cif and spaceex/phaver. Master's thesis, Verimag Research Lab, 2011.

[31] T. A. Henzinger. The theory of hybrid automata. In *IEEE Symposium on Logic in Computer Science (LICS)*, page 278, Washington, DC, USA, 1996. IEEE Computer Society.

[32] C. A. R. Hoare. *Communicating sequential processes*, volume 178. Prentice-hall Englewood Cliffs, 1985.

[33] T. T. Johnson. *Uniform Verification of Safety for Parameterized Networks of Hybrid Automata*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL 61801, 2013.

[34] C. Le Guernic and A. Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 4(2):250–262, 2010.

[35] C. Livadas, J. Lygeros, and N. A. Lynch. High-level modelling and analysis of tcas. In *IEEE Real-Time Systems Symposium*, pages 115–125, 1999.

[36] K. Manamcheri, S. Mitra, S. Bak, and M. Caccamo. A step towards verification and synthesis from Simulink/Stateflow models. In *Proc. of the 14th Intl. Conf. on Hybrid Systems: Computation and Control (HSCC)*, pages 317–318. ACM, 2011.

[37] R. Moore. *Interval analysis*. Prentice-Hall series in automatic computation. Prentice-Hall, 1966.

[38] M. Neher, K. R. Jackson, and N. S. Nedialkov. On taylor model based integration of odes. *SIAM J. Numer. Anal*, 45:2007.

[39] A. Pinto, L. Carloni, R. Passerone, and A. Sangiovanni-Vincentelli. Interchange format for hybrid systems: Abstract semantics. In J. P. Hespanha and A. Tiwari, editors, *Hybrid Systems: Computation and Control*, volume 3927 of *Lecture Notes in Computer Science*, pages 491–506. Springer Berlin Heidelberg, 2006.

[40] A. Pinto, A. L. Sangiovanni-Vincentelli, L. P. Carloni, and R. Passerone. Interchange formats for hybrid systems: Review and proposal. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control*, volume 3414 of *Lecture Notes in Computer Science*, pages 526–541. Springer Berlin Heidelberg, 2005.

[41] A. Platzer and J.-D. Quesel. KeYmaera: A hybrid theorem prover for hybrid systems (system description). In A. Armando, P. Baumgartner, and G. Dowek, editors, *Automated Reasoning*, volume 5195 of *Lecture Notes in Computer Science*, pages 171–178. Springer Berlin Heidelberg, 2008.

[42] S. Ran, J. Lin, Y. Wu, J. Zhang, and Y. Xu. Converting ptolemy ii models to spaceex for applied verification. In X.-h. Sun, W. Qu, I. Stojmenovic, W. Zhou, Z. Li, H. Guo, G. Min, T. Yang, Y. Wu, and L. Liu, editors, *Algorithms and Architectures for Parallel Processing*, volume 8630 of *Lecture Notes in Computer Science*, pages 669–683. Springer International Publishing, 2014.

[43] P. Schrammel and B. Jeannet. From hybrid data-flow languages to hybrid automata: A complete translation. In *Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '12. ACM, 2012.

[44] D. van Beek, M. Reniers, R. Schiffelers, and J. Rooda. Foundations of a compositional interchange format for hybrid systems. In A. Bemporad, A. Bicchi, and G. Buttazzo, editors, *Hybrid Systems: Computation and Control*, volume 4416 of *Lecture Notes in Computer Science*, pages 587–600. Springer Berlin Heidelberg, 2007.

APPENDIX

A. HEATER SYSTEM INPUT AND OUTPUT

In this appendix, we show the tool inputs and outputs for a heater system interacting with the temperature in a room, as shown in Figure 2. Sample input and output file for HYST are below.

A.1 SpaceEx

Listing 1: SpaceEx heater example input system.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<sspaceex xmlns="http://www-verimag.imag.fr/xml-namespaces/sspaceex" version="0.2" math="SpaceEx">
  <component id="ofOnn">
    <param name="x" type="real" local="false" d1="1" d2="1" dynamics="any" />
    <param name="t" type="real" local="false" d1="1" d2="1" dynamics="any" />
    <param name="Tmax" type="real" local="false" d1="1" d2="1" dynamics="const" />
    <location id="1" name="off" x="361.0" y="314.0" width="218.0" height="128.0">
      <invariant>x &gt;= 18 &amp; 0 &lt;= t &amp; t &lt;= Tmax</invariant>
      <flow>x' == -8 * x &amp; t' == 1</flow>
    </location>
    <location id="2" name="on" x="827.0" y="331.0" width="244.0" height="126.0">
      <invariant>x &lt;= 22 &amp; 0&lt;= t &amp; t &lt;= Tmax</invariant>
      <flow>x' == -8 * (x - 30) &amp; t' == 1</flow>
    </location>
    <transition source="1" target="2">
      <guard>x &lt;= 18.1</guard>
      <labelposition x="-31.0" y="3.0" width="76.0" height="50.0" />
      <middlepoint x="579.0" y="381.5" />
    </transition>
    <transition source="2" target="1">
      <guard>x &gt;= 21.9</guard>
      <labelposition x="-38.0" y="-52.0" width="78.0" height="50.0" />
      <middlepoint x="591.0" y="261.5" />
    </transition>
  </component>
  <component id="sys1">
    <param name="x" type="real" local="false" d1="1" d2="1" dynamics="any" controlled="true" />
    <param name="t" type="real" local="false" d1="1" d2="1" dynamics="any" controlled="true" />
    <param name="Tmax" type="real" local="false" d1="1" d2="1" dynamics="const" controlled="true" />
    <bind component="ofOnn" as="ofOnn_1" x="295.0" y="170.0" width="58.0" height="62.0">
      <map key="x">x</map>
      <map key="t">t</map>
      <map key="Tmax">Tmax</map>
    </bind>
  </component>
</sspaceex>
```

Listing 2: SpaceEx heater example input configuration.

```
system = sys1
initially = "x==20 & t==0 & Tmax = 0.20 & loc(ofOnn_1)==off"
forbidden = "x==19 & loc(ofOnn_1)==off"
scenario = supp
directions = oct
set-aggregation = chull
sampling-time = 0.001
time-horizon = 1.0
iter-max = 5
output-variables = "t, x"
output-format = GEN
rel-err = 1.0E-12
abs-err = 1.0E-13
flowpipe-tolerance = 0.001
```

A.2 Flow*

Listing 3: Flow* heater example input file.

```
hybrid reachability
{
 state var x, t

 setting
 {
  fixed steps .001
  time .2
  remainder estimation 1e-4
  QR precondition
  adaptive orders { min 3, max 8 }
  cutoff 1e-15
  precision 53
  output out
  max jumps 99999999
  print on
 }

 modes
 {
  off
  {
```

Listing 3 (Cont.): Flow* heater example input file.

```
   poly ode 1
   {
    x' = -8.0 * x
    t' = 1.0
   }
   inv
   {
    t >= 0
    t <= .2
    x >= 18
   }
  }

  on
  {
   poly ode 1
   {
    x' = -8.0 * (x - 30.0)
    t' = 1.0
   }
   inv
   {
    t >= 0
    t <= .2
    x <= 22
   }
  }
 }

 jumps
 {
  off -> on
  guard
  {
   x <= 18.1
  }
  reset
  {
  }
  parallelotope aggregation {}

  on -> off
  guard
  {
   x >= 21.9
  }
  reset
  {
  }
  parallelotope aggregation {}
 }

 init
 {
  off
  {
   t in [0, 0]
   x in [20, 20]
  }
 }
}
```

A.3 HyCreate

Listing 4: HyCreate heater example input file.

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.7.0_65" class="java.beans.XMLDecoder">
 <object class="containers.HyCreateData">
  <void property="automatonName">
   <string>heaterLygeros</string>
  </void>
  <void property="dimensions">
   <string>x, t</string>
  </void>
  <void property="globalText">
   <string>// Made using SpaceExExport from model file ../../examples/heaterLygeros/heaterLygeros.xml</string>
  </void>
  <void property="initialStates">
   <string>20.0 , 20.0 ; 0.0 , 0.0 ; off</string>
  </void>
  <void property="modes">
   <void method="put">
    <string>off</string>
    <object class="containers.ModeData">
     <void property="derivative">
      <void method="add">
       <string>return new Interval(-8.0 * $x, -8.0 * $x);</string>
      </void>
      <void method="add">
       <string>return new Interval(1.0, 1.0);</string>
      </void>
     </void>
     <void property="invariant">
      <string>return ($t.min &lt;= 0.2 &amp;&amp; $t.max &gt;= 0.0) &amp;&amp; ($x.max &gt;= 18.0);</string>
     </void>
     <void property="minMaxPoints">
      <void method="add">
```

Listing 4 (Cont.): HyCreate heater example input file.

```
       <string>return null; // derivative has no min/max points in any box</string>
      </void>
      <void method="add">
       <string>return null; // derivative has no min/max points in any box</string>
      </void>
     </void>
     <void property="regridRatio">
      <string>1.7, 1.7</string>
     </void>
    </object>
   </void>
   <void method="put">
    <string>on</string>
    <object class="containers.ModeData">
     <void property="derivative">
      <void method="add">
       <string>return new Interval(-8.0 * ($x - 30.0), -8.0 * ($x - 30.0));</string>
      </void>
      <void method="add">
       <string>return new Interval(1.0, 1.0);</string>
      </void>
     </void>
     <void property="invariant">
      <string>return ($t.min &lt;= 0.2 &amp;&amp; $t.max &gt;= 0.0) &amp;&amp; $x.min &lt;= 22.0;</string>
     </void>
     <void property="minMaxPoints">
      <void method="add">
       <string>return null; // derivative has no min/max points in any box</string>
      </void>
      <void method="add">
       <string>return null; // derivative has no min/max points in any box</string>
      </void>
     </void>
     <void property="regridRatio">
      <string>1.7, 1.7</string>
     </void>
    </object>
   </void>
  </void>
  <void property="options">
   <void property="plotOptions">
    <void property="plotXDimensionIndex">
     <int>1</int>
    </void>
    <void property="plotYDimensionIndex">
     <int>0</int>
    </void>
    <void property="visualizeAfterComputation">
     <boolean>false</boolean>
    </void>
    <void property="visualizeDuringComputation">
     <boolean>false</boolean>
    </void>
   </void>
   <void property="reachabilityTime">
    <string>0.2</string>
   </void>
   <void property="simulationOptions">
    <void property="simulationType">
     <object class="java.lang.Enum" method="valueOf">
      <class>containers.ModelSimulationOptions$SimulationType</class>
      <string>REACHABILITY_ONLY</string>
     </object>
    </void>
   </void>
   <void property="timeStep">
    <string>0.001</string>
   </void>
  </void>
  <void property="transitions">
   <void method="add">
    <object class="containers.TransitionData">
     <void property="from">
      <string>off</string>
     </void>
     <void property="guard">
      <string>return $x.min &lt;= 18.1;</string>
     </void>
     <void property="to">
      <string>on</string>
     </void>
    </object>
   </void>
   <void method="add">
    <object class="containers.TransitionData">
     <void property="from">
      <string>on</string>
     </void>
     <void property="guard">
      <string>return ($x.max &gt;= 21.9);</string>
     </void>
     <void property="to">
      <string>off</string>
     </void>
    </object>
   </void>
  </void>
  <void property="versionString">
   <string>File Version 3</string>
  </void>
 </object>
</java>
```

A.4 dReach

Listing 5: dReach heater example input.

```
// Hybrid Automaton in dReach
// Converted from file: ..\..\examples\heaterLygeros\heaterLygeros.xml

//Vars
[0, 1000] time;
[-1000,1000] x;
[-1000,1000] clock;

// off
{
  mode 1;
  invt:
    (and (x >= 18.0) (and (0.0 <= clock) (clock <= 0.2)));
  flow:
    d/dt[x] = -8.0 * x;
    d/dt[clock] = 1.0;

  jump:
    // off -> on (1 -> 2)
      (x <= 18.1) ==> @2(and (x' = x)(clock' = clock) );
}
// on
{
  mode 2;
  invt:
    (and (x <= 22.0) (and (0.0 <= clock) (clock <= 0.2)));
  flow:
    d/dt[x] = -8.0 * (x - 30.0);
    d/dt[clock] = 1.0;

  jump:
    // on -> off (2 -> 1)
      (x >= 21.9) ==> @1(and (x' = x)(clock' = clock) );
}

init: @1 (and (x = 20.0) (and (clock = 0.0) ));

goal: @1 (and (x = 20.0) (and (clock = 0.0) ));
```