The present work was submitted to the LuFG Theory of Hybrid Systems

MASTER OF SCIENCE THESIS

COMPARING THE EXPRESSIVITY AND USABILITY OF HYBRID SYSTEMS' MODELING LANGUAGES

Sabrina Kielmann

Examiners:
Prof. Dr. Erika Ábrahám
apl. Prof. Dr. Thomas Noll

Additional Advisor:
Stefan Schupp

Aachen, March 5, 2018

Abstract

The increasing usage of digital controllers to control continuous systems has resulted in an increased demand of safety verification of such hybrid systems. Over the past two decades various tools have been developed for this purpose, each implementing a different approach. The high number of tools with different approaches and specialties requires a profound comparison to choose a suitable tool. In this thesis we compare state of the art tools for safety verification of hybrid systems. Our comparison focuses on tools implementing flowpipe construction-based methods. Additionally to correctness of results we analyze the expressivity of the model description languages as well as general usability. Additionally a challenging "crash-test" benchmark set is developed to characterize the usability of the tools. To support the communities favorite, but rarely supported interchange format CIF3, the C++ library HyPro for state set representations with the tool HyDRA based on it is extended to this input format.

Eidesstattliche Versicherung

Name, Vorname

Matrikelnummer (freiwillige Angabe)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum

Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Ort, Datum

Unterschrift

Contents

Chapter 1

Introduction

The number of embedded systems in our everyday life increased over the last years and decades. Embedded systems controlling physical quantities are called cyber-physical systems (CPS). An example for a simple CPS already arrived in our lives is a self-regulating heater, which controls the heating based upon the actual temperature. A more experimental CPS is a collision avoiding system controlling self-driving vehicles in a platoon.

Many CPS control safety-relevant systems and therefore have to meet high quality standards. Due to safety reasons certain states must not be reached by the system. To ensure a systems safety the CPS can be modeled with a transition automaton including states describing the unsafe behavior. The physical variables are modeled with ordinary differential equations. This kind of automaton is called hybrid system and the safety of a CPS can be proven with a reachability analysis of its modeled hybrid system.

The reachability problem of the abstracted hybrid system in general is undecidable. Tools for reachability analysis compute therefore over-approximations of the set of reachable states. These are used for safety analysis, if an unsafe behavior is not reachable in the over-approximation it will not be reachable in the systems real behavior.

There are several tools for reachability and safety analysis of hybrid systems using different approaches and thus having different strengths and abilities. For choosing a suitable tool it is necessary to compare the different tools in expressivity and usability.

In this thesis we present a selection of tools to be analyzed and other related work in Chap. 2. Before going into detail, we define in Chap. 3 hybrid automata and introduce different state set representation. In Chap. 4 we compare the expressivity of the tools languages. Additionally we extend the input format of the C++ library HyPro for state set representations with the tool HyDRA based on it. Chapter 5 develops a benchmark set to characterize tools in different aspects. Chapter 6 contains the tools' usability comparison with respect to their analysis parameters, their state set representations, their aggregation methods and experimental results. In Chap. 7 we conclude with a summary of the presented results, discuss them and give an outlook about possible future work.

Chapter 2

Related Work

There are several tools performing reachability analysis of hybrid systems like Ariadne [CBGV12], Cora [AD14], dReach [KGCC15], Flow* [CÁS13], HyCreate [Bak13], HyDRA, HyLAA [BD17a], iSAT [Egg14], KeYmaera [PQ08] and SpaceEx [FLGD+11].

Due to the high quantity of different tools there is a high demand of benchmark suites. Actually there are only few published comparisons and benchmark suites, as [SÁC+15] describing a wide range of tools with different approaches and give their main characteristics. The discussed tools are Ariadne, Cora, dReach, Flow*, HSolver [RS05], iSAT, KeYmaera and SpaceEx. [CSBM+15] develops the first manifold benchmark set and covers the tools dReach, Flow* and SpaceEx.

For a profound analysis of a selection of tools we focus on tools for reachability analysis of linear hybrid systems. Flow*, HyDRA and SpaceEx as flowpipe-constructing tools and HyLAA as a discrete-time tool.

Flow* [CÁS13] is a well studied and established tool which introduced the Taylor model-based flow-pipe construction analysis for (non-)linear hybrid systems. It is the only tool analyzed in the thesis which is also able to analyze non-linear hybrid systems.

HyDRA is a new and soon to be published flowpipe-constructing tool based on the HyPro library [SÁMK17]. HyPro is a free and open-source C++ programming library for linear hybrid systems and uses Flow* like syntax. It offers the possibility to exchange between different state set representations, geometric as well as symbolic ones. HyDRA uses the different representation and a CEGAR-like (Counterexample Guided Abstraction Refinement) approach to improve their reachability analysis. It is the first tool which performs flowpipe construction reachability analysis using counterexamples to improve their result.

HyLAA [BD17a] is a relatively new Python based tool and has a common author with HyCreate and HyST [BBJ15]. It uses discrete time analysis for hybrid systems and performs as first tool simulation-equivalent reachability for hybrid automata with affine dynamics, which is a slightly weaker property than traditional reachability. Its purpose is to be able to analyze much larger scaled benchmarks than other tools before. One of the other inventions is trace-guided set deaggregation, which reduces the size of over-approximation from aggregation.

SpaceEx [FLGD+11] as a well studied and established tool, too, is specialized for large scaled benchmarks and is able to analyze much bigger scenarios than other tools. Its analysis algorithms are an implementation of the Le Guernic-Girard (LGG) algorithm as well as its enhancement (STC).

Comparisons and benchmark suites need a common format to interchange their benchmarks to test them with different tools. In the community of hybrid reachability analysis there is a recommended common interchange format (CIF3) [AvBR13] covering different kinds of automata.
CIF3 is a modeling language not only for hybrid automata but also for a much wider field as an automata-based modeling language for the specification of discrete event, timed, and hybrid systems. CIF3 supports synchronizing events, controllers, observers and monitors, clocks, parallel composition and grouping of automata, and much more specializations. It does offer not only linear arithmetic, but also non-linear, trigonometric and other functions.

Chapter 3

Preliminaries

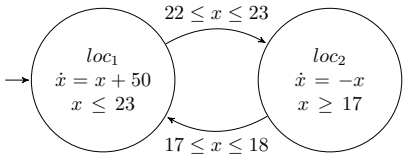
In this chapter we present the required definitions, three examples of hybrid automata and an overview of possible state set representations. The definitions and figures are taken from [Á15] if not stated otherwise.

3.1 Hybrid Automata and Reachability Analysis

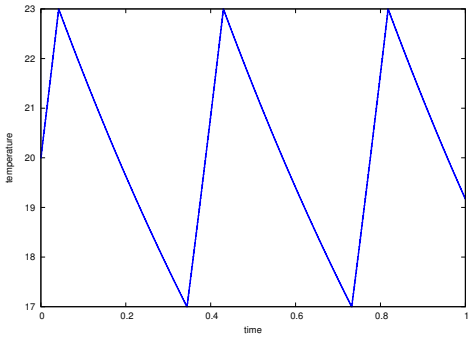
We use hybrid automata to model systems with mixed discrete and continuous behavior. The discrete behavior is modeled with a transition automaton, the continuous behavior with linear ordinary differential equations (ODE). We define a hybrid automaton as follows.

Definition 3.1.1. *Hybrid automaton [CÁS13]*
A hybrid automaton is denoted by a tuple
$\mathcal{A} = (Loc, Var, Inv, Flow, Lab, Trans, Guard, Reset, Init)$ *wherein*

- Loc *is a finite set of discrete states which are also called locations or modes.*
- Var *consists of* n *real-valued variables for some integer* $n > 0$.
- Inv *associates a mode* $l \in Loc$ *an invariant* $I_l \subseteq \mathbb{R}^n$ *such that the variables can only take the values in* I_l *while* $\mathcal{A}$ *is in the mode* l.
- $Flow$ *is a function that associates a mode* $l \in Loc$ *a continuous dynamics* $\dot{x} = f_l(x, t)$ *for* $x \in \mathbb{R}^n$.
- Lab *is the set of labels for the discrete transitions.*
- $Trans \subseteq Loc \times Lab \times Loc$ *is the set of discrete transitions or jumps associated with their labels among the modes.*
- $Guard$ *assigns a transition* $\alpha \in Trans$ *a guard* $G_\alpha \subseteq \mathbb{R}^n$ *such that the jump* α *can be executed if and only if the state variables are of the values in* G_α.
- $Reset$ *associates a transition* $\alpha \in Trans$ *a reset mapping* $\Pi_\alpha : \mathbb{R}^n \to \mathbb{R}^n$ *which updates the values of the variables after the execution of* α.
- $Init \subset Loc \times \mathbb{R}^n$ *is the initial state set of* $\mathcal{A}$. *Every state in* $Init$ *also satisfies the mode invariants.*



(a) Hybrid automaton.



(b) Example execution.

Figure 3.1: Hybrid automaton and example execution of a self-regulating heater.

A state of $\mathcal{A}$ is a pair (l, ν) wherein l denotes the current location and $\nu \in \mathbb{R}^n$ is the valuation of the variables.

Figure 3.1a shows an automaton modeling a self-regulating heater. It consists of two locations, modeling the temperature x while heating loc_1 and not heating loc_2. The temperature in both locations is limited due to the guards, $x \leq 23$ and $x \geq 17$. To change the mode of the heater the automaton needs to take a transition. The transition from loc_1 to loc_2 has a guard $22 \leq x \leq 23$ which ensures that the temperature is high enough before turning the heater off.

A simulation of the behavior of a hybrid automaton includes as well the discrete as the continuous part. Both are simulated differently according to the following definition.

Definition 3.1.2. *Run of a hybrid automaton*
An execution (run) of a hybrid automaton $\mathcal{A}$ is a sequence of states (l_0, ν_0), (l_1, ν_1), (l_2, ν_2), ... which can be infinite such that $(l_0, \nu_0) \in Init$ and for two successive states (l, ν), (l', ν') either of the following two evolutions holds.

- *Discrete evolution (jump)*
  There exists a jump $\alpha = (l,a,l') \in Trans$ such that $\nu \in Guard(\alpha)$, $\nu' \in Inv(l')$, and $Reset(\alpha) = \nu'$.
- *Continuous evolution (time delay)*
  We have $l = l'$ and $Flow(l) = f$ with $f(0) = \nu$, $f(t) = \nu'$ and for all $0 \leq t' \leq t$ it holds $f(t') \in Inv(l)$.

Figure 3.1b shows an example execution of the hybrid automaton from Fig. 3.1a. We see continuous evolution of the hybrid automaton while the plots function increases and decreases. Every change of direction of the plots function marks discrete jumps.

Since we model realistic systems and some behavior is not exact describable through ODEs, we need to add uncertainty. Uncertainty in a hybrid system can be part as well of the affine dynamics of the flow conditions as of the affine functions of the resets.

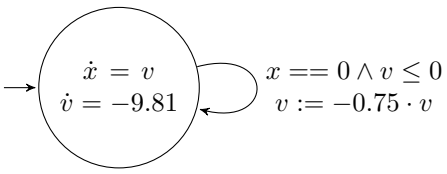


Figure 3.2: Bouncing ball.

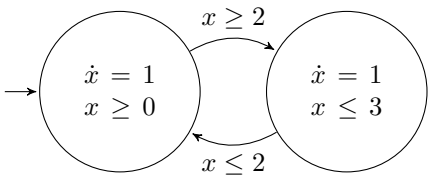


Figure 3.3: Chattering behavior.

Definition 3.1.3. *Uncertainty [FLGD+11]*
A flow condition with time-variant uncertainty is defined by $\dot{x} = Ax + Bu(t) + c$, *where* $Ax + c$ *is autonomous and* $Bu(t)$ *time-variant, for* $u(t)$ *in a bounded convex set* $\mathcal{U}$.
A reset function $\dot{x} = Rx + Sw + v$ *is pure autonomous, but with an uncertain* w *for* w *in a bounded convex set* $\mathcal{W}$.

Since a hybrid system and its execution is an abstraction from a real process and its behavior, non realistic problems can occur, like infinitely many jumps in finite time.

Definition 3.1.4. *Zenoness*
An infinite path fragment π *is Zeno iff it is time-convergent and infinitely many discrete actions are executed within* π. *A path is time-convergent when the sum of its time spend with continuous evolution is finite and thus the execution time is finite.*

In Fig. 3.2 we see a well known example of zeno behavior. A ball drops, bounces from the ground, changes directions and lowers its velocity. Every bounce the velocity decreases with a given factor, but never reaches zero. This results in decreasing time spend between two bounces.

Figure 3.3 shows an artificial automaton which models a second aspect of zeno behavior. While in the bouncing ball example the time between two jumps decreases, in this example are paths, elapsing no time between two jumps. An execution with infinitely many jumps between both locations in no time is non-realizable behavior, since their execution would require infinitely fast processors.

We model and execute a hybrid system to get insights about the systems behavior. We want to know how the temperature changes over time in automaton 3.1a, or if the temperature maybe increases to a life-threatening level. To obtain this information, we compute the reachable states of a hybrid automaton.

Definition 3.1.5. *A state* s *is reachable in a given hybrid automaton* $\mathcal{A}$ *with initial states* $Init$ *when there exists a execution of* $\mathcal{A}$, *starting with a state* $i \in Init$ *and including the state* s *in its sequence of states.*

Definition 3.1.6. *Hybrid reachability problem*
Given a hybrid automaton $\mathcal{A}$, *the reachability problem on* $\mathcal{A}$ *is to verify whether a given state is reachable. A bounded version of the problem is to determine whether the state is reachable in a bounded time interval.*

Since reachability of hybrid systems is in most cases undecidable [HKPV98], reachability analysis often focus on the more scalable (still undecidable) bounded case.

Additional to the above defined hybrid reachability problem, the presented tools implement different definitions of reachability. Two tools use another definition of boundedness which adds a bound for number of jumps.

Figure 3.1b visualizes a subset of the computed reachable states of the self-regulating heater. If we are not only interested in the reachable states but also in the safety of the system, we need to specify the behavior which the system must not reach. For this purpose there are unsafe sets in reachability analysis which are checked to be not reachable.

Definition 3.1.7. *Unsafe set*
An unsafe set is a subset of all possible states from an automaton $\mathcal{A}$. If a state from the unsafe set is reachable, a hybrid system is called unsafe, otherwise it is called safe.

For the reachability analysis performed by the analyzed tools we present a general forward-reachability algorithm, briefly explain two different approaches and give examples of the used state set representations.

```
Input: set Init of initial states
Output: set R of reachable states
Algorithm:
R^new := Init;
R := ∅;
while (R^new ≠ ∅){
  R:= R ∪ R^new ;
  R^new := Reach (R^new )\ R;
}
```

Listing 3.1: General forward reachability analysis algorithm.

In Listing 3.1 we see an algorithm to compute reachable states with forward analysis. It starts with the initial states and computes iterative new states from the actual reachable states. The new states are reached due to continuous and discrete evolution. The presented tools build upon this general algorithm. For a bounded time analysis, a time bound verification is added. The tools use different approaches to implement `Reach`: *flow-pipe construction reachability analysis* and *simulation-equivalent reachability analysis*.

Both approaches use a divide and conquer method with time steps to divide the problem into smaller pieces. The flow-pipe construction approach covers the continuous evolution within one time step with one flow-pipe. In Figure 3.4 we see an example of flow-pipe construction with boxes. The flow-pipe starts in the left blue box and is computed over time. Intersection with the transition guard displayed in red results in two boxes intersecting with the guard. Aggregating the two intersected box segments into one box yields the over-approximated green box. The green aggregated box is the new actual state set and after applying the reset function it is displayed as the right blue box.

The simulation-equivalent reachability analysis uses time-discrete analysis. The time-variant uncertain inputs are meant to be piecewise constant, i.e. a new value $u(t) \in \mathcal{U}$ is only chosen at multiples of the time step. Discrete transitions can only be taken at multiples of the time step. The integrity of the invariant is only checked at the multiples of the time step.

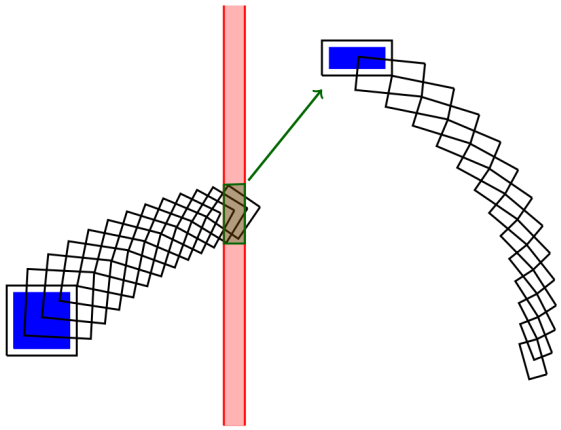


Figure 3.4: Example of flow-pipe construction and intersection with a guard.

3.2 State Set Representations

There are different ways to represent the computed reachable states. They divide into two categories, the geometric and the symbolic state set representations. For every representation we give an example and a brief description.

The geometric state set representations used by the tools in this thesis are boxes, generalized stars, octagons, polyhedra and zonotopes. Boxes and octagons are special cases of polyhedra and generelized stars are a superset of zonotopes.

Definition 3.2.1. *A polyhedron in $\mathbb{R}^n$ is the solution set to a finite number of linear inequalities with real coefficients in n real variables. A bounded polyhedron is called polytope.*

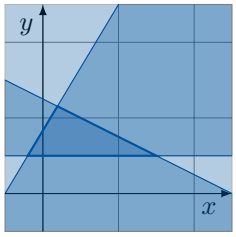
Definition 3.2.2. *A set $\mathcal{S}$ is called convex, if $\forall x,y \in \mathcal{S} : \forall \lambda \in [0,1] \subseteq \mathbb{R} : \lambda x + (1-\lambda)y \in \mathcal{S}$.*

Definition 3.2.3. *Given a set $V \in \mathbb{R}^n$, the convex hull $conv(V)$ of V is the smallest convex set that contains V.*

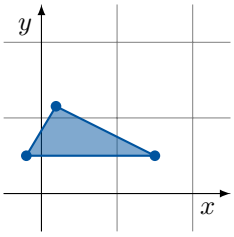
All polyhedra used in this thesis are convex sets. Depending on the form of the representation we distinguish between $\mathcal{H}$-polytopes, $\mathcal{V}$-polytopes and zonotopes.
A $\mathcal{H}$-polytope is described by its defining inequalities. The polytope is the intersection of all halfspaces defined through the inequalities.
A $\mathcal{V}$-polytope is defined by a vertex set. The polytope is the convex hull of all vertices contained in the vertex set.

Figure 3.5 shows the difference of $\mathcal{V}$-polytope and $\mathcal{H}$-polytope representation covering the same variable valuation.

A zonotope or parallelotope is a convex polyhedron defined by a center and a finite number of generator terms. It is always central symmetric to its center. In other words, a zonotope is defined by a center c to which line segments $l_i = \beta_i \cdot g_i, 1 \leq \beta_i \leq 1$ are added via Minkowski sum. This is illustrated in Fig. 3.6.



(a) $\mathcal{H}$-polytope state set representation.



(b) $\mathcal{V}$-polytope state set representation.

Figure 3.5: Comparison of $\mathcal{H}$-polytope and $\mathcal{V}$-polytope representation.

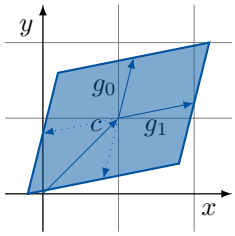


Figure 3.6: Example for zonotope state set representation.

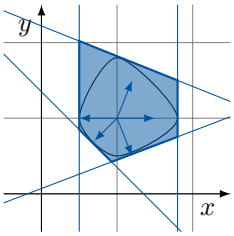


Figure 3.7: Example for support function state set representation.

Definition 3.2.4. *Zonotope / Parallelotope [ASB08]*
A Zonotope is a set

$$Z = \left\{ x \in \mathbb{R}^n | x = c + \sum_{i=1}^{p} \beta_i \cdot g_i, -1 \leq \beta_i \leq 1 \right\}$$

with $c, g_1, \ldots, g_p \in \mathbb{R}^n$. The vectors $g_i, \ldots, g_p$ are referred to as the generators and c as the center of the zonotope.

A generalized star is a convex polyhedron and a superset of the class of zonotopes. Like a zonotope a generalized star is defined by a center and a set of generators. But in contrast to a zonotope the factors β_i are not limited by $\beta_i \in [-1,1]$, but a defining predicate can be chosen. This means in short, a generalized star is like a zonotope but with a predicate $P : \mathbb{R}^p \to \top, \bot$ defining and limiting the factors $\beta = (\beta_i, \ldots, \beta_p)$.

The tools presented in this thesis use two different symbolic state set representation, support functions and Taylor Models.
A support function is an exact representation of a state set. They were first used for hybrid reachability analysis in [LGG09]. Since this representation is not efficient for computing intersections, some tools convert them into (over-approximated) polyhedra. Figure 3.7 shows how a polyhedron can be obtained from a support function. The inequalities for the polyhedron are computed with direction vectors $l_i \in \mathbb{R}^n$ with $l_i \cdot x \leq \rho_{\mathcal{S}}(l_i)$.

Definition 3.2.5. *Support functions [LGG09]*
Let $\mathcal{S} \in \mathbb{R}^n$ be a compact convex set; the support function of $\mathcal{S}$ is $\rho_{\mathcal{S}} : \mathbb{R}^n \to \mathbb{R}$ defined by $\rho_{\mathcal{S}}(l) = max_{x \in \mathcal{S}} l \cdot x$.

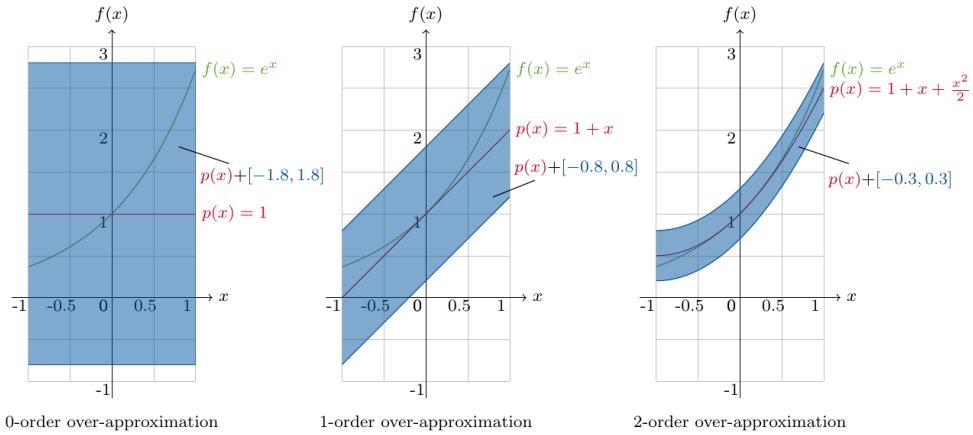


Figure 3.8: Example of Taylor Models over-approximaing e^x in $x \in [-1,1]$.

The other used symbolic representation are Taylor Models, which uses Taylor polynomials and their remainders to represent an over-approximation of a state set.

Definition 3.2.6. *Taylor polynomial*
A Taylor polynomial is a k-order approximation of a smooth function $f : D \to \mathbb{R} \in \mathcal{C}^k$ *which means k times differentiable. It is defined by* $\sum_{n=0}^{k} \frac{f^{(n)}(a)}{n!}(x-a)^n$.

A Taylor Model consist of a pair (p,I) where p is a Taylor polynomial and I is an interval remainder. A Taylor Model covers and over-approximates the variable values for a given time interval. In Fig.3.8 we see an illustration how Taylor Models can over-approximate the function $f(x) = e^t$ in the interval $x \in [-1,1]$.

Chapter 4

Comparing Modeling Languages

Chapter 2 gives a short description of the different tools. In this chapter we look more closely at the underlying languages and their expressivity.
We analyze the languages of the tools Flow*, HyDRA, HyLAA and SpaceEx as well as the Compositional Interchange Format (CIF3). We will present exemplaric code snipplets, for the full inputs files see appendix A.
The tools analyzed in this thesis have two different approaches to store data of the hybrid system. The tools Flow*, HyDRA and HyLAA as well as the language CIF3 store their data in a single file. SpaceEX uses two files to store its data, the modeling file in XML style with the automaton itself and the settings file, including the initial and unsafe sets.

4.1 Variables Declaration

The modeling languages declare the automatons variables differently, see Fig. 4.1 for a syntax comparison.

In Flow*, HyDRA and HyLAA variables names are simply stated in the beginning.

In contrast to the other tools in the modeling language of SpaceEx every variable has attributes additional to its name.

- A `local` variable can only be affected inside the given component.
- The dynamics can be constant (`const`), or arbitrary (`any`) with respect to the conditions of Sec. 4.3.
- The attribute `controlled` is important for composition of two or more components. Simplified, a controlled variable x cannot be modified outside of the component that *owns* it, beyond what is possible in the component itself.
- The attributes `d1` and `d2` specifiy the variables dimensions and for all scalar `real` values this equals `d1="1",d2="1"`. The possibility of vectors seems to be not used by the tool.

Because CIF3 has more data types than the described tools, inter alia booleans, integer, reals, lists and tuples, variables are declared with name and data type.

```
state var x, t
```

(a) Flow* and HyDRA.

```
ha.variables = ["x", "t"]
```

(b) HyLAA.

```
<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" />
```

(c) SpaceEx.

```
cont x;
```

(d) CIF3.

Figure 4.1: Comparison of variable declaration.

4.2 State sets

In a hybrid system there are two important state sets to specify, the initial and the unsafe set. Figures 4.2 and 4.3 show the syntax for initial and unsafe sets in the analyzed languages. Table 4.1 summarizes the result of the following comparison in a shortened overview.

Initial sets For initial sets in a modeling language we consider how many locations are possible in the initial set and how the set of states can be described formally.

The tools HyDRA, HyLAA and SpaceEx as well as the language CIF3 allow more than one location in the initial set. If a tool allows no location in the initial set instantiation, all locations are initial. Only Flow* is limited to one location in the initial set.

For the variable valuations in initial sets every tool has the same power in expressivity, all allow bounded, convex sets and the representations are equivalent.
In Flow* syntax the variable instantiation are expressed with intervals.
In HyDRA syntax the variable instantiation are expressed with intervals and conjunctions of linear constraints.
HyLAA uses conjunction of linear constraints to express the initial values. According to [BD17b] their algorithm is only intended for bounded inputs. But in fact, HyLAA does not abort if a model contains unbounded variables. Thus a computation of a model containing unbounded variables may not be sound and may even not terminate.
SpaceEx is the only tool which allows conjunctions and disjunctions of linear constraints for the variable instantiation.
The modeling language CIF3 has a wider range of possibilities compared to the described tools, e.g. conjunction, disjunctions and negations of linear constraints and also algebraic functions, including trigonometric ones.

	Flow*	HyDRA	HyLAA	SpaceEx	CIF3
Number of locations in initial set	exact one	at least one	at least one	any	any
Variables representation in initial sets	interval	interval, conj. of constr.	conj. of constraints	conj./disj. of constraints	alg. fcts. / logical formulas
Unsafe sets possible	yes	yes	yes[1]	yes	no
Number of locations in unsafe set	any	any	any[1]	any	-
Variables representation in unsafe sets	interval, conj. of constraints	interval, conj. of constraints	conj./disj. of constraints[1]	conj./disj. of constraints	-

Table 4.1: Comparison of state set declaration.

Unsafe sets Unwanted behavior of hybrid systems is modeled by unsafe sets. We compare the number of locations and the valuation representations.

For Flow* we can specify multiple unsafe sets. An unsafe set consist of a location and possibly a conjunction of linear constraints. A location without constraints means all valuations inside this location.
For HyDRA we can specify multiple unsafe sets. Either an unsafe set is a conjunction of linear constraints or a location with possibly conjunctions of linear constraints. A location without constraints means all valuations inside this location.
HyLAA uses an error location concept. The hybrid system has one (or more) sink location which is dedicated to represent the unsafe set. If a automaton must not reach a given valuation in one location, there is a transition with this valuation as its guard leading to the error state.
In SpaceEx there are several options to define the unsafe set. All possible combinations of linear constraints and locations with conjunctions and disjunctions are allowed, including only a single location or a single linear constraint.
In contrast to the tools CIF3 does not support unsafe sets since it is not part of the automaton model.

Conversions If a modeling language does not match the desired specification for the automaton model, the model may be converted to a appropriate one.

If the initial set contains more than one location and the tool is not able to handle this, the automaton model has to be transformed. A new initial location *init* is added to automaton and for every locations *loc* in the initial set new transitions $init \rightarrow loc$ are added. Their reset functions are the origin initial variable valuations of *loc*.
If the initial or unsafe set contains disjunctions of linear constraints and the tool cannot express this, the automaton model has to be transformed. An initial or unsafe set containing location *loc* with linear constraints $constr_1$ and $constr_2$ of the form $loc \wedge (constr_1 \vee constr_2)$ can be transformed into two sets $loc \wedge constr_1$ and $loc \wedge constr_2$.

[1] only one unsafe location, with incoming transitions modeling "unsafe" behavior

```
init {
    on {x in [20, 21]}
}
```

(a) Flow* and HyDRA.

```
def define_init_states(ha):
rv = []

constraints = []
constraints.append(LinearConstraint([-1, 0], -20))
constraints.append(LinearConstraint([1, 0], 21))
constraints.append(LinearConstraint([0, 1], 0))
constraints.append(LinearConstraint([-0, -1], -0))
rv.append((ha.modes['on'], constraints))

return rv
```

(b) HyLAA.

```
initially = "20<=x & x<=21 & loc(heater)==on & t==0 & T==20"
```

(c) SpaceEx.

```
cont x;
initial x >= 20;
initial x <= 21;
[...]
location on:
        initial;
```

(d) CIF3.

Figure 4.2: Comparison of initial sets declaration.

```
unsafe{
    on{x>=30}
    off{x>=30}
}
```

(a) Flow* and HyDRA.

```
_error = ha.new_mode('_error')
_error.is_error = True
trans = ha.new_transition(on, _error)
trans.condition_list.append(LinearConstraint([-1, -0], -30))
trans = ha.new_transition(off, _error)
trans.condition_list.append(LinearConstraint([-1, -0], -30))
```

(b) HyLAA.

```
forbidden = "x>=30"
```

(c) SpaceEx.

Figure 4.3: Comparison of unsafe sets declaration.

	Flow*	HyDRA	HyLAA	SpaceEx	CIF3
Time-variant uncertainty in aff. dynamics	interval arithmetic	-	conj. of constraints[1]	conj. of constraints	conj./disj. of constraints
Reset functions	yes	yes	no[2]	yes	yes

Table 4.2: Comparison of affine dynamics.

4.3 Affine Dynamics

In a hybrid system we find affine dynamics and functions in two components, the flow conditions and the reset functions. As they have a high similarity, their characteristics are analyzed together. Additionally to the affine dynamics, both may contain uncertainty.

From Definition 3.1.3: A flow condition with time-variant uncertainty is defined as $\dot{x} = Ax + Bu(t) + c$, where $Ax + c$ is autonomous and $Bu(t)$ time-variant, for $u(t)$ in a bounded convex set $\mathcal{U}$. A reset function $\dot{x} = Rx + Sw + v$ is pure autonomous, but with an uncertain w for w in a bounded convex set $\mathcal{W}$.

Time-variant uncertainty is a part of flows and conditions. All tools except HyDRA have the same expressivity, they can handle bounded, convex sets representing uncertainty. CIF3 is able to express unbounded, non-convex sets. Figure 4.4 shows an example for a simple affine dynamic flow condition with time-variant uncertainty. The represented ODE is $x' = 2 \cdot x + 2 \cdot u(t) + 3$ with $u \in [-1,1]$.

In Flow* the time-variant uncertainty is expressed by interval arithmetic. It is not possible to multiply the interval with a scalar in the input file. The multiplication has to be done pre-model and stated in interval form.
HyDRA does not support time-variant uncertainty at the moment.
HyLAA syntax uses matrices and vectors to express the affine dynamics and functions. For the bounds of the uncertainty set it uses conjunctions of linear constraints. At the moment time-variant uncertainty is only permitted when there are no invariants in the locations and no discrete transitions.
SpaceEx uses conjunction of linear constraints in the invariants to specify the bounds of a time-variant uncertain variable. The uncertain variables were defined in the components with arbitrary dynamics.
CIF3 uses conjunctions and disjunctions of linear constraints to specify the bounds of a time-variant uncertain variable. The uncertain variables are initialized as normal continuous variables without constraints. The specifying constraints are localized outside the locations with the keyword `invariant`.

[1] only in models with a single location without invariant
[2] resets are not supported yet, but will be in the future

```
x' = 2.0 * x + [-2.0, 2.0] + 3.0
```

(a) Flow*.

```
a_matrix = np.array([ \
    [2], \
    ], dtype=float)
c_vector = np.array([3], dtype=float)
location.set_dynamics(a_matrix, c_vector)

u_constraints_a = np.array([[1], [-1]], dtype=float)
u_constraints_b = np.array([1, 1], dtype=float)
b_matrix = np.array([[2]], dtype=float)
location.set_inputs(u_constraints_a,u_constraints_b,b_matrix)
```

(b) HyLAA.

```
<param name="u" type="real" d1="1" d2="1" local="false"
        dynamics="any" controlled="true"/>
[...]
    <invariant> [...] u&lt;=1 &amp; u&gt;=-1 </invariant>
    <flow> x' == 2*x + 2*u + 3</flow>
```

(c) SpaceEx.

```
cont u;
invariant -1<=u, u<=1;
[...]
equation x'=2*x + 2*u +3;
```

(d) CIF3.

Figure 4.4: Comparison of time-variant uncertainty.

```
inv {x<=23 t<=20}
```

(a) Flow* and HyDRA.

```
on.inv_list.append(LinearConstraint([1, 0], 23))
on.inv_list.append(LinearConstraint([0, 1], 20))
```

(b) HyLAA.

```
<invariant> x &lt;= 23 &amp; t&lt;=20 </invariant>
```

(c) SpaceEx.

```
invariant x <= 23, t <= 20;
```

(d) CIF3.

Figure 4.5: Comparison of invariant declaration.

```
guard {x >= 22}
```

(a) Flow* and HyDRA.

```
trans.condition_list.append(LinearConstraint([-1, -0], -22))
```

(b) HyLAA.

```
<guard> x &gt;= 22 </guard>
```

(c) SpaceEx.

```
edge when x>=22 goto off;
```

(d) CIF3.

Figure 4.6: Comparison of guard declaration.

4.4 Constraints

In a hybrid system apart from the definition of initial and unsafe sets, there are two different kinds of constraints. Invariants specifying valid behavior for each location and guards attached to the transitions. Guards have to be satisfied before taking a transition to another location.
Continuous evolution at locations is limited by invariants, guards are associated to discrete evolution.

All tools use conjunctions of linear constraints to represent their invariants and guards. The tools differ in the used syntax, see Figs. 4.5 and 4.6.
CIF3 has the ability to express constraints with conjunction, disjunctions and negations of various kinds of functions including algebraic and trigonometric ones.

4.5 Parallel Composition

Only SpaceEx and CIF3 are able to express parallel composition. If a tool does not support parallel compositional systems, the hybrid system can be converted into a flat one with only one automaton.

4.6 Implementation of an Additional Input Format for the HyPro Library

The HyPro library and the tool HyDRA based on it use Flow* like syntax as their input formats. With Flow* syntax it is not possible to use disjunctions in state set declarations and parallel composition directly[1]. Since many other tools do not use Flow* syntax, benchmarks have to be transformed into Flow* syntax with possible negative bloating effects.

To solve this issue, a CIF3 parser was added to HyPro. CIF3 outperforms other alternatives, since

- its powerful expressivity, as shown in the previous sections,
- the developers of the transformation tool HyST "would prefer tool developers use standardized interchange formats for their input models like the Compositional Interchange Format (CIF)"[2],
- other tools, e.g. SpaceEx, also supports CIF3 as a possible input format, [GF11]
- its syntax is less complex without losing expressivity.

The alternative to use the de facto standard interchange format, the SpaceEx language[3][4], was discarded due to the immense advantages of a CIF3 implementation. Two disadvantages of the SpaceEx input format compared to CIF3 are: unnecessary variable attributes like dimensions, and the usage of XML compatible math symbols like `&lt=` instead of $\leq$.

Since CIF3 is a very powerful language which covers a wide range of automata and systems, we implemented only a subset. For the first implementation we started with the exact subset recognized by our actual parser.
For the implementation of the CIF3 parser we use ANTLR4 [Par13] as a parser generating tool. The generated parser converts given inputs automatically into abstract syntax trees (AST). From ASTs we compute and derive the hybrid system in the HyPro datastructure.

Additionally the original CIF3 grammar had to be changed due to technical reasons without effect to the language.

- For better usability and maintenance we divided the CIF3 grammar into three subgrammars, hybrid automaton, locations and expressions.
- ANTLR4 builds the AST using the first matching rule. Therefore we changed the order of alternatives for some rules to preserve the language. We also added annotations to some rules' alternatives for simpler identification in the AST.
- Since we uses a subset of the CIF3 language, there are rules without meaning. To handle an input file with one of those rules, the parser aborts and gives a meaningful error message with detailed information.

[1] see Secs. 4.2 and 4.5

[2] from "http://verivital.com/hyst/", 20.01.2018

[3] The ARCH17 workshop, which connects different developers, delivered the benchmark set "Continuous and Hybrid Systems with Linear Continuous Dynamics" [ABC+17] in SpaceEx format.

[4] The benchmarks of continuous and hybrid systems [THS], which are part of the HyPro project, are provided in Flow* and SpaceEx syntax. Even non-linear benchmarks has SpaceEx files.

Besides the advantages of CIF3 as a new input format for HyPro, there are also two aspects not matching the requirements.
CIF3 does only support modeling the hybrid automaton itself and cannot express settings or unsafe sets. Therefore the parser reads a second input file containing this information.
HyPro uses interval arithmetic in the initial states, but CIF3 is not able to use this kind of arithmetic. The required conversion uses conjunctions of linear constraints. This is equivalent to the interval input.

Chapter 5

Developing a Minimal Benchmark Set

Chapter 4 shows differences and similarities in expressivity and syntax of the presented languages. In this chapter we present the first step to compare the tools with experimental results.

Since the considered tools were developed with different theoretical approaches, the tools differ also in their performance. We developed a minimal "crash-test" benchmark set, to test the tools in different situations. With this benchmark set tools can be categorized by their ability to prove safety of the benchmarks. Additional purposes are a forecast from the results to actual performance in other benchmarks and help tool developers improve their tools with respect to the tested aspects.

The benchmarks splits up into three categories: over-approximation due to the dynamics, over-approximation due to jumps and artificial abstraction problems. The benchmarks *LargeInitialSet*, *LargeTVuncertainty* and *ManyVariables* cover different aspects of over-approximation due to the dynamics. *ParallelEdges* and *Many Jumps* cover different aspects of over-approximation due to the jumps. The benchmarks *Chattering* and *ZenoBehavior* cover non-realistic problems occurring due to abstractions.

LargeInitialSet Figure 5.1
The automaton has simple linear affine dynamics, but a large initial set. With larger initial sets, the over-approximation increases. The time-horizon is $T = 5$ with initial valuation $x \in [-10,10]$, $t = 0$ and unsafe set $x \geq 1700$.

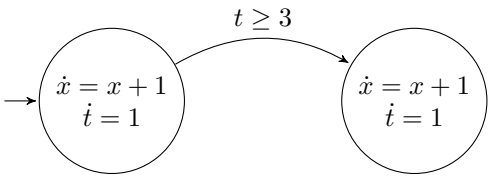


Figure 5.1: Benchmark *LargeInitialSet*.

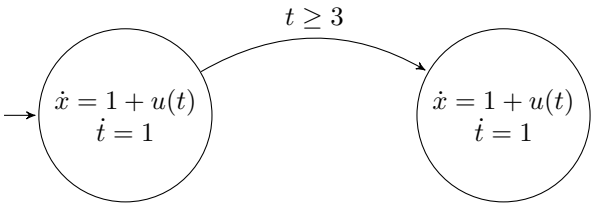


Figure 5.2: Benchmark *LargeTVuncertainty*.

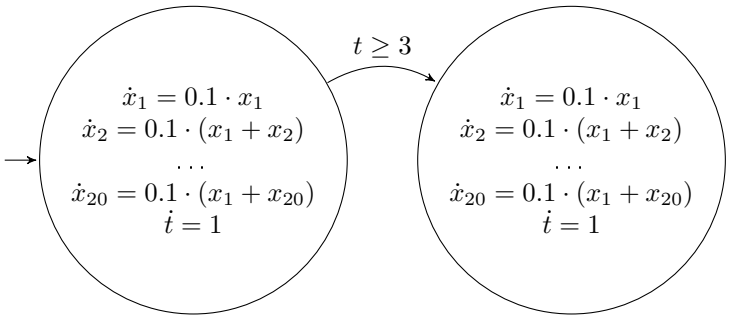


Figure 5.3: Benchmark *ManyVariables*.

LargeTVuncertainty Figure 5.2
This benchmark has a small initial valuation ($x = 0$ and $t = 0$), but a large time-variant uncertainty in the affine dynamics $u(t) \in [-10,10]$. With larger time-variant uncertainty the over-apprimation increases. The time-horizon is $T = 5$ and the unsafe set $x \geq 80$.

ManyVariables Figure 5.3
This benchmark tests a tools capability to handle models with many variables. More variables results in higher computation time and may worsen the over-approximation. The time-horizon is $T = 5$ with initial valuation $x_1 = 1, \ldots, x_{20} = 1$, $t = 0$ and unsafe set $x_1 \geq 2.05$.

ParallelEdges Figure 5.4
This benchmark tests the capability of a tool to handle parallel edges. Proving the safety of this benchmark reveals a smaller over-approximation due to aggregation. The time-horizon is $T = 5$ with initial valuation $x = 1$, $t = 0$ and unsafe set $x \geq 2$.

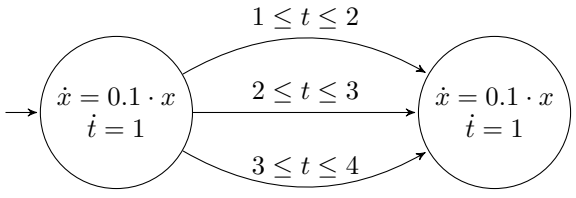


Figure 5.4: Benchmark *ParallelEdges*.

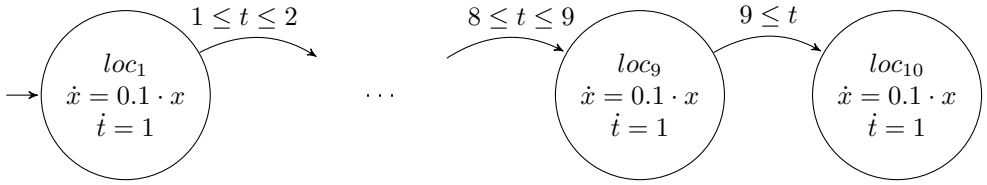


Figure 5.5: Benchmark *ManyJumps*.

ManyJumps Figure 5.5
This benchmark is an automaton with ten locations with the same dynamics. The difficulty to beat in this benchmark is the increasingly over-approximation with every jump. The time-horizon is $T = 11$ with initial valuation $x = 1$, $t = 0$ and unsafe set $x \geq 8$.

Chattering and Zeno Figures 5.6 and 5.7
Some models of physical problems lead to chattering or zeno behavior. In reality this is not possible, therefore it is helpful to detect this unwanted behavior.
The benchmark *Chattering* models chattering behavior around $x = 2$ with time horizon $T = 5$, initial valuation $x = 0$, $t = 0$ and unsafe set $x \geq 6$. The problem to be solved in this benchmark is to detect the chattering behavior to terminate and prove safety within time or reaching a given bound within time, e.g. number of jumps or iterations.
The benchmark *ZenoBehavior* is the bouncing ball example and models the typical zeno behavior. The problem to be solved in this benchmark is to detect the zeno behavior for two reasons: to be able to terminate and prove safety within time or reach a given bound within time, e.g. number of jumps or iterations; with decreasingly value of x the probability increases to run into trouble with the chosen number representation. The time-horizon is $T = 20$ with initial valuation $x = 10$, $v = 0$ and unsafe set $x \geq 11$.

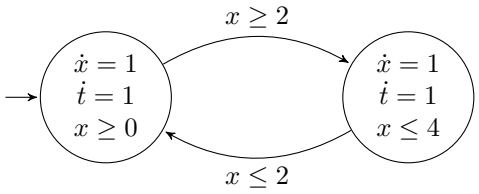


Figure 5.6: Benchmark *Chattering*.

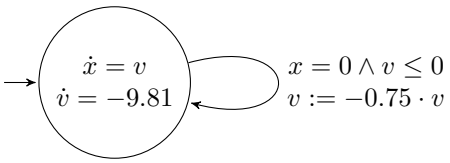


Figure 5.7: Benchmark *ZenoBehavior*.

Chapter 6

Tool Comparison

In this chapter we compare the tools Flow*, HyDRA, HyLAA and SpaceEx with respect to as well theoretical aspects as experimental results on a selected set of benchmarks. The analyzed theoretical aspects are the tools settings, the different state set representations and aggregation methods.
Additionally we give experimental results for three benchmark sets, one with artificial benchmarks covering single aspects, e.g. a large initial set or a large number of variables, one with well known small benchmarks and finally one containing very complex, realistic models.

6.1 Settings

Besides an automaton model for a hybrid system we also need analysis parameter given as settings to perform a reachability analysis for the system. Settings describe how and in which limits an analysis computation runs. Some are identical for every tool, but most options are specific to one tool. In Appendix B we present and explain a selection of the most important settings for every tool.

Figure B.1 shows a selection of the possible settings for Flow*. The options `fixed` and `adaptive` for `steps` and `orders` are mutually exclusive. The output files are written into the directory outputs.
HyDRA uses a subset of the Flow* settings syntax, shown in Fig. B.2.
Since HyDRA has additional functionalities compared to Flow*, the additional settings are added to the command line. These functionalities are clustering, multi threading and the option to choose the state set representation. A description is shown in Fig. B.3
The HyLAA settings are composed of two parts, the settings for the computation, see Fig. B.4, and the plotting settings, see Fig. B.5. For the computational setting, most options have default values thus the user only needs to specify time step size and a time bound.
Figures B.6 describes a selection of possible SpaceEx settings. The STC scenario is a enhancement of the LGG scneario. It "produces fewer convex sets for a given accuracy and computes more precise images of discrete transitions" [Fre12].
For a time bounded computation, the user has to use a clock component or a variable representing the time.

Comparing the settings for the different tools there are the following similarities and differences.
Time steps and time bounds All tools have a time step option. Flow*, HyDRA and HyLAA need a *global* time bound, SpaceEx has an optional *local* time bound. A global time bound defines a time bounded computation in terms of Def 3.1.6. A local time bound is an upper bound on the time span considered for each flow-pipe.
Additional computation bounds Flow* and HyDRA need a bounded number of jumps. SpaceEx offers the option to bound the number of iterations. The iteration bound is the total number of discrete jumps in the computation. These options are not equal since Flow* and HyDRA compute until paths' number of jumps reach the jump bound whereas SpaceEx computes until it reaches in total the iteration bound.
State set aggregation and clustering HyDRA, HyLAA and SpaceEx offers option to turn off aggregation. Flow* and SpaceEx offers choice of aggregation method. Only HyDRA and SpaceEx provide optional clustering before aggregation.
Output options HyLAA is as the only tool able to produce images and videos directly. Flow*, HyDRA and SpaceEx are able to produce vertices list output for further advanced plot generation. Whereas HyLAA does not provide a vertices list output.
Multi-threading Only HyDRA and HyLAA provide multi-threading. SpaceEx has a multi-threated version XSpeed [RGD+15], but does not provide the feature itself.

6.2 State Set Representation and Aggregation

The choice of the state set representation is crucial for the overall result of reachability analysis of hybrid systems, see the definitions and examples in Chap. 3. This and the used aggregation method have great impact on a computations time efficiency and precision.

State Set Representation

Chapter 3 presents and describes the possible state set representations. They divide into two categories, the geometric and the symbolic representations. Boxes, generalized stars, polyhedra and zonotopes are geometric while Taylor Models and support functions are symbolic state set representations.

In the following we present the state set representations used by the different tools.
Flow* uses the symbolic approach of Taylor Models as its state set representation.
HyDRA offers boxes, $\mathcal{V}$-polyhedra, $\mathcal{H}$-polyhedra, zonotopes and support functions as state set representations. The user is free to choose from this representations.
HyLAA uses generalized stars as state set representation. At the moment the defining predicates are limited to conjunctions of linear constraints.
SpaceEx changes it state set representation autonomous operation-depended between support functions and $\mathcal{H}$-polyhedra. SpaceEx uses support functions for computing linear mapping, minkowski sum and convex hulls. For intersection and checking containment, SpaceEx switches to $\mathcal{H}$-polyhedra. The options for polyhedra are boxes, octagons and arbitrary numbered facet polyhedra with given facet bound or facet directions.

State Set Aggregation

Aggregating two (or more) state sets has both good and bad impact on efficiency. In Fig. 3.4 we see an example aggregation of two interseting boxes with a transition guard. Positive for the computation time is the reduced number of state sets. The unavoidable over-approximation during the aggregation is negative for the precision.

Flow* provides two possible approaches for aggregating several Taylor Models. The approaches are interval aggregation and parallelotope aggregation. Interval aggregation computes an interval enclosure for all intersections. For the parallelotope aggregation approach one can specify critical directions. Parallelotopes have the advantage to be easily convertible into a Taylor Model. The aggregation method can be chosen differently for every jump.

HyDRA uses the convex hull as aggregation method for the state sets.
It also offers an optional clustering step before aggregation. The possibilities are no clustering (`-1`), cluster all state sets into one (`0`) and to choose a maximal number of segments which are clustered together. Smaller number indicates higher accuracy.

HyLAA performs an aggregate and deaggregate approach. It does aggregation at every jump, but reaching a new guard, it tries to generate a concrete path to this guard from the initial states. If there is no such path, the actual state set is computed from a too large over-approximated aggregated star. This star will be searched with backtracking and deaggregated into smaller stars from which it was aggregated. The computation continues with the smaller stars. By this approach the tool tries to get the advantages of aggregation, but with less precision loss.

SpaceEx performs a combination of clustering and aggregation for state sets. At first the sets are clustered due to a given clustering percentage. Zero means no clustering at all, `100` that all sets are clustered into one single set. Everything in between reduces the number of groups to be aggregated, smaller values indicate higher accuracy. The aggregation computes the convex hull of each (clustered) group of state sets.

6.3 Experimental Results

In this section we present experimental results benchmarking the different tools. We start with the artificial benchmark set from Chap. 5; followed by the well known benchmarks from [CSBM+15]. Finally we evaluate the benchmark set from the Applied Verification for Continuous and Hybrid Systems workshop 2017 (ARCH17 [ARC17]).

All examples in this section are transformed with HyST [BBJ15], a source transformation and translation tool for hybrid automaton models.
Since we perform benchmarks with tools using different approaches of reachability analysis, the results may not be comparable. Flow*, HyDRA and SpaceEx use flow-pipe construction based reachability analysis whereas HyLAA uses time-discrete simulation-equivalent reachability analysis.

	Flow*	HyDRA	HyLAA	SpaceEx STC	SpaceEx LGG
LargeInitialSet	*fail*	**2.14**	*fail*	*fail*	*fail*
LargeTVuncertainty	0.31	-	-	**0.01**	*fail*
ManyVariables	*fail*	63.44	6.29	**0.22**	*fail*
ParallelEdges	1.12	11.43	*fail*	**0.05**	0.11
ManyJumps	*fail*	**84.80**	*timeout*	*fail*	*fail*
Chattering	370.40	*timeout*	*timeout*	**0.04**	*fail*
ZenoBehavior	**47.27**	*timeout*	-	*fail*	*timeout*

Table 6.1: Minimal benchmark set results, running time in seconds. Legend: -: tool is not able to participate, *fail*: fail to prove safety, *timeout*: running time > 10 min.

6.3.1 Minimal Benchmark Set

In this section we present the experimental results of the minimal benchmark set introduced in Chap. 5. The used settings are shown in Appendix C.

From Table 6.1 we derive the tools performance and usability in different aspects of reachability analysis. The fastest time for each benchmark is marked bold. The tools Flow*, HyDRA and SpaceEx STC are equally successful since they are able to prove safety for the same number of benchmarks. SpaceEx STC is by far the fastest tool in proving safety. There are three distinguishing benchmarks for which only one tool is able to prove their safety.
HyLAA and SpaceEx LGG are least successful since both are only able to prove the safety of one benchmark. HyLAAs *timeouts* are caused by overly used deaggregation.

HyDRA and HyLAA did both not participate in the LargeTVuncertainty benchmark, since it covers aspects the tools cannot handle at the moment. HyDRA is not able to handle time-variant uncertainty. HyLAA is not able to handle time-variant uncertainty with discrete jumps, but a benchmark without jumps is not hybrid but purely dynamic.
HyLAA additionally did not participate in the ZenoBehavior benchmark since it uses resets, which HyLAA is not capable to handle. This kind of zeno behavior can only be modeled with resets.

6.3.2 Well Known Benchmarks

We revisit the benchmark set from [CSBM+15], which covers some well known and used benchmarks. The bouncing ball, the two tank system [RLP98], the rod reactor [AHH96], the 5-D linear switch [THS], the filtered oscillator [FLGD+11] and the continuous vehicle platoon benchmarks [MKA08, MA10]. The versions used in this thesis are taken from [THS].

Two benchmarks were omitted since we already cover them in the ARCH17 benchmark set. In the following we give a brief description of each benchmark.

- The bouncing ball example is a well know benchmark with a single location. A ball drops from a predefined height, when reaching the ground it changes

directions and lowers its velocity. This example can lead to zeno behavior due to the decreasing times between the jumps which never reach zero and is thus time-convergent.

- The two tank system models liquid levels of two tanks with different sources and drains. Tank one has one constant inflow and one with a switch. A drain at the bottom of tank one transports liquid from tank one into tank two. The second tank has two drains, one constantly draining and one with a switch. The model automaton has four location for all possible combinations of the switches. The safety property is to prove that the second tanks liquid level never drops under a certain value.

- The rod reactor example models a reactor tank with two rods cooling it down with different dynamics. This leads to three locations, heating up and cooling down with two different speeds.

- The 5-D linear switch example is a hybrid automaton with uncertain input with five variables and five states with randomly generated convergent dynamics. The transitions are based on the first variables value.

- The filtered oscillator example consists of a 2-dimensional oscillator circuit and a k-dimensional filter. The total number of variables is $k + 2$. The difficulty of this example scales up with its number of variables.

- The 5 / 10 vehicle platoon example is similar to the 3 vehicle platoon benchmark from ARCH17 but with one crucial difference. In this case the communication cannot be lost and it is not a hybrid benchmark, but a pure continuous one.

The experimental results for this benchmark set are shown in Table 6.2. The fastest time for each benchmark is marked bold. The used initial and unsafe sets as well as the used settings are described in Appendix C.

Remarkable in the results of this benchmark set is the very good success rate in proving safety of Flow* and SpaceEx. HyDRA can prove safety for half of the benchmark set. In contrast to the other tools HyLAA participate only in two benchmarks but is the fastest in proving safety of these systems.

Comparing the results from the minimal benchmark sets to the results of the well known benchmark set we see some differences. In contrast to the well known benchmark set, Flow*, HyDRA and SpaceEx STC are equally successful in the minimal benchmark set. The Space LGG scenario was only able to prove safety of one minimal benchmark, but in the well known benchmark set it is very successful in proving safety.

The SpaceEx LGG scenario proves safety for the smaller benchmark as the fastest tool while Flow* is faster in proving safety for the higher dimensioned benchmarks. This may result from the chosen intersection representation and aggregation method for SpaceEx.

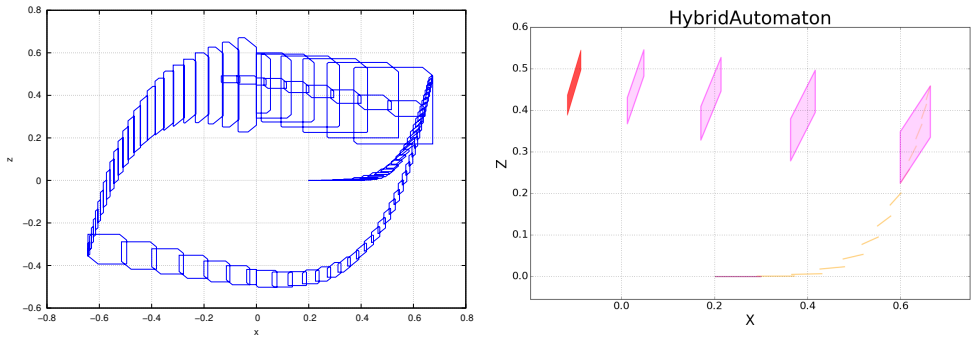
In Figs. 6.2 and 6.3 we see very similar over-approximations of the reachable states for the benchmarks rod reactor and the 5 vehicle platoon.

	Flow*	HyDRA	HyLAA	SpaceEx STC	SpaceEx LGG
bouncing ball	0.42	1.24	-	0.26	**0.11**
two tank system	0.96	-	-	0.35	**0.11**
rod reactor	2.29	10.12	-	3.16	**1.53**
5-D lin. switch	2.56	-	-	0.61	**0.26**
filt. oscillator 4	2.29	**0.85**	-	1.69	1.40
filt. oscillator 8	**7.06**	*fail*	-	10.66	11.86
filt. oscillator 16	**27.13**	*fail*	-	161.82	112.48
filt. oscillator 32	**116.19**	*timeout*	-	*timeout*	*timeout*
5 vehicle platoon	1.19	1.03	**0.06**	3.79	3.19
10 vehicle platoon	2.50	2.93	**0.08**	15.50	23.16

Table 6.2: Well known benchmarks benchmark set results, running time in seconds. Legend: -: tool is not able to participate, *fail*: fail to prove safety, *timeout*: running time $>$ 10 min.

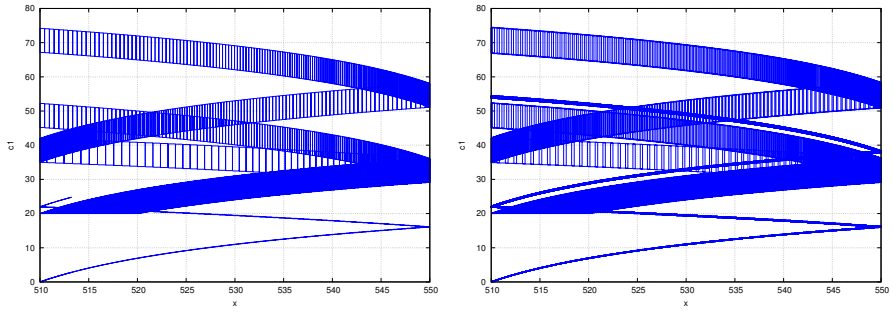
The rod reactor plots in Fig. 6.2 start at $c_1 = 20$ with an initial temperature of $x \in [510{,}520]$. Reaching a temperature $550[°C]$ or $510[°C]$ the automaton model takes a discrete transition to start or end cooling down.
The benchmark plots in Fig. 6.3 show the relative distances in platoon from the first vehicle to the leader (e_1) and between the first and the second vehicle (e_5). The plots starts around (1,1) and both relative distances increase. While decreasing of the relative distances, the possible valuation sets also decrease.

HyLAA does not participate in the filtered oscillator benchmarks although it should be able to prove its safety, but the produced outcome did not match the expectations, see Fig. 6.1. Since HyLAA participates only in benchmarks without jumps in the non-artificial benchmarks we can see its good performance in computing affine dynamics, but cannot compare its performance for discrete transitions to the other tools.



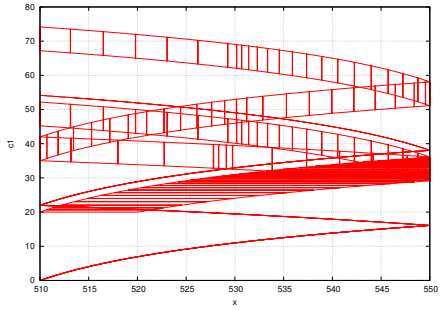
(a) Flow* (b) HyLAA

Figure 6.1: Comparison of Flow* and HyLAA for the filtered oscillator 4 benchmark.

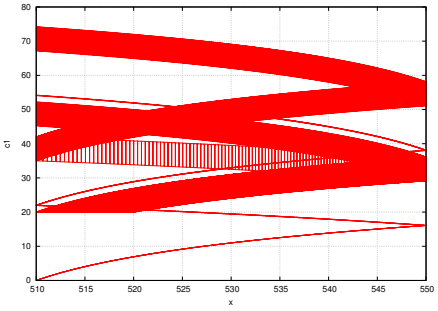


(a) Flow*

(b) HyDRA

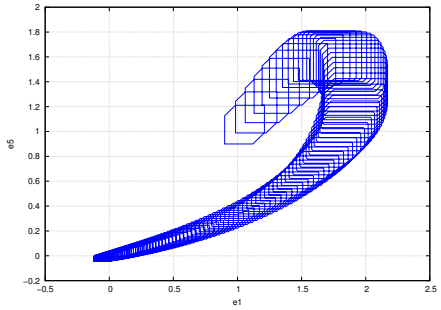


(c) SpaceEx STC

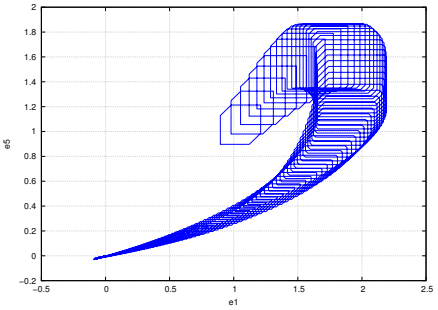


(d) SpaceEx LGG

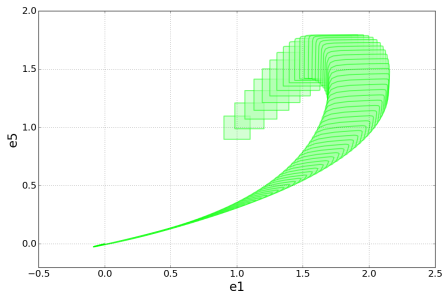
Figure 6.2: Results for the rod reactor benchmark.



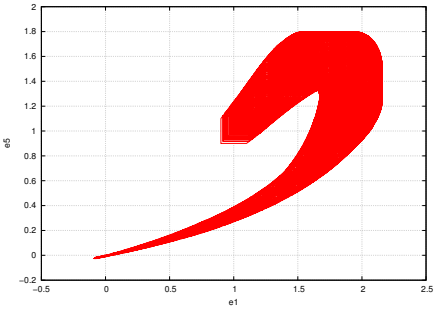
(a) Flow*



(b) HyDRA



(c) HyLAA



(d) SpaceEx STC

Figure 6.3: Results for the 5 vehicle platoon benchmark.

6.3.3 ARCH17 Benchmark

This benchmark set was part of the ARCH17 workshop, as a friendly competition for tools with different areas of applications. We choose the suitable benchmark set for our application area: Continuous and Hybrid Systems with Linear Continuous Dynamics.
There are three scenarios with different models and specifications.
Some of the most difficult benchmarks were not solved by any of the tools correctly and in time, we omit these in our results. Instead we will present the results of six benchmarks for which at least one tool got a correct result in time.

In the following the benchmarks are described briefly, for further information see [ABC+17].

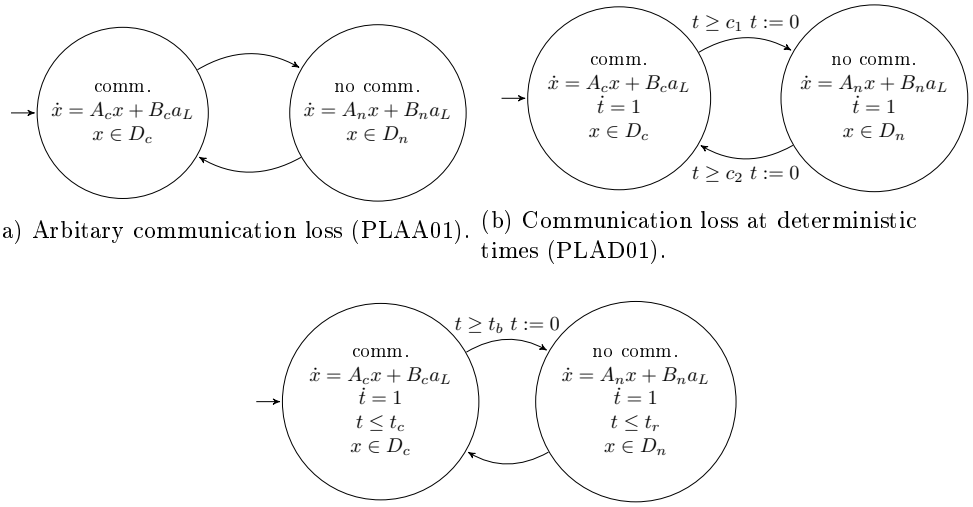
The **building benchmark** is a large scaled simulation of movement in the Los Angeles University Hospital which has eight floors each with three degrees of freedom, namely displacements in x and y directions and rotation. This results in 24 variables, but due to practical problems the affine dynamics has to be transformed into a model with 48 variables.
The description of the concrete affine dynamics and the uncertain part can be found in [TNJ16]. This benchmark has only one location without jumps and can be thus modeled purely linear continuous.
There are two models, one in which the uncertain inputs can change arbitrarily over time (BLDF01). In the other model the inputs are only uncertain in their initial value and constant over time: $u(0)\mathcal{U}$, $\dot{u}(t) = 0$, modeled with an additional constant variable in the initial set (BLDC01).
There are three unsafe set specifications used in both models. We show only the results for the one safe specification (BDS01): For all $t \in [0, 20]$, $x_{25}(t) \leq 5.1 \cdot 10^{-3}$. The other two provided specifications are only indented as sanity checks. They should be computed with the same settings as the safety property.

The **platoon benchmark** [MK14] simulates the distance of three vehicles driving in a platoon. It is modeled with two locations and time-varying input variables describing the leaders behavior. The variables describe for every vehicle the relative distance to its predecessor, its relative velocity, and the acceleration of the vehicle.
The vehicles communicate among each other. There are three models describing the communication loss:

- PLAA01: loss is arbitrary over time,
- PLADxy: loss occurs at fixed times with counters for loss and reestablishment,
- PLANxy: loss occurs at non-deterministic times in defined delimiters.

In PLAD01 counters c_1 and c_2 for loss and reestablishing of the communication are both set to 5; in PLAN01 the loss occurs in $t \in [t_b,t_c]$ for $t_b = 10$ and $t_c = 20$, the communication is reestablished not later than $t_r = 20$ after loss.
For a abstraction of the communication loss models see Fig. 6.4.

The given specifications are safety properties for the distance between the vehicles, which should be at least a given minimum. There are two classes of specifications, the



(a) Arbitary communication loss (PLAA01).

(b) Communication loss at deterministic times (PLAD01).

(c) Communication loss at nondeterministic times (PLAN01).

Figure 6.4: Communication loss for the platoon benchmark.

bounded (BNDxy) case with time horizon $T = 20$ and the unbounded case (UNBxy) with no time horizon.
Every specification sets their minimum d_{min} distance with xy= d_{min}. The safety property is defined as: every distance between the following vehicles is greater than or equal to the minimal distance given. In the benchmark set there are minimal distances of 50, 42 and 30 for both the bounded as well the unbounded case chosen. The formal safety property is to prove for the relative distances x_1, x_4 and x_7: $x_i \geq -d_{min}$. The choice of the coordinate system is such that the minimum distance is a negative value.
We present for every benchmark model only the smallest verified bound.

The **gearbox benchmark** [CMT15] models the motion of two meshing gears. Shifting gears in a motor consists of a sleeve-gear disengaging from the actual gear and meshing with the new gear. This benchmark models the sleeves meshing.

Meshing two gears can lead to different problems. If the gears are in a bad position, the meshing process can be too slow. During the meshing process the cumulated impulse can gets to high and one gear may break.

The benchmark model (GRBX01) simulates trajectory of the sleeve relative to the new gear during the meshing process. The position of the first gear is modeled as time-invariant uncertain input.

The safety specification (MES01) checks two aspects. First, whether the meshing process is fast enough and reaches the meshed location within $t < 0.2$ seconds. Secondly it checks whether the cumulated impulse is less than $20[Nm]$ over the whole time.

The **results** of all benchmarks with their models and specifications are shown in Table 6.3. In contrast to the results from [ABC+17], in this thesis the computation times are much higher in the building benchmark. Therefore we decided to split both

	Flow*	HyDRA	HyLAA	SpaceEx STC	SpaceEx LGG
BLDC01-BDS01 T=1	487.92	5.89	**0.19**	5.22	3.61
BLDC01-BDS01 T=20	*timeout*	135.17	**3.47**	23.51	26.76
BLDF01-BDS01 T=1	430.06	-	**0.23**	5.27	4.70
BLDF01-BDS01 T=20	*timeout*	-	**4.25**	33.99	59.62
PLAD01-BND42	9.52	-	-	**5.70**	5.11
PLAN01-BND50	168.26	-	-	**30.90**	78.42
PLAD01-UNB50	-	-	-	**10.96**	16.91
GRBX01-MES01	7.14	-	-	**0.14**	*timeout*

Table 6.3: ARCH17 benchmark set results, running time in seconds. Legend: -: tool is not able to participate, *fail*: fail to prove safety, *timeout*: running time > 10 min.

building benchmarks into two benchmarks with different time horizons. The fastest time for each benchmark is marked bold. The used settings are described in Appendix C.

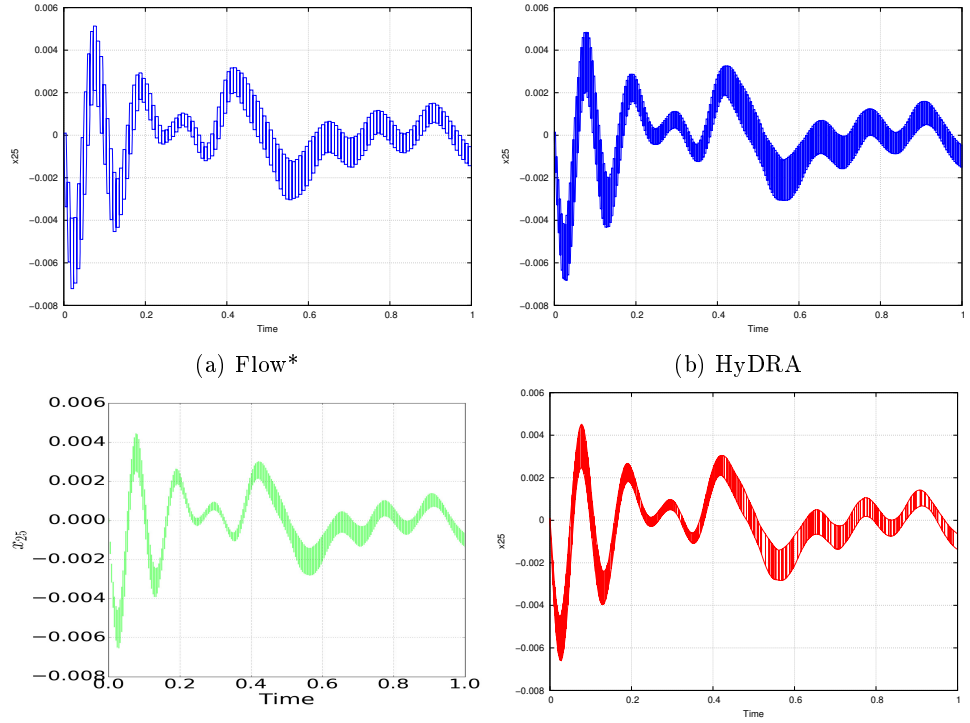
Flow* is the slowest tool for this benchmark set since it has worse running times than all other successful tools.[1] This may results from the tools design to analyze non-linear dynamics.
HyDRA is the most limited tool in this benchmark set with respect to the coverage.
HyLAA only participated in one scenario with two benchmark models, but performed with the best running times.
SpaceEx is the most versatile tool since it computes the most results for this benchmark set, but with a lack of speed in the building scenario[2]. The STC scenario seems to be the better choice, since it finishes in every but one benchmark faster than the LGG scenario.

The tools results match the forecast from the results of the minimal benchmark set.
HyDRAs and HyLAAs limited number of participated benchmarks in the ARCH17 benchmark set corresponds to not participating in *LargeTVuncertainty*. The difference in computational time between Flow* and SpaceEx are apparent in both benchmark sets. Flow*s inability to compute the building benchmark with time horizon $T = 20$ are foreseen in the inability to prove the safety of the *ManyVariables* benchmark.

From Figs. 6.5 and 6.6 we conclude that all tools have a similar level of precision. In Fig. 6.5 we see for the building benchmark the movement in direction x_{25} over time. Figure 6.6 displays the relative distance e_1 over time with communication loss at deterministic times. We see the distance rapidly increasing at the beginning of every communication loss and reestablishment.

[1] In contrast to results from [ABC+17], the author of this thesis was not able to compute the building benchmark with Flow* for $T = 20$ within time (for a continuous model the memory of 8GB plus 8GB swap was not sufficient, for hybrid model the computation had a timeout).

[2] In the original ARCH17 paper the much faster single direction option was used for the building benchmark, but it computes an unbounded set which cannot be printed.



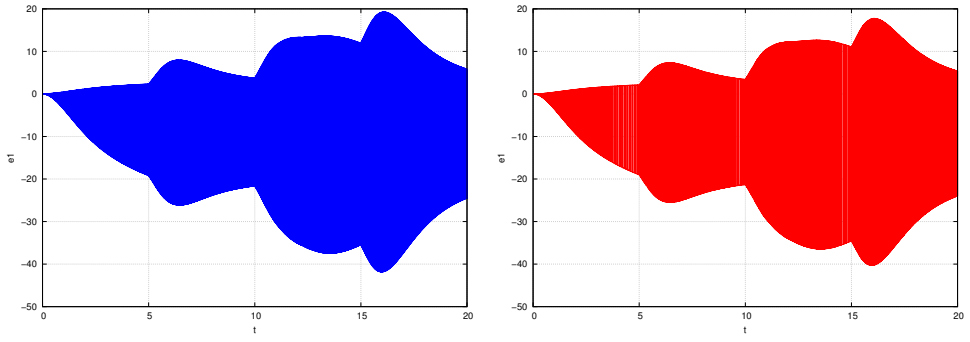
(a) Flow*

(b) HyDRA

(c) HyLAA

(d) SpaceEx

Figure 6.5: Results for the BDC01 benchmark, movement in direction x_{25} over time.



(a) Flow*

(b) SpaceEx

Figure 6.6: Results for the PLAD01 benchmark, relative distance $e1$ over time.

Chapter 7

Conclusion

The goal of this thesis was to compare expressivity and usability of hybrid systems modeling languages. Besides theoretical aspects also practical application of reachability analysis was taken into account.

As presented in Chap. 4-6 we did a profound analysis of the input languages of Flow*, HyDRA, HyLAA, SpaceEx and CIF3, implemented a CIF3 parser for HyPro, developed a minimal "crash-test" benchmark set and analyzed experimental results for different benchmark sets.

We compared the input languages of said tools. Our comparison thereby focused on how affine dynamics, uncertainty, state sets, constraints and parallel composition can be expressed.
CIF3 and the specification language used in SpaceEx are the most versatile and expressive languages. The other languages can obtain equivalent models with model transformations, but with possible negative bloating effects. Exceptions are as well resets and time-variant uncertainty in combination with invariants or transitions for HyLAA as time-variant uncertainty in general for HyDRA. These are at the moment not handleable by the tools.

The CIF3 parser implemented in this thesis for HyPro extended the possible input formats of the library for future extensions and better interchangeability of benchmarks.

In the developed minimal benchmark set each benchmark tests one special aspect which may occur in real world scenarios. With the result tools can be characterized due to their ability of proving safety of the contained benchmarks. To be versatile in modeling realistic systems, a tool must be capable of handling large uncertainty, a large number of variables and a large number of jumps. Additionally a tool should detect and deal with chattering and zeno behavior.

The usability analysis splits into three parts: the possible analysis parameters, the state set representations with aggregation methods and experimental results.
Overall all tools have similar options for their analysis parameters, but with some special features and restrictions. SpaceEx is the only tool able to verify safety over

unbounded time for systems which converge to a fixpoint. Flow* and HyDRA can only verify safety for a bounded number of jumps. Only HyDRA and HyLAA can speed up their computation with multi-threading (for SpaceEx exists a separate multi-threated version).
The analysis of the offered state set representations and aggregation methods yields HyDRA as the most versatile tool. It offers four different types of geometric and one symbolic state set representation. Additional to aggregation it offers the possibility of prior clustering for discrete jump successor computation. The other tools offer only one state set representation, and only SpaceEx is able to cluster state sets, too.

For a profound analysis of the usability the tools are executed on different benchmark sets.
In the minimal benchmark set Flow*, HyDRA and SpaceEx STC are equally successful with different strength. HyLAA and SpaceEx LGG are least successful proving safety only for one benchmark.
In the realistic and complex benchmark set SpaceEx has the most successful computed benchmarks. HyLAA participated only in two benchmarks, but was significant faster.
In a set of well known benchmarks Flow* and SpaceEx as well with STC as with LGG outperform the other two tools. While HyLAA participate only in two benchmarks, HyDRA is able to prove safety of half of the benchmarks.

The results from the minimal benchmark set explain the results of the ARCH17 quite good, but for the well known benchmarks the forecast fails. It is remarkable that the established tools are most successful in the well known benchmarks. Since they are well known, these tools had the possibility to measure and tune their performances on them for many years.
Since in the minimal benchmark set every tool failed to prove safety for at least two benchmarks, the in this thesis developed minimal benchmark set can be used as a new challenging benchmark set to actual distinguish tools' performances.

7.1 Future Work

Not all challenges are addressed in detail in this thesis. We see several aspects which could be focused on in future development.

Analyzing a wider range of tools would improve and extend the language and tool comparison with respect to diversity. Interesting candidates for further analysis are KeyMaera as an interactive hybrid tool combining deductive, real algebraic, and computer-algebraic prover technologies and dReach as a bounded model checking tool using constraint solving techniques for (non-)linear hybrid systems.

Extending the minimal benchmark set to other aspects of reachability analysis would improve its challenging and distinguishing character. These aspect may be non-linear dynamics, urgent transitions or include compositional hybrid systems.

A profound analysis of the relationship of traditional and simulation-equivalent reachability analysis may improve the quality of comparing tools using different kinds of reachability analysis approaches.

Bibliography

[Á15] Erika Ábrahám. Lecture notes: Modeling and analysis of hybrid systems. 2015.

[ABC+17] Matthias Althoff, Stanley Bak, Dario Cattaruzza, Xin Chen, Goran Frehse, Rajarshi Ray, and Stefan Schupp. Arch-comp17 category report: Continuous and hybrid systems with linear continuous dynamics. In *ARCH17*, volume 48 of *EPiC Series in Computing*, pages 143–159. EasyChair, 2017.

[AD14] Matthias Althoff and John M Dolan. Online verification of automated road vehicles using reachability analysis. *IEEE Transactions on Robotics*, 30(4):903–918, 2014.

[AHH96] Rajeev Alur, Thomas A Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.

[ARC17] *ARCH17. 4th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 48 of *EPiC Series in Computing*. EasyChair, 2017.

[ASB08] Matthias Althoff, Olaf Stursberg, and Martin Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 4042–4048. IEEE, 2008.

[AvBR13] DE Nadales Agut, Dirk A van Beek, and JE Rooda. Syntax and semantics of the compositional interchange format for hybrid systems. *The Journal of Logic and Algebraic Programming*, 82(1):1–52, 2013.

[Bak13] Stanley Bak. Hycreate: A tool for overapproximating reachability of hybrid automata. *Retrieved January*, 17:2016, 2013.

[BBJ15] Stanley Bak, Sergiy Bogomolov, and Taylor 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*, pages 128–133. ACM, 2015.

[BD17a] Stanley Bak and Parasara Sridhar Duggirala. Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, pages 173–178. ACM, 2017.

[BD17b] Stanley Bak and Parasara Sridhar Duggirala. Simulation-equivalent reachability of large linear systems with inputs. In *International Conference on Computer Aided Verification*, pages 401–420. Springer, 2017.

[CÁS13] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263. Springer, 2013.

[CBGV12] Pieter Collins, Davide Bresolin, Luca Geretti, and Tiziano Villa. Computing the evolution of hybrid systems using rigorous function calculus. *IFAC Proceedings Volumes*, 45(9):284–290, 2012.

[CMT15] Hongxu Chen, Sayan Mitra, and Guangyu Tian. Motor-transmission drive system: a benchmark example for safety verification. In *ARCH14-15*, volume 34 of *EPiC Series in Computing*, pages 9–18. EasyChair, 2015.

[CSBM+15] Xin Chen, Stefan Schupp, Ibtissem Ben Makhlouf, Erika Abraham, Goran Frehse, and Stefan Kowalewski. A benchmark suite for hybrid systems reachability analysis. In *Proc. of the 7th NASA Formal Methods Symp. (NFM'15)*, volume 9058 of *LNCS*, pages 408–414. Springer, 2015.

[Egg14] Andreas Eggers. *Direct handling of ordinary differential equations in constraint-solving-based analysis of hybrid systems*. PhD thesis, Universität Oldenburg, 2014.

[FLGD+11] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV)*, LNCS. Springer, 2011.

[Fre12] Goran Frehse. A brief experimental comparison of the stc and lgg analysis algorithms in spaceex. 2012.

[GF11] Manish Goyal and Goran Frehse. Translation between cif and spaceex. Technical report, PHAVer, Technical Report, MULTIFORM consortium, 2011.

[HKPV98] Thomas A Henzinger, Peter W Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.

[KGCC15] Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. dreach: δ-reachability analysis for hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 200–205. Springer, 2015.

[LGG09] Colas Le Guernic and Antoine Girard. Reachability analysis of hybrid systems using support functions. In *CAV*, volume 5643, pages 540–554. Springer, 2009.

[MA10] Jan P Maschuw and Dirk Abel. Longitudinal vehicle guidance in networks with changing communication topology. *IFAC Proceedings Volumes*, 43(7):785–790, 2010.

[MK14] Ibtissem Ben Makhlouf and Stefan Kowalewski. Networked cooperative platoon of vehicles for testing methods and verification tools. In *Proc. of ARCH14-15*, pages 37–42, 2014.

[MKA08] Jan P Maschuw, Günter C Keßler, and Dirk Abel. Lmi-based control of vehicle platoons for robust longitudinal guidance. *IFAC Proceedings Volumes*, 41(2):12111–12116, 2008.

[Par13] Terence Parr. *The definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013.

[PQ08] André Platzer and Jan-David Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In *International Joint Conference on Automated Reasoning*, pages 171–178. Springer, 2008.

[RGD+15] Rajarshi Ray, Amit Gurung, Binayak Das, Ezio Bartocci, Sergiy Bogomolov, and Radu Grosu. Xspeed: Accelerating reachability analysis on multi-core processors. In *Haifa Verification Conference*, pages 3–18. Springer, 2015.

[RLP98] Marcus Rubensson, Bengt Lennartson, and Stefan Pettersson. Convergence to limit cycles in hybrid systems-an example. *IFAC Proceedings Volumes*, 31(20):683–688, 1998.

[RS05] Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. In *International Workshop on Hybrid Systems: Computation and Control*, pages 573–589. Springer, 2005.

[SÁC+15] Stefan Schupp, Erika Ábrahám, Xin Chen, Ibtissem Ben Makhlouf, Goran Frehse, Sriram Sankaranarayanan, and Stefan Kowalewski. Current challenges in the verification of hybrid systems. In *International Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems*, pages 8–24. Springer, 2015.

[SÁMK17] Stefan Schupp, Erika Ábrahám, Ibtissem Ben Makhlouf, and Stefan Kowalewski. Hypro: A c++ library of state set representations for hybrid systems reachability analysis. In *NASA Formal Methods Symposium*, pages 288–294. Springer, 2017.

[THS] Theory of Hybrid Systems RWTH Aachen University, Benchmarks of continuous and hybrid systems. `https://ths.rwth-aachen.de/research/projects/hypro/benchmarks-of-continuous-and-hybrid-systems/`. [Online; accessed 20-February-2018].

[TNJ16] Hoang-Dung Tran, Luan Viet Nguyen, and Taylor T Johnson. Large-scale linear systems from order-reduction (benchmark proposal). In *ARCH16*, 2016.

Appendix A

Examples Displayed in Different Input Languages

Example 3.1a "Heater"

Listing A.1: Flow* and HyDRA syntax

```
hybrid reachability
{
state var x, t

setting {
  fixed steps 0.01
  time 20
  remainder estimation 1e-4
  QR precondition
  gnuplot octagon t,x
  fixed orders 10
  cutoff 1e-15
  precision 53
  output heaterFlowStar
  max jumps 1000
  print on
}

modes {
  on {
    lti ode {x' = 50-x t'=1}
    inv {x<=23 t<=20}
  }
  off {
    lti ode {x' = -x t'=1}
    inv {x>=17 t<=20}
  }
}

jumps {
  on -> off
  guard {x >= 22}
  reset {}
  parallelotope aggregation {}
```

```
  off -> on
  guard {x <= 18}
  reset {}
  parallelotope aggregation {}
}

init {
  on {x in [20, 21]}
  }
}
unsafe {
  on{x>=30}
  off{x>=30}
}
```

Listing A.2: HyLAA syntax

```
def define_ha():
  ha = LinearHybridAutomaton()
  ha.variables = ["x", "t"]

  on_loc1 = ha.new_mode('on_loc1')
  a_matrix = np.array([ \
    [-1, 0], \
    [0, 0], \
  ], dtype=float)
  c_vector = np.array([50, 1], dtype=float)
  on_loc1.set_dynamics(a_matrix, c_vector)
  on_loc1.inv_list.append(LinearConstraint([1, 0], 23))
  on_loc1.inv_list.append(LinearConstraint([0, 1], 20))

  off_loc1 = ha.new_mode('off_loc1')
  a_matrix = np.array([ \
    [-1, 0], \
    [0, 0], \
  ], dtype=float)
  c_vector = np.array([0, 1], dtype=float)
  off_loc1.set_dynamics(a_matrix, c_vector)
  off_loc1.inv_list.append(LinearConstraint([-1, -0], -17))
  off_loc1.inv_list.append(LinearConstraint([0, 1], 20))

  _error = ha.new_mode('_error')
  _error.is_error = True

  trans = ha.new_transition(on_loc1, off_loc1)
  trans.condition_list.append(LinearConstraint([-1, -0], -22))

  trans = ha.new_transition(off_loc1, on_loc1)
  trans.condition_list.append(LinearConstraint([1, 0], 18))

  trans = ha.new_transition(on_loc1, _error)
  trans.condition_list.append(LinearConstraint([-1, -0], -30))

  trans = ha.new_transition(off_loc1, _error)
  trans.condition_list.append(LinearConstraint([-1, -0], -30))

  return ha

def define_init_states(ha):
  rv = []

  constraints = []
```

```
    constraints.append(LinearConstraint([-1, 0], -20))
    constraints.append(LinearConstraint([1, 0], 21))
    constraints.append(LinearConstraint([0, 1], 0))
    constraints.append(LinearConstraint([-0, -1], -0))
    rv.append((ha.modes['on_loc1'], constraints))

    return rv


def define_settings():
    plot_settings = PlotSettings()
    plot_settings.plot_mode = PlotSettings.PLOT_NONE
    plot_settings.xdim = 1
    plot_settings.ydim = 0

    settings = HylaaSettings(step=0.01, max_time=20.0,
          plot_settings=plot_settings)

    return settings

def run_hylaa(settings):
    ha = define_ha()
    init = define_init_states(ha)

    engine = HylaaEngine(ha, settings)
    engine.run(init)

    return engine.result

if __name__ == '__main__':
    run_hylaa(define_settings())
```

Listing A.3: SpaceEx syntax, model file

```
<?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="clock">
  <param name="t" type="real" local="false" d1="1"
      d2="1" dynamics="any" />
  <param name="T" type="real" local="false" d1="1"
      d2="1" dynamics="const" />
  <location id="1" name="loc1">
    <invariant>t&lt;=T</invariant>
  <flow>t'==1</flow>
  </location>
</component>
<component id="heater">
  <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="T" type="real" local="false" d1="1"
      d2="1" dynamics="const" controlled="true" />
    <location id="1" name="on">
    <invariant> x &lt;= 23 &amp; t&lt;=T </invariant>
    <flow> x' == 50-x &amp; t'==1</flow>
  </location>
  <location id="2" name="off">
    <invariant> x &gt;= 17 &amp; t&lt;=T </invariant>
    <flow> x' == -x &amp; t'==1</flow>
  </location>
```

```
  <transition source="1" target="2">
    <guard> x &gt;= 22 </guard>
  </transition>
  <transition source="2" target="1">
    <guard> x &lt;= 18 </guard>
  </transition>
</component>
<component id="system">
  <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="T" type="real" local="false" d1="1"
      d2="1" dynamics="const" controlled="true" />
  <bind component="heater" as="heater">
    <map key="x">x</map>
    <map key="t">t</map>
    <map key="T">T</map>
    </bind>
  </component>
</sspaceex>
```

Listing A.4: SpaceEx syntax, configuration file

```
system = system
initially = "20<=x & x<=21 & loc(heater)==on & t==0 & T==20"
forbidden = "x>=30"
scenario = stc
directions = oct
set-aggregation = chull
sampling-time = 0.01
time-horizon = 10
iter-max = 1000
output-variables = "t, x"
output-format = GEN
rel-err = 1.0E-12
abs-err = 1.0E-13
flowpipe-tolerance = 0.001
```

Listing A.5: CIF3 syntax

```
automaton heater:
  cont x;
  cont t = 0;
  initial x >= 20;
  initial x <= 21;

  location on:
    initial;
    equation x' = 50-x, t' = 1;
    invariant x <= 23, t <= 20;
    edge when x>=22 goto off;

  location off:
    equation x' = -x;
    equation t' = 1;
    invariant x >= 17;
    invariant t <= 20;
    edge when x<=18 goto on;
end
```

Time-Variant Uncertain Example

An example for a simple affine dynamic flow condition with time-variant uncertainty. The represented ODE is $x' = 2 * x + 2 * u + 3$ with $u \in [-1,1]$.

Listing A.6: Flow* syntax

```
hybrid reachability
{
state var x

setting
{
  fixed steps 0.01
  time 10
  remainder estimation 1e-4
  identity precondition
  fixed orders 10
  cutoff 1e-15
  precision 53
  output out
  max jumps 1000
  print off
}

modes {
  location {
    lti ode {x' = 2.0 * x + [-2.0, 2.0] + 3.0}
    inv {x <= 23.0}
  }
}

jumps { }

init {
  location {x in [0, 0]}
  }
}
```

Listing A.7: HyLAA syntax

```
def define_ha():
  ha = LinearHybridAutomaton()
  ha.variables = ["x"]

  location = ha.new_mode('location')
  a_matrix = np.array([ \
    [2], \
  ], dtype=float)
  c_vector = np.array([3], dtype=float)
  location.set_dynamics(a_matrix, c_vector)

  u_constraints_a = np.array([[1], [-1]], dtype=float)
  u_constraints_b = np.array([1, 1], dtype=float)
  b_matrix = np.array([[2]], dtype=float)
  location.set_inputs(u_constraints_a, u_constraints_b, b_matrix)
  location.inv_list.append(LinearConstraint([1], 23))

  return ha
```

```
def define_init_states(ha):
  rv = []

  constraints = []
  constraints.append(LinearConstraint([1], 0))
  constraints.append(LinearConstraint([-1], -0))
  rv.append((ha.modes['location'], constraints))

  return rv

def define_settings():
  plot_settings = PlotSettings()
  plot_settings.plot_mode = PlotSettings.PLOT_NONE

  settings = HylaaSettings(step=0.01, max_time=10.0,
        plot_settings=plot_settings)

  return settings

def run_hylaa(settings):
  ha = define_ha()
  init = define_init_states(ha)

  engine = HylaaEngine(ha, settings)
  engine.run(init)

  return engine.result

if __name__ == '__main__':
  run_hylaa(define_settings())
```

Listing A.8: SpaceEx syntax, model file

```
<?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="uncertain">
  <param name="x" type="real" d1="1" d2="1"
      local="false" dynamics="any" controlled="true"/>
  <param name="u" type="real" d1="1" d2="1"
      local="false" dynamics="any" controlled="true"/>
  <location id="1" name="location">
    <invariant> x &lt;= 23 &amp; u&lt;=1 &amp; u&gt;=-1 </invariant>
    <flow> x' == 2*x + 2*u + 3</flow>
  </location>
</component>
<component id="system">
  <param name="x" type="real" local="false" d1="1"
      d2="1" dynamics="any" controlled="true" />
  <param name="u" type="real" local="false" d1="1"
      d2="1" dynamics="any" controlled="true" />
  <bind component="uncertain" as="uncertain">
    <map key="x">x</map>
    <map key="u">u</map>
  </bind>
</component>
</sspaceex>
```

Listing A.9: SpaceEx syntax, configuration file

```
system = system
initially = "x==0 & loc(uncertain)==location"
forbidden = ""
scenario = stc
directions = oct
set-aggregation = chull
sampling-time = 0.01
time-horizon = 10
iter-max = 1000
rel-err = 1.0E-12
abs-err = 1.0E-13
flowpipe-tolerance = 0.001
```

Listing A.10: CIF3 syntax

```
automaton uncertain:
  cont x =0;
  cont u;
  invariant -1<=u, u<=1;

  location loc:
    initial;
    equation x'=2*x + 2*u +3;
    invariant x<=23
end
```

Appendix B

Settings for Different Tools

`fixed steps`	time steps
`adaptive steps`	adaptive time steps in a range, e.g. `{ min 0.1 , max 0.2}`
`time`	time bound
`remainder estimation`	remainder estimation in each integration step and in each dimension, use a single value for all dimension, e.g. `1e-5` or different for every dimension, e.g. `{x :[0.1 ,0.101], y :[ -0.01 ,0.06]}`
`* precondition`	preconditioning method, use `identity` or `QR`. For more imformation see [CÁS13].
`gnuplot`	plot options. Use `octagon`, `matlab` or `grid x` with x the number of grids paving. Followed by the variables to be plotted.
`fixed orders`	Taylor Models order, higher values indicates higher accuracy
`adaptive orders`	range of the Taylor Models order, e.g. `{min 3, max 8}`, higher values indicates higher accuracy
`cutoff`	the cutoff threshold, e.g. `1e-15`
`precision`	the precision used by MPFR library, higher values indicates higher accuracy
`output`	name of the output file
`max jumps`	bound for jumps depth of the computation
`print`	enable (`on`) or disable (`off`) printing

Figure B.1: Selection of Flow* setting

`fixed steps`	time steps
`time`	time bound
`identity precondition`	preconditioning method, see [CÁS13].
`gnuplot octagon`	plot option followed by the variables to be plotted.
`output`	name of the output file
`max jumps`	bound for jumps depth of the computation
`print`	enable (`on`) or disable (`off`) printing

Figure B.2: Selection of HyDRA settings (inside file)

`clustering`	clustering method, see 6.2
`representation`	state set representation initially used. Use `box`, `support_function`, `zonotope`, `polytope_h`, `polytope_v`
`threads`	number of threads to be used as as worker threads. Default is 1
`plotoutput`	path to the output file
`plotoutputformat`	plot output format. Use `tex` or `gnuplot`

Figure B.3: Selection of HyDRA settings (command line)

`step`	step size
`max_time`	time bound
`_settings`	name of the plot settings object, see Fig. B.5
`process_urgent_guards`	enabled (`True`) urgent transitions
`counter_example_filename`	if we want to print counterexamples, e.g. `"counter_examples.py"`
`simulation.threads`	auto-detect number of system cores (default) or specify a number
`simulation.sim_mode`	use `SIMULATION` for full time range simulation (default) or `MATRIX_EXP` for matrix exponential(expm) for first step then do matrix multiplication

Figure B.4: Selection of HyLAA settings

`plot_mode`	general plot mode, use `PlotSettings.PLOT_NONE`, `PlotSettings.PLOT_IMAGE` or `PlotSettings.PLOT_MATLAB`
`xdim`	variable for x-dimension
`ydim`	variable for y-dimension
`label.x_label`	label for x-dimension
`label.y_label`	label for y-dimension
`label.title`	titel for the plot
`label.axes_limits`	axes limites of the form `(x_min, x_max, y_min, y_max)`
`make_video`	computes a video of the computed states, e.g. `("building.mp4", frames=220, fps=40)`

Figure B.5: Selection of HyLAA plot settings

`system`	name of the component to be analyzed
`initially`	initial set, see Fig. 4.2
`forbidden`	unsafe set, see Fig. 4.3
`scenario`	Le Guernic-Girard (`supp`) or STC scenario (`stc`)
`directions`	polyhedra used for intersections, use `box`, `oct` or constraints e.g. `x>=25`
`set-aggregation`	Aggregation method, use `chull` or `none`
`clustering`	clustering method, see 6.2
`sampling-time`	time step size
`time-horizon`	only local
`iter-max`	maximal number of iterations, negative values results in computation until a fixpoint is found
`output-variables`	variables to be written into the output file
`output-format`	use textual `TXT`, vertice list `GEN` or JavaView JVX format `JVX`
`rel-err`	used when comparing floating point values and deciding whether they are considered equal. This impacts mainly tests for containment and emptiness of objects. Smaller values inidcates highter accuracy
`abs-err`	see `rel-err`

Figure B.6: Selection of SpaceEx settings

Appendix C

Settings Used in the Benchmarks

Minimal Benchmark Set

For all benchmarks the tools ran with the following settings. The values for Flow*, HyDRA and HyLAA are derived from transformation with HyST. Time bound can be found in the benchmark descriptions and time step size is 0.01.
Flow*: fixed order 5, remainder estimation 1e-4, identity precondition, cutoff 1e-15, precision 53, max jumps 999
HyDRA: max jumps 999, r = support_function, t=4
HyLAA: standard setting
SpaceEx: directions = box, set-aggregation = chull, iter-max = 1000, rel-err = 1.0E-12, abs-err = 1.0E-13, flowpipe-tolerance = 0.001

Well Known Benchmarks Set

For all benchmarks the tools ran with the following settings. The values for Flow*, HyDRA and HyLAA are derived from transformation with HyST. For the used time bounds, time step sizes, initial and unsafe sets, number of jumps or iterations, see Table C.1.
Flow*: fixed order 5, remainder estimation 1e-5, identity precondition, cutoff 1e-15, precision 128
HyDRA: r = support_function, t=4
HyLAA: standard setting
SpaceEx: directions = oct, set-aggregation = none, iter-max = 1000, rel-err = 1.0E-12, abs-err = 1.0E-13, flowpipe-tolerance = 0.05

	T	t	init	unsafe	j	i
bouncing ball	10	0.1	$10 \leq x \leq 10.2$	$x \geq 10.7$	20	20
two tank system	5	0.01	$x \in [1.5,2.5]$, $x_2 = 1$	$x_2 \leq -0.7$	50	50
rod reactor	50	0.1	$x \in [510,520]$	shutdown	20	50
5-D lin. switch	1	0.1	$x_1 = 3.1$, $x_2 = 4$, $x_3 = x_4 = x_5 = 0$	$x_1 \leq -1.5$	5	5
filt. oscillator 4, 8, 16, 32	4	0.05	$x \in [0.2,0.3]$, $y \in [-0.1,0.1]$, $a = 0$ for all $a \in Var \backslash \{x,y\}$	$y \geq 0.5$	20	20
5 / 10 vehicle platoon	20	0.01	$x \in [0.9,1.1]$ for all $x \in Var$	$x_1 \in [-0.5,0.2]$	1	-1

Table C.1: Settings for the well known benchmark set. Legend: T: time horizon, t: time step size, init: initial variable valuation, unsafe: unsafe set, j: jump bound for Flow* and HyDRA, i: value of iter-max for SpaceEx.

ARCH17 benchmark set

Building benchmark

Initial and unsafe sets are taken from [TNJ16]. Further settings are:
Flow*: fixed steps 0.008, remainder estimation 1e-4, QR precondition, fixed orders 25, cutoff 1e-15, precision 100, max jumps 100
HyDRA: fixed steps = 0.04, r = support_function, t = 4, max jumps 100
HyLAA: step = 0.04
SpaceEx: directions = box, sampling-time = 0.004, rel-err = 1.0e-9, abs-err = 1.0e-12, flowpipe-tolerance=1e-3

Platoon benchmark

Initial and unsafe sets are taken from [MK14]. Further settings are:
Flow*: fixed steps 0.01, remainder estimation 1e-4, identity precondition, fixed orders 3, cutoff 1e-12, precision 100, max jumps 999
SpaceEx: directions = box, set-aggregation = chull, flowpipe-tolerance = 1, flowpipe-tolerance-rel = 0, simu-init-sampling-points = 0, iter-max = 200, verbosity = m, output-error = 0, rel-err = 1.0E-9, abs-err = 1.0E-12

Gear benchmark

The initial set is $0 \times 0 \times [0.0168, 0.0166] \times [0.0029, 0.0031] \times 0$ and the unsafe sets are $free \wedge t \geq 0.2$ and $x_5 \geq 20$. Further settings are:
Flow*: fixed steps 0.001, time 0.2, remainder estimation 1e-4, identity precondition, fixed orders 3, cutoff 1e-15, precision 100, max jumps 4
SpaceEx: directions = oct, set-aggregation = none, sampling-time = 1, flowpipe-tolerance = 0.001, flowpipe-tolerance-rel = 0, time-horizon = 0.2, iter-max = 200,verbosity = m, output-error = 0, rel-err = 1.0E-12, abs-err = 1.0E-15