Verified Reachability Analysis of Continuous Systems

Fabian Immler*

Institut für Informatik, Technische Universität München
immler@in.tum.de

Abstract. Ordinary differential equations (ODEs) are often used to model the dynamics of (often safety-critical) continuous systems.

This work presents the formal verification of an algorithm for reachability analysis in continuous systems. The algorithm features adaptive Runge-Kutta methods and rigorous numerics based on affine arithmetic. It is proved to be sound with respect to the existing formalization of ODEs in Isabelle/HOL. Optimizations like splitting, intersecting and collecting reachable sets are necessary to analyze chaotic systems. Experiments demonstrate the practical usability of our developments.

Keywords: Numerical Analysis, Rigorous Numerics, Validated Numerics, Ordinary Differential Equation, Continuous System, Interactive Theorem Proving.

1 Introduction

Many real-world systems with continuous dynamics can be modeled with ordinary differential equations (ODEs). An important task is to determine for a set of initial states all reachable states. This requires to compute enclosures for solutions of ODEs, which is done by tools for guaranteed integration (e.g., VNODE-LP [21] or COSY [5]) and also by tools for reachability analysis of hybrid systems (with the state-of-the-art tool for linear dynamics SpaceEx [13] and tools supporting non-linear dynamics like Flow* [9], HySAT/iSAT [12], or Ariadne [4]). Such tools aim at computing safe overapproximations, an intended use is often the analysis of safety-critical systems. Therefore any effort to improve the level of rigor is valuable, and such efforts have been undertaken already: Nedialkov [21] implemented VNODE-LP using literate programming such that correctness of the code can be examined by human experts. Taylor models, which are used to represent reachable sets in COSY, Flow*, and Ariadne, have been formalized in theorem provers in the context of Ariadne [10] but also as a generic means for validated numerics [8,25].

Here we present the formal verification of an algorithm for reachability analysis of continuous systems. The algorithm splits, reduces and collects reachable

* Supported by the DFG RTG 1480 (PUMA).

C. Baier and C. Tinelli (Eds.): TACAS 2015, LNCS 9035, pp. 37–51, 2015.
DOI: 10.1007/978-3-662-46681-0_3

sets during the analysis, crucial features for being able to analyze chaotic systems. Propagation of reachable sets is implemented using higher-order Runge-Kutta methods with adaptive step size control. The formal verification of all those algorithms is a novel contribution and a qualitative improvement on the level of trust that can be put into reachability analysis of continuous systems. Experiments show that our algorithms allow to analyze low-dimensional, nonlinear systems that are out of reach for many of the existing tools. Nevertheless, our work should not be considered a rival to the existing tools or concepts, which are more mature and flexible. We would rather like to demonstrate that formal verification does not exclude competitive performance.

We build on our formalization of affine arithmetic and the Euler method [15]. The verification is carried out with respect to the theory of ODEs in the interactive theorem prover Isabelle/HOL [22]. Every definition and theorem we display in this document possesses a formally proved and mechanically checked counterpart. The development is available in the Archive of Formal Proofs [18].

1.1 Related Work: ODEs and ITPs

In addition to the previously mentioned work on analysis of continuous systems, there also exists related work on differential equations formalized in theorem provers: Spitters and Makarov [20] implement Picard iteration to calculate solutions of ODEs in the interactive theorem prover Coq, but restricted to relatively short existence intervals. Boldo *et al.* [6] approximate the solution of one particular partial differential equation with a C-program and verify its correctness in Coq. Platzer [23] uses a different approach in that he does not do numerical analysis but uses differential invariants to reason symbolically about dynamical systems in a proof assistant.

2 Main Ideas

In what follows, we consider the problem of computing reachable sets for systems defined by an autonomous ODE $\dot{x} = f(x)$ with $f : \mathbb{R}^n \to \mathbb{R}^n$. We denote the solution depending on initial condition x_0 and time t with $\varphi(x_0, t)$. Reachability analysis aims at computing (or overapproximating) all states of the system that are reachable from some set of initial states $X_0 \subseteq \mathbb{R}^n$ within a time horizon $T \subseteq \mathbb{R}$, i.e., the set $\varphi(X_0, T)$.

We will start by illustrating the main ingredients of our algorithm for reachability analysis. We do not claim originality for those ideas, however combining all of them for numerically solving ODEs and especially formally verifying them is, to the best of our knowledge, a novel contribution.

Rigorous Numerics. First of all, in any numerical computation, continuous, real-valued quantities are approximated with finite precision. One therefore needs to cope with round-off errors. Reasoning about them explicitly gets very tedious. We therefore take the approach of set-based computing, or *rigorous numerics*:

The idea is to compute with sets instead of single values and abstract all kinds of errors (including round-off) by including them into the set. The data structure we choose is *affine forms*, they represent sets called *zonotopes* and have been successfully applied in hybrid systems analysis [14].

Guaranteed Runge-Kutta Methods with Step Size Adaptation. Bouissou *et al.* [7] presented the idea to turn "classical" numerical Algorithms into guaranteed methods by using affine arithmetic. They illustrated their approach on a *stiff* (i.e., numerical approximations requiring very small step sizes in parts of the state space) ODE, which makes adaptive step size control necessary. In general, automatic step size adaptation improves the performance of any numerical method, as it avoids wasting computational time on "easy" parts of the solution and maintains high accuracy on "hard" parts of the solution.

Splitting. Zonotopes are convex sets, this leads of course to loss of precision when non-convex sets need to be enclosed. But non-linear dynamics produce non-convex sets, which is why a purely zonotope based approach is likely to fail because of more and more increasing overapproximations. The immediate approach is to split the sets before they grow too large, and have the union of smaller sets represent the larger non-convex set.

Reduction. While splitting sets allows to maintain precision in the presence of non-convex sets, it leads to problems when the dynamics produce large sets. Especially when analyzing chaotic systems, small initial sets expand rapidly – due to the dynamics of the system, not necessarily because of inaccurate computations. This may produce a prohibitively large number of split sets. Any possibility to reduce the size of reachable sets therefore is a valuable improvement because it helps to reduce the number of sets. Our method is based on the idea that whenever a reachable set flows through a hyperplane, it can be reduced to the intersection with that hyperplane. We got the idea of reducing to transversal hyperplanes from Tucker's [24] algorithm, which reduces reachable sets after every step to axis-perpendicular hyperplanes. Bak [3] also proposed to perform reductions transversal to the flow. But in his setting, the user needs to come up with suitable reductions.

3 Verification

We formalize all of the previous "main ideas" using the interactive theorem prover Isabelle/HOL [22]. We build on Isabelle/HOL's library for multivariate analysis and the formalization of ODEs [17]. Our algorithms are formalized as monadic programs using Lammich's [19] framework. In such programs, we write $x \leftarrow y$ to bind x to the result of y, which may also fail. We write $x \in X$ to choose an arbitrary element x from the set X.

3.1 Reachability in Continuous Systems

In order to verify our algorithms, we need of course a specification. We assume a continuous system where the evolution is governed by a continuous *flow* $\varphi(x,t)$, i.e., $\varphi(\varphi(x_0,t),s) = \varphi(x_0,t+s)$. We formalize reachability with the ternary predicate $\curvearrowright$, where $X \curvearrowright_{C_X} Y$ holds if the evolution flows every point of $X \subseteq \mathbb{R}^n$ to $Y \subseteq \mathbb{R}^n$ and does not leave the set C_X in the meantime.

$$X \curvearrowright_{C_X} Y := \forall x \in X.\ \exists t \geq 0.\ \varphi(t,x_0) \in Y \wedge (\forall 0 \leq s \leq t.\ \varphi(s,x_0) \in C_X)$$

C_X can therefore be used to describe safety properties during the reachability analysis. This predicate allows to easily combine steps in reachability analysis according to the rule $X \curvearrowright_{C_X} Y \wedge Y \curvearrowright_{C_Y} Z \implies X \curvearrowright_{(C_X \cup C_Y)} Z$.

3.2 Rigorous Numerics: Affine Arithmetic

Rigorous (or *guaranteed*) numerics means computing with sets that guarantee to enclose the quantities of interest. The most basic data structure to represent sets is intervals, but those suffer from the wrapping effect – enclosing rotated boxes with boxes leads to large overapproximations. Moreover dependencies between variables are lost, e.g. for an enclosure $x \in [-1;1]$, the term $x - x$ evaluates to $[-2;2]$ in interval arithmetic.

Affine arithmetic [11] improves over interval arithmetic by tracking linear dependencies. For this one utilizes affine forms, represented by a list of *generators* $\langle a_0, \ldots, a_k \rangle$ with $a_i \in \mathbb{R}^n$. An affine form is the formal expression $a_0 + \sum_{0<i\leq k} \varepsilon_i \cdot a_i$ where the formal variables ε_i are called *noise symbols*. The set $\gamma\langle a_0, \ldots, a_k \rangle$ represented by an affine form is called a *zonotope* and is given as the set of all elements when the formal variables ε_i range over $[-1;1]$: $\gamma\langle a_0, \ldots, a_k \rangle = \{a_0 + \sum_{0<i\leq k} \varepsilon_i \cdot a_i \mid -1 \leq \varepsilon_i \leq 1\}$

Affine forms track linear dependencies, because the formal variables are treated symbolically. Examining the dependency problem from before, if we have the affine form $1 \cdot \varepsilon_1$ representing the enclosure $x \in \gamma(1 \cdot \varepsilon_1) = [-1;1]$, then evaluating $x - x$ in affine arithmetic yields $1 \cdot \varepsilon_1 - 1 \cdot \varepsilon_1 = 0 \cdot \varepsilon_1$. The result represents therefore the exact quantity $\{0\}$. Any linear operation $A : \mathbb{R}^n \to \mathbb{R}^n$ can be represented exactly, as it distributes over the generators of the affine form: $A(\gamma(\langle a_0, \ldots, a_k \rangle)) = \gamma\langle Aa_0, \ldots, Aa_k \rangle$. Nonlinear operations like multiplication or division are linearized, adding the linearization error as a new noise symbol. Provided with safe estimations on round-off errors, those can be included in computations with affine forms as well. In general, all kinds of uncertainties can be added using Minkowski addition $X \oplus Y = \{x + y \mid x \in X \wedge y \in Y\}$, which can be implemented efficiently for affine forms by taking a disjoint union of the generators.

3.3 Guaranteed Runge-Kutta Methods

Having presented the background on rigorous numerics, we will now concentrate on solving ODEs numerically. A classical approach is given by Runge-Kutta

methods, which approximate the solution in a series of discrete steps in time. We assume from now on an autonomous ODE $\dot{x} = f(x)$ and $f \in \mathcal{C}^2(\mathbb{R}^n, \mathbb{R}^n)$ twice continuously differentiable. Recall that we denote the solution for initial value x_0 at time t with $\varphi(x_0, t)$. Runge-Kutta methods are one-step methods: they discretize the time into a *grid* of times $t_0, \ldots, t_i, \ldots$ with step size $h_i = t_{i+1} - t_i$ and compute a series of steps $x_i \approx \varphi(x_0, t_i)$. The *discretization error* $|\varphi(x_i, h_i) - x_i|$ is obtained via Taylor series expansions of the solution and the Runge-Kutta method.

Runge-Kutta methods can be turned into guaranteed methods by evaluating the approximate steps using rigorous numerics, e.g., in affine arithmetic. To be guaranteed, it is necessary to explicitly include the discretization error in the set representation. In order to obtain a safe estimate for the discretization error, one first needs to prove that the solution exists for the desired step and find an a-priori bound on the solution.

A unique solution for an initial value x_0 exists for stepsize h if the iteration given by the Picard operator $P_h : \mathcal{C}^\infty([0;h], \mathbb{R}^n) \to \mathcal{C}^\infty([0;h], \mathbb{R}^n)$ with $P_h(\varphi) = (t \mapsto x_n + \int_0^t f(\varphi(s))\mathsf{d}s)$ has a unique fixed point, which can be reduced to finding a post fixed point for an overapproximating operator $Q_h : \mathcal{P}(\mathbb{R}^n) \to \mathcal{P}(\mathbb{R}^n)$ with $Q_h(X) = X_n + [0;h] \cdot f(X)$.

cert-stepsize is defined to choose a step size h and iterate Q_h until a post fixed point C is reached, i.e., $Q_h(C) \subseteq C$. If that does not succeed, the iteration restarts with a smaller step size. *cert-stepsize* returns the chosen step size and the post fixed point, which certifies the existence of a unique solution for the chosen step size. The post fixed point also gives an *a-priori* bound on the solution:

Theorem 1 (Certification of Step). *If $x_0 \in X_0$ and cert-stepsize$(X_0) = (h, C)$, then there exists a unique solution $\varphi(x_0, [0;h]) \subseteq C$.*

The most basic Runge-Kutta method is the method of Euler, it approximates the solution $\varphi(x_0, h)$ with the linear function with the slope given by the ODE f at instant t: $\varphi(x_0, h) \approx x_0 + h \cdot f(x_0)$. The right-hand side of this approximation is exactly the first two terms of a Taylor series expansion of the solution φ. When evaluating f at different points, one can achieve that the Taylor series expansions match up to higher order, which is the idea of Runge-Kutta methods.

We verified a generic two-stage Runge-Kutta method $\mathit{rk2}_h(x) = x + h \cdot \psi_h(x)$, with $\psi_h(x) = (1 - \frac{1}{2p})f(x) + \frac{1}{2p}f(x + hpf(x))$. Then $\mathit{rk2}_h(x_0)$ approximates the solution: $|\varphi(x_0, h) - \mathit{rk2}_h(x_0)| \in \mathcal{O}(h^3)$. We assume $0 < p \leq 1$ for the parameter p, one can choose e.g., $p = 1$, to obtain the classical method of Heun.

For non-guaranteed methods, it suffices to show via Taylor series expansions of φ and $\mathit{rk2}_h$ that the solution and Runge-Kutta approximation differ by some remainder term in $\mathcal{O}(h^3)$. For a guaranteed method, an explicit estimate for the remainder term is needed, which requires higher derivatives of f. We denote by $f'(x) : \mathbb{R}^n \to \mathbb{R}^n$ the derivative (the linear mapping given by the Jacobian matrix) of f at x and with $f''(x) : \mathbb{R}^n \to \mathbb{R}^n \to \mathbb{R}^n$ the derivative of f' (a bilinear mapping).

Algorithm 1. Step of Runge-Kutta method

1: **function** *rkstep*(X_0)
2: $(h, C) \leftarrow$ *cert-stepsize*(X_0)
3: $R \leftarrow$ *rk2-remainder*$_h(X_0, C)$
4: $C' \leftarrow$ *rk2-remainder*$_{[0;h]}(X_0, C)$
5: $X_1 \leftarrow$ *rk2*$_h(X_0) \oplus R$
6: $X_C \leftarrow$ *rk2*$_{[0;h]}(X_0) \oplus C'$
7: $\varepsilon \leftarrow$ *width*(R)
8: **return** $(h, \varepsilon, X_1, X_C)$

When we set $I = [0; 1]$ and $T = [0; h]$ as enclosures for the occurring mean values, the following expression for the remainder term can be deduced and proved correct:

$$\textit{rk2-remainder}_h(X, X_C) := \frac{h^3}{6} f''\big(X_C\big)\big(f(X_C)\big)\big(f(X_C)\big) + \\ + \frac{h^3}{6} f'\big(X_C\big)\big(f'(X_C)(f(X_C)\big) - \frac{h^3 p}{4} f''\big(X + hpIf(X)\big)\big(f(X)\big)\big(f(X)\big)$$

Theorem 2 (Remainder of Two-Stage Runge-Kutta). *If* $\varphi(x_0, t) \in X$ *and* $\varphi(x_0, T) \subseteq X_C$*, then* $\varphi(x_0, h) - \textit{rk2}_h(X) \in \textit{rk2-remainder}_h(X, X_C)$*.*

With Algorithm 1, *rkstep*, we compute one step of the guaranteed Runge-Kutta method: C is a first, rough enclosure for the solution over the interval $[0; h]$, which is used to compute a tighter enclosure X_C over the interval and an even tighter one X_1 at the time instant h. The algorithm then satisfies the following specification, which follows from Theorems 1 and 2.

Theorem 3 (Step of Runge-Kutta Method). *Assume* $x_0 \in X_0$ *and* $\textit{rkstep}(X_0) = (h, \varepsilon, X_1, X_C)$*. Then there exists a unique solution* $\varphi(x_0, [0; h]) \subseteq X_C$ *with* $\varphi(x_0, h) \in X_1$*, or in terms of the reachability predicate* $X_0 \curvearrowright_{X_C} X_1$*.*

Note that the computation (in particular for *rk2-remainder*) requires the higher derivatives f', f'' of f, which Isabelle/HOL can automatically derive from the symbolic representation of f. The quantity ε did not occur in the specification. It gives the size of the remainder term, the discretization error. We can therefore use ε to guide step size control in section 3.7.

3.4 Splitting

In the previous section we had developed the analysis of the discretization error, which is unfortunately not the only source of error. Errors are introduced due to linearization of operations on affine forms: non-convex sets are enclosed in the convex zonotopes. These errors are quadratic in the size of the zonotope, acceptable precision can therefore be maintained if the size of the zonotopes is kept small. Zonotopes generated by $\langle a_0, \ldots, a_n \rangle$ can be split

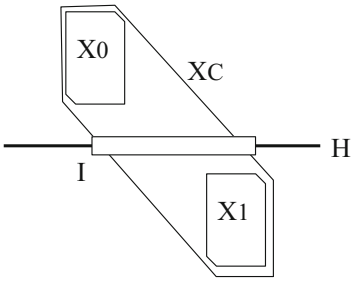

Fig. 1. Idealized reduction

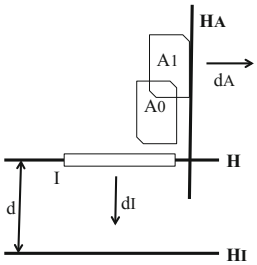

Fig. 2. Selection of hyperplanes

by halving one of the generators a_i, i.e., setting $\mathit{split}\ (\langle a_0, \ldots, a_n\rangle, i) = (\langle a_0 - a_i/2, a_1, \ldots, a_{i-1}, a_i/2, a_{i+1}, \ldots, a_n\rangle, \langle a_0 + a_i/2, a_1, \ldots, a_{i-1}, a_i/2, a_{i+1}, \ldots, a_n\rangle)$ The range of the resulting zonotopes encloses the range of the argument, which follows from the definition of γ.

Theorem 4 (Splitting). $\mathit{split}(X) = (Y, Z) \implies \gamma(X) \subseteq \gamma(Y) \cup \gamma(Z)$.

3.5 Reduction of Reachable Sets

Too many splits impair performance, which is why the size of the reachable sets must be reduced whenever possible. The idea is to reduce reachable sets by looking at how the flow passes through a given hyperplane H.

The general idea is to start with a reachable set X_0 above the hyperplane and perform one Runge-Kutta step $\mathit{rkstep}(X_0) = (h, \varepsilon, X_1, X_C)$ towards a reachable set X_1 below the hyperplane, see Figure 1. The enclosure for the flow between X_0 and X_1 is given by X_C, which means that every flow that eventually reaches X_1 has to pass through the intersection $I := X_C \cap H$. Therefore the computation of reachable sets can continue with I instead of X_1, which is of advantage if I is smaller than X_1.

However, the situation is in general a bit more complicated because X_1 cannot be guaranteed to lie below H, or only with very large step sizes. Also the dynamics might just "scratch" the hyperplane, i.e., not completely passing through it. To cope with those difficulties, Algorithm 2 is used to compute the intersection of the flow from reachable set X_0 with the hyperplane H: it iterates Runge-Kutta steps until the set has passed through H. It also allows to *abort* the iteration if e.g., the flow has changed its dominating direction during the iteration.

The relation between the reachable set and the computed intersection can be expressed with the reachability predicate $X \curvearrowright_{C_X} Y$. In addition, we write $H^{\geq}$ for the half-plane above H. This allows to specify the outcome of *intersect-flow*: Every flow starting from X above the half-plane reaches the intersection.

Theorem 5 (Intersection of Flow from X with Hyperplane H). $\mathit{intersect\text{-}flow}(X, H) = (A, \mathcal{X}, \mathcal{I}) \implies \left(X \cap H^{\geq}\right) \curvearrowright_{\mathcal{X}} \left(A \cup \bigcup_{I \in \mathcal{I}} I\right)$

Algorithm 2. Intersection of Flow from X with Hyperplane H

1: **function** *intersect-flow*(X, H)
2: $\quad\mathcal{I}, \mathcal{X} = \emptyset$
3: $\quad$**while** $\neg(X \text{ below } H) \wedge \neg$*abort*$(X)$ **do**
4: $\quad\quad(h, \varepsilon, X_1, X_C) \leftarrow$ *rkstep*(X)
5: $\quad\quad\mathcal{I} \leftarrow \mathcal{I} \cup \{X_C \cap H\}$ ▷ intersection of zonotope with hyperplane
6: $\quad\quad\mathcal{X} \leftarrow \mathcal{X} \cup \{X_C\}$
7: $\quad\quad X \leftarrow X_1$
8: $\quad$**return** $(X, \mathcal{X}, \mathcal{I})$

The crucial step of Algorithm 2 is the computation of the intersection of the zonotope X_C with the hyperplane H in line 5, which can only be done approximately. The verification of this is a nontrivial task [16].

3.6 Summarization of Intersections

When the intersection is computed by flowing the reachable set through the hyperplane step by step, we get a set $\mathcal{I}$ consisting of individual intersections I_i. Many of the sets I_i usually overlap, in order to avoid redundant enclosures, it is desirable to remove the overlaps. Ideally, this could be done using set difference, an operation under which zonotopes are not closed. Therefore an overapproximation has to suffice. The overapproximation lays a grid of (hyper-)rectangles $R_k = [r_k^-; r_k^+]$ over the interval enclosure $[I^-; I^+]$ of $\bigcup_{I\in\mathcal{I}} I$: $[I^-; I^+] = \bigcup_k R_k$. Then we shrink every element R_k to R'_k such that the union still encloses I: $R'_k = R_k \cap [r'^-_k; r'^+_k]$ where $[r'^-_k; r'^+_k]$ is the interval enclosure of $\bigcup_{i.\ I_i\cap R_k\neq\emptyset} I_i$, i.e. the union of all I_i that overlap with R_k. This process might even remove some of the sets R'_k.

The only important proposition to prove is that the so computed collection is a safe overapproximation, i.e., we have the following theorem:

Theorem 6 (Summarization of Intersections). $\bigcup_{I\in\mathcal{I}} I \subseteq \bigcup_k R'_k$

3.7 Reachability Analysis

Up to now, we only considered single steps of the reachability analysis, either a Runge-Kutta step, or reducing a reachable set onto a hyperplane. In order to compute reachable sets for larger time intervals, these steps need to be iterated.

The whole reachability analysis algorithm consists again of several parts: The first part, *flow-towards-plane*, iterates Runge-Kutta steps to flow a collection of reachable sets towards a given hyperplane. This iteration includes step-size adaption, splitting of zonotopes, and finally the intersection. *flow-towards-plane* takes place in a loop of *reach* that decides which plane to flow to next.

Flowing towards One Plane. The loop of Algorithm 3, *flow-towards-plane*, maintains three kinds of sets: Flowing sets $\mathcal{F}$, intersected sets $\mathcal{I}$ and aborted

Algorithm 3. Flowing Towards one Plane

```
1: function flow-towards-plane(F_0, H)
2:     F ← F_0, I ← ∅, A ← ∅
3:     while F ≠ ∅ do
4:         (X, h) ∈ F
5:         F ← F \ {X}
6:         if width(X) ≥ max-width then                    ▷ splitting is needed
7:             (X, Y) ← split(X)
8:             F ← F ∪ {(X, h), (Y, h)}
9:         else
10:            (h, ε, X_1, X_C) ← rkstep(X)
11:            assert(safe(X_C))
12:            if reject(ε) then F ← F ∪ {(X, h/2)}          ▷ step size control
13:            else if X_C ∩ H ≠ ∅ then
14:                (A, X, I') ← intersect-flow(X, H)
15:                assert(safe(X))
16:                A ← A ∪ {A}; I ← I ∪ I'
17:            else if abort(X) then          ▷ abort when direction of flow changes
18:                A ← A ∪ {X}
19:            else F ← F ∪ {(X, adapt-stepsize(h, ε))}    ▷ step size control
20:    return (A, I)
```

sets $\mathcal{A}$, all reachable sets are checked to be *safe* with respect to some given specification in the loop (lines 12,17). The sets in $\mathcal{F}$ are associated with a step size h. All sets are supposed to flow towards a given plane H. Inside the loop, *flow-towards-plane* decides if sets need to be split (line 7), it performs a Runge-Kutta step in line 11 and decides from the discretization error ε whether the step size was too large and needs to be rejected (line 13). If close to the hyperplane, an intersection is performed. Sets may also be aborted when the direction of the flow changes (line 19). If otherwise successful, the step size is allowed to grow in line 22, depending on the discretization error.

Assuming that $\bigcup_{F\in\mathcal{F}_0} F \subseteq H^{\geq}$, the invariant that the algorithm maintains in its while loop is given in the following theorem.

Theorem 7 (Invariant of *flow-towards-plane*).

$$\left(\bigcup_{F\in\mathcal{F}_0} F\right) \curvearrowright_{\mathit{safe}} \left(\left(\bigcup_{X\in(\mathcal{A}\cup\mathcal{F})} X \cap H^{\geq}\right) \cup \left(\bigcup_{I\in\mathcal{I}} I \cap H\right)\right)$$

The flows ending in $\mathcal{A}$ or $\mathcal{F}$ can be restricted to the half-space above, because the parts of the sets below the plane is taken care of by the intersection. They need to be restricted because it cannot be guaranteed that they are always above H (splitting might introduce overapproximations). The flows to the intersections I need to be restricted to the plane, because the computed sets can also be overapproximations.

Flowing from Plane to Plane. Algorithm *flow-towards-plane*$(\mathcal{F}_0, H)$ flows reachable sets from $\mathcal{F}_0$ towards a plane H and returns sets $\mathcal{I}$ that intersect the

Algorithm 4. Reachability from Plane to Plane

```
1: function reach(F_0)
2:     H ← choose-plane(F_0, d), X ← ∅, 𝔉 ← {(F_0, H)}
3:     while 𝔉 ≠ ∅ do
4:         (F, H) ∈ 𝔉
5:         𝔉 ← 𝔉 \ {F}
6:         (A, I) ← flow-towards-plane(F, H)
7:         H_A ← choose-plane(A, 0)          ▷ aborted sets – collect as soon as possible
8:         H_I ← choose-plane(I, d)          ▷ regular intersection – collect after distance d
9:         if abort(H) then X ← X ∪ I
10:        else 𝔉 ← 𝔉 ∪ (I, H_I)
11:        𝔉 ← 𝔉 ∪ (A, H_A)
12:    return X
```

plane H and sets $\mathcal{A}$ that have been aborted before. Then *choose-plane* selects different planes that determine where the sets in $\mathcal{I}$ and $\mathcal{A}$ supposed to flow next. We sketch *choose-plane* only informally: For the sets in $\mathcal{I}$, one determines the strongest direction $d_{\mathcal{I}}$ imposed by the dynamics and has them flow towards a plane located a certain distance d in the strongest direction and perpendicular to that direction (Figure 2). For the aborted sets in $\mathcal{A}$, one similarly determines the strongest direction $d_{\mathcal{A}}$, but places the plane directly next to the sets. Reducing the sets with intersections directly after switching the direction of the flow turned out to be an effective means to keep the reachable sets small. For simplicity, we only choose axis-perpendicular hyperplanes: experiments have suggested that arbitrary hyperplanes do not necessarily lead to better performance.

The final result for our reachability Algorithm 4 reads as follows: if the algorithm *reach*$(\mathcal{F}_0)$ returns $\mathcal{X}$, then the sets from $\mathcal{F}_0$ flow towards $\mathcal{X}$, passing only through safe sets:

Theorem 8. *reach*$(\mathcal{F}_0) = \mathcal{X} \implies \left(\bigcup_{F\in\mathcal{F}_0} F\right) \curvearrowright_{\mathit{safe}} \left(\bigcup_{X\in\mathcal{X}} X\right)$.

4 Implementation

We presented our algorithms on an abstract level, but refined them (still verified in Isabelle/HOL) towards an executable specification, using Lammich's [19] framework. From the executable specification, Isabelle/HOL allows to generate Standard ML code. When executing it, we have to trust the (mostly syntactic) translation from terms in Isabelle/HOL to Standard ML. We also trust the compiler (PolyML 5.5.2) together with its library for big integers.

The working sets $\mathcal{I}, \mathcal{F}, \mathcal{A}, \mathfrak{F}$ in Algorithms 2, 3, and 4 for example are implemented using lists. Their elements, the reachable sets X, A, I are represented by affine forms, which are represented by the list of their generators $\langle a_0, \ldots, a_k\rangle$. Most of the generators of an affine form are zero, which is why affine forms are represented more efficiently as sparse lists. Moreover we keep the invariant that the sparse lists are sorted, which allows for efficient implementation of binary

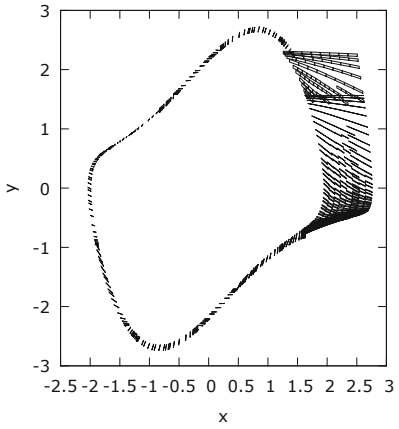

Fig. 3. Van-der-Pol, $w = 175$

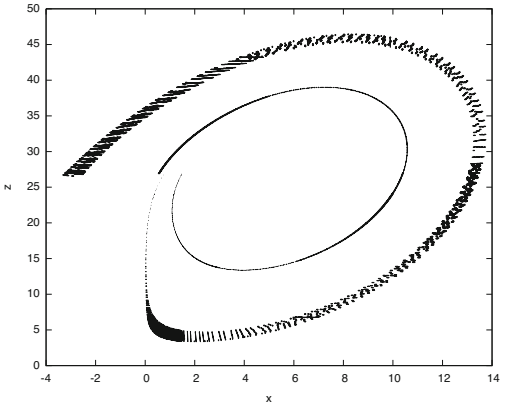

Fig. 4. Least (inner) and most (outer) chaotic IVP of the Lorenz system under study

operations like addition or multiplication. Real numbers are implemented using pairs of integers $m, e \in \mathbb{Z}$, which represent the real number $m \cdot 2^e$. For these idealized floating point numbers, rounding is performed explicitly.

The abstract algorithms we presented here consist of roughly 300 lines of code in our abstract formalization. Including the library for affine arithmetic and real numbers, the generated code consists of more than 5500 lines. The verification of the algorithms presented here can be estimated with approximately 4500 lines of code, but this number does not include the mathematical background theory about ODEs, which consists of about 6000 lines.

5 Experiments

We evaluate the performance and capabilities of our algorithm on small, classical examples of nonlinear ODEs and compare our implementation with VNODE-LP (version 0.3) and the Taylor model based tool Flow* (version 1.2). Both tools perform neither splitting nor some sort of reduction. We also try to do comparisons with Bak's [3] approach, which we call Flow*-PI: Bak experiments in Flow* with manually declaring hyperplanes ("pseudo-invariants") for reduction. Recall that in contrast to Flow*-PI, our algorithm determines the hyperplanes for reductions automatically.

Van-der-Pol. For the Van-der-Pol oscillator (Figure 3, plotted from the output of our verified algorithm), which is given by the ODE $\dot{x} = y; \quad \dot{y} = (1-x^2)y - x$, we consider initial value problems $x_0 \in 1.25 + w \cdot [0, 0.01], y_0 \in [2.28; 2.32]$ and vary the size of the initial set with the parameter w. For $w = 30$, Althoff [1] reports a run-time of 23 seconds. Since different parameters (e.g., step size, order of Taylor models, error tolerance) can be chosen for the different tools, it is hard to perform an objective comparison. We tried to be fair by setting the parameters to result

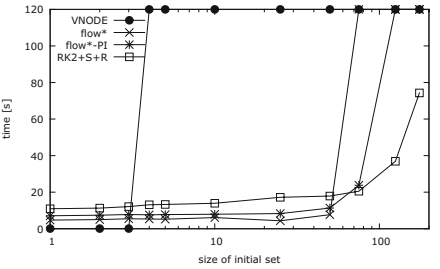

Fig. 5. Run-time for growing initial sets of the Van-der-Pol system

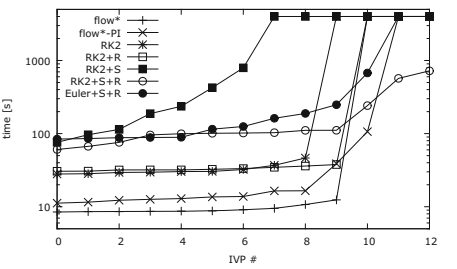

Fig. 6. Run-time for increasingly chaotic initial value problems of the Lorenz system

in comparable step sizes (0.01) for Flow* and our algorithm. An adaptive order of 4-6 seemed like the best compromise between performance and accuracy for Flow*, a further parameter is 10^{-5} for the remainder estimation.

Figure 5 summarizes the results of our experimentation: it shows the run-time for VNODE-LP, Flow*, Flow*-PI and our tool RK2+S+R (splitting and reduction enabled). Failed attempts are set to 120 seconds. VNODE-LP can only handle small initial sets. The tool Flow* can handle initial sets up to size $w = 50$, and takes between 3 and 8 seconds. For the same problems, our tool takes between 10 and 18 seconds. It scales with larger initial sets and is the only one that can handle $w \in \{125, 175\}$. This is due to the very effective reduction taking place at $x \approx 1.5$, when $y \approx -1$, as can be seen in Figure 3. Manually inserting hyperplanes for reduction at $y = 0$ and $x = 1.5$ allows Flow*-PI to integrate $w = 75$ in 24 seconds. We were unable to come up with hyperplanes that would allow Flow*-PI integration for larger values of w.

Lorenz. Consider the classical Lorenz system $\dot{x} = 11.8x - 0.29(x + y)z$; $\dot{y} = -22.8y + 0.29(x + y)z$; $\dot{z} = -2.67z + (x + y)(2.2x - 1.3y)$ in Jordan normal form. We experiment with 13 initial sets of width 0.005 along the line segment between $(0.74, 2.21, 27)$ and $(1.5, 2.25, 27)$. The dynamics exhibits with smaller values for x more and more chaotic behavior. Enclosures of the least and most chaotic problem (computed with our verified algorithm) are depicted in Figure 4.

We toggle the different optimizations of our tool in order to study their respective effects. Moreover we compare our tool with Flow* and Flow*-PI (VNODE-LP fails to integrate any of those problems). For Flow*-PI, we chose to reduce at $x = 2, z = 27$, and $x = 6$, which gives reductions similar to our algorithm: compare Figure 4, where one can see reductions at $x \approx 1.5$ (at $z \approx 5$) and $z = 27$ (at $x \approx 13$). The results are summarized in Figure 6 and we interpret them as follows. Flow* is fastest, but fails on the three most chaotic problems. Flow*-PI allows to solve one additional problem. The Runge-Kutta method with reduction and splitting (RK2+S+R) allows to solve all of the problems, utilizing the Euler-method (Euler+S+R) shows similar scaling behavior but is less efficient. Just RK2 and RK2 with reduction (RK2+R) are more efficient when the dynamics is less chaotic, but promptly fail (similar to Flow*) when chaos takes over. In

Table 1. Comparison for two particular IVPs of the Lorenz system

IVP	method	step size	time [s]	error(x)
#8: $(0.94, 2.16, 27)$	*rk2*, 10^{-5}	$7 \cdot 10^{-4}$	194	0.14
	rk2, $2 \cdot 10^{-4}$	$2 \cdot 10^{-3}$	67	0.24
	rk2, $2 \cdot 10^{-2}$	$5 \cdot 10^{-4}$	286	0.9
	Flow*	$5 \cdot 10^{-3}$	13	0.02
	Flow*-PI	$5 \cdot 10^{-3}$	16	0.3
#10: $(0.79, 2.14, 27)$	*rk2*, 10^{-5}	$2 \cdot 10^{-4}$	595	0.3
	rk2, $2 \cdot 10^{-4}$	$6 \cdot 10^{-4}$	241	0.5
	rk2, $2 \cdot 10^{-2}$	$1 \cdot 10^{-4}$	1648	1.3
	Flow*	$5 \cdot 10^{-4}$	121	5.8
	Flow*-PI	$5 \cdot 10^{-4}$	106	0.5

summary, this shows that splitting is essential for handling chaotic systems, but (as can be seen at RK2+S) does not scale without reduction.

For another comparison, we study the effect of different strategies for step-size adaptation: we vary the threshold of discretization error for rejecting steps between $10^{-5}, 10^{-4}, 10^{-2}$. Table 1 shows that (at least for good performance) a compromise needs to be found: small local errors require more, smaller steps, but allowing for too large local errors results in larger sets, therefore more splitting and worse performance.

Comparing the performance of *rk2*, $2 \cdot 10^{-4}$ with Flow* and Flow*-PI in Table 1, we can see that on the easier problem #8, Flow* is very efficient: it achieves better precision despite larger step size. On the more complicated problem #10, Flow* fails to achieve the same accuracy, because the reachable sets grow too large. This problem is successfully addressed by the reductions performed in Flow*-PI and our method. Compared with Flow*-PI, our method achieves with slightly larger step sizes the same accuracy, it is a bit more than twice as slow, but it does not need manual interaction for choosing the reductions.

6 Conclusion

We presented a formally verified analyzer for continuous systems given by ODEs. Its performance is in the range of other, non-verified tools, and even scales better than them in the presence of large initial sets and chaotic dynamics. More importantly, our algorithm introduces a new level of mathematical rigor and therefore trust to analyzers for continuous systems.

Discussion. There is no single best approach to reachability analysis of ODEs, therefore many of our design decisions were guided heuristically. Optimizations like splitting and reduction to hyperplanes are only effective for low-dimensional systems. Concerning splitting of reachable sets, an alternative could be to use a more complex data structure like Taylor models that directly represent non-convex sets. It seems, however, that splitting is also necessary for Taylor model

based analysis tools, as could be seen in section 5. Another possibility to reduce the reachable sets without geometric intersections has been proposed by Althoff [2], but it depends on the problem at hand which one is more efficient and/or precise.

Future work. Since we support intersection of reachable sets with hyperplanes, we should be able to generalize the approach to handle switching surfaces of hybrid systems. Moreover we aim to propagate more topological information (e.g. partial derivatives) of the flow in order to be able to certify the computations for the existence of the Lorenz-attractor [24].

Acknowledgements. I would like to thank the anonymous reviewers for their helpful feedback and in particular for pointing me to Bak's work [3].

References

1. Althoff, M.: Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In: Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control, HSCC 2013, pp. 173–182. ACM, New York (2013)
2. Althoff, M., Krogh, B.H.: Avoiding geometric intersection operations in reachability analysis of hybrid systems. In: Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2012, pp. 45–54. ACM, New York (2012)
3. Bak, S.: Reducing the wrapping effect in flowpipe construction using pseudo-invariants. In: Proceedings of the 4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems, CyPhy 2014, pp. 40–43. ACM, New York (2014)
4. Balluchi, A., Casagrande, A., Collins, P., Ferrari, A., Villa, T., Sangiovanni-Vincentelli, A.L.: Ariadne: a framework for reachability analysis of hybrid automata. In: Proceedings of the 17th International Symposium on Mathematical Theory of Networks and Systems (MTNS 2006), Kyoto, Japan (July 2006)
5. Berz, M., Makino, K.: Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. Reliable Computing 4(4), 361–369 (1998)
6. Boldo, S., Clment, F., Fillitre, J.C., Mayero, M., Melquiond, G., Weis, P.: Wave equation numerical resolution: A comprehensive mechanized proof of a C program. Journal of Automated Reasoning 50(4), 423–456 (2013)
7. Bouissou, O., Chapoutot, A., Djoudi, A.: Enclosing temporal evolution of dynamical systems using numerical methods. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 108–123. Springer, Heidelberg (2013)
8. Brisebarre, N., Joldeş, M., Martin-Dorel, É., Mayero, M., Muller, J.-M., Paşca, I., Rideau, L., Théry, L.: Rigorous polynomial approximation using Taylor models in Coq. In: Goodloe, A.E., Person, S. (eds.) NFM 2012. LNCS, vol. 7226, pp. 85–99. Springer, Heidelberg (2012)
9. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 258–263. Springer, Heidelberg (2013)

10. Collins, P., Niqui, M., Revol, N.: A validated real function calculus. Mathematics in Computer Science 5(4), 437–467 (2011)
11. de Figueiredo, L., Stolfi, J.: Affine arithmetic: Concepts and applications. Numerical Algorithms 37(1-4), 147–158 (2004)
12. Fränzle, M., Herde, C., Ratschan, S., Schubert, T., Teige, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. Journal on Satisfiability, Boolean Modeling and Computation 1, 209–236 (2007)
13. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011)
14. Girard, A.: Reachability of uncertain linear systems using zonotopes. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 291–305. Springer, Heidelberg (2005)
15. Immler, F.: Formally verified computation of enclosures of solutions of ordinary differential equations. In: Badger, J.M., Rozier, K.Y. (eds.) NFM 2014. LNCS, vol. 8430, pp. 113–127. Springer, Heidelberg (2014)
16. Immler, F.: A verified algorithm for geometric zonotope/hyperplane intersection. In: Proceedings of the 2015 Conference on Certified Programs and Proofs, CPP 2015, pp. 129–136. ACM, New York (2015)
17. Immler, F., Hölzl, J.: Numerical analysis of ordinary differential equations in Isabelle/HOL. In: Beringer, L., Felty, A. (eds.) ITP 2012. LNCS, vol. 7406, pp. 377–392. Springer, Heidelberg (2012)
18. Immler, F., Hölzl, J.: Ordinary differential equations. Archive of Formal Proofs (February 2015), Formal proof development, `http://afp.sf.net/devel-entries/Ordinary_Differential_Equations.shtml`
19. Lammich, P.: Refinement for monadic programs. Archive of Formal Proofs (2012), Formal proof development, `http://afp.sf.net/entries/Refine_Monadic.shtml`
20. Makarov, E., Spitters, B.: The Picard algorithm for ordinary differential equations in Coq. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) ITP 2013. LNCS, vol. 7998, pp. 463–468. Springer, Heidelberg (2013)
21. Nedialkov, N.: Implementing a rigorous ODE solver through literate programming. In: Rauh, A., Auer, E. (eds.) Modeling, Design, and Simulation of Systems with Uncertainties, Mathematical Engineering, vol. 3, pp. 3–19. Springer, Heidelberg (2011)
22. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL. LNCS, vol. 2283. Springer, Heidelberg (2002)
23. Platzer, A.: The complete proof theory of hybrid systems. In: Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science, LICS 2012, pp. 541–550. IEEE Computer Society, Washington, DC (2012)
24. Tucker, W.: A rigorous ODE solver and Smale's 14th problem. Foundations of Computational Mathematics 2(1), 53–117 (2002)
25. Zumkeller, R.: Formal global optimisation with Taylor models. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 408–422. Springer, Heidelberg (2006)