Extraction of a computer-certified ODE solver

Grigory Devadze, Lars Flessing, Stefan Streif*

Abstract—Reliably determining system trajectories is essential in many analysis and control design approaches. To this end, an initial value problem has to be usually solved via numerical algorithms which rely on a certain software realization. Because software realizations can be error-prone, proof assistants may be used to verify the underlying mathematical concepts and corresponding algorithms. In this work we present a computer-certified formalization of the solution of the initial value problem of ordinary differential equations. The concepts are performed in the framework of constructive analysis and the proofs are written in the Minlog proof system. We show the extraction of a program, which solves an ODE numerically and provide some possible optimization regarding the efficiency. Finally, we provide numerical experiments to demonstrate how programs of a certain high level of abstraction can be obtained efficiently. The presented concepts may also be viewed as a part of preliminary work for the development of formalized nonlinear control theory, hence offering the possibility of computer-assisted controller design and program extraction for the controller implementation.

I. INTRODUCTION

We consider the initial value problem

\[ \dot{x} = f(t, x), \quad x(0) = x_0, \]

where \( f : D \rightarrow \mathbb{R}^n \) is locally Lipschitz in \( x \), continuous in \( t \) and is defined on \( D \subseteq \mathbb{R}^+ \times \mathbb{R}^n \) with \( (0, x_0) \in D \). It is well known that the ordinary differential equations (ODE) do not possess closed-form solutions in general. Thus the initial value problem is an important subject in numerical system analysis and controller design. Usually to apply a numerical method, one would need to predetermine the necessary parameters e.g. precision and step size which may be difficult to obtain beforehand in general. Additionally, pure numerical approximations are not suitable for safety critical applications which require a certain level of rigor and correctness guarantees (see [1] for general IEEE-754 floating point number issues or [2, 3] for numerical issues which arise in avionics).

The gap between a mathematical concept and its implementation is sometimes also called algorithmic uncertainty. The importance of deriving (as opposed to just implementing) correct precision-aware controllers within the so-called algorithmic control theory has been highlighted in [4]. Due to the mathematical nature of the problem, computer-assisted formal methods may be used to close this gap and to guarantee correctness fully [5]. In this work we rely on the use of proof assistant software, which have been already successfully applied for important mathematical problems [6]. Moreover we are interested in the so-called proofs-as-programs paradigm to investigate the relation of proofs with computer programming. This procedure is also called program extraction, i.e. a certain proof of the existential theorem corresponds to a computer program which realizes the desired properties correctly. This paradigm can be especially interesting for controller design. Consider e.g., a controller implemented as a software program which has to guarantee a desired property such as stability of the closed-loop system. Note that even though development tools such as MATLAB possess certain possibilities for formal code verification [7], this does not guarantee that the implemented controller fulfills the desired properties because the controller itself did not go through a formal analysis. In this work we contribute towards formalized nonlinear control theory with automated program extraction. Please note that related works are reviewed in Section II. As a first step towards this goal, we focus on initial value problems for ODEs, which is undoubtedly a key concept in control theory.

Contributions and outline of this work. The aim of the present work is to provide computer-assisted proofs for the initial value problem (1). As we are interested in the proofs-as-programs paradigm, we obtain an extracted program, which solves the ODE numerically. Our framework is carried out within a particular realization of exact real arithmetic: constructive analysis. The extracted program is still of a certain high level of abstraction, but can be also run as a usual numerical program.

We start with a review of related work on guaranteed and formal methods in Section II. The employed proof system and the preliminaries of formalization are explained in Section III. The main result, i.e. the formalization of the Cauchy-Euler approximation is derived in Section IV. In Section V we show the application of the derived program with some initial examples and compare the result with existing approaches in proof assistants. In Section 5 we will give an outlook for how these concepts can be further used as a preliminaries for the development of formalized nonlinear control theory.

II. RELATED WORKS

To overcome the limitations of missing guarantees in numerical schemes, many different approaches for verified integration have been proposed. [8] presented a method based on local modelling with high-order Taylor polynomials and a bound on the remainder term. [9] utilized the ValEncIA-TVP to verify whether the enclosure of all reachable states remains within given bounds for known control strategies. However these approaches are still numerical and although the used
concepts are of a high abstraction level (e.g. Schrauder's fixed point theorem in ([8]), they do not have a formal connection to the actual programming code. In the context of dynamical systems, [10] formally proves the correctness of a program implementing the numerical resolution of a wave equation, with Frama-C ([11]). In [12], the authors formalize variational equations related to the solutions of ordinary differential equations in Isabelle/HOL [13]. With respect to the initial value problem the work of [14] provides the verified implementation of an approximation of a solution of an ODE based on arbitrary precision floating-point numbers. In the present work, the implementation is done by the accurate representation of special infinite sequences of rational numbers. This allows a parametrization of the algorithm by the demanded error bound with the guarantee that this bound holds. Similarly it was done in [15] where the algorithm is derived from the Picard-Lindelöf theorem. However, the level of abstraction of the theorem is very high, such that the running times of the algorithm on even small instances are unusable in practice. In the current work, the level of abstraction is reduced and the approximate solution of the initial value problem is derived efficiently in a direct way by means of the program extraction techniques. Additionally the correctness of the programs follows immediately given from the Soundness Theorem [16].

Proof assistants also play an important role in formal verification of control systems. [17] used Isabelle for implementation of their control system formal verification based on Hybrid Hoare logic. The Why3 platform [18] coupled with MATLAB/Simulink was used in [19, 20] to perform simple stability checks of linear discrete systems with quadratic Lyapunov functions. [21] mentioned several proof assistant software tools to be considered for implementation of formal verification of control systems, including [22], HOL-Light [23], Isabelle/HOL and Lean [24]. A closely related work is due to [25] where the authors present the framework for the certified programs for robots based on constructive is due to [25] where the authors present the framework for the certified programs for robots based on constructive

III. MINLOG PROOF SYSTEM

In this section we introduce some preliminary concepts of the proof system \(\text{Minlog}\), the notion of totality and the concept of inductive definitions. Afterwards we give the details how the formalization works in constructive setting.

A. Necessary preliminaries

In the present work, the analysis and formalization are done within the framework of constructive analysis with a special focus on a concrete and efficient computational realization as well as formal verification.

The interactive proof system \(\text{Minlog}\) has been developed with the special intention of the program extraction from mathematical proofs. The atomic constructions are done by introducing of free algebras as types which are given by its constructors. The computation is implemented efficiently by the usage of the so-called normalization-by-evaluation. The reason of using of typed theories is that the normalization-by-evaluation technique, which is applied to the proof terms, needs typed formulas. As an example consider the construction of the type \(\text{nat}\) of natural numbers \(\mathbb{N}\) by stating that \(\text{zero} = \text{nat}\) and \(\text{successor} = \text{nat}\) and the computational rules are given recursively as:

\[
\begin{align*}
\text{NatTimes}\ n\ \text{zero} & \rightarrow \text{zero} \\
\text{NatTimes}\ n\ \text{succ}\ m & \rightarrow \text{NatTimes}\ n\ m + n
\end{align*}
\]

B. Inductive definitions and theorem proving

Before one starts to prove theorem with the help of the computer, one has to state some definitions which are relevant for the problem statement. In \(\text{Minlog}\) it is done by the introducing the so-called inductively defined predicate constant (IDPC). Informally speaking, these are formations of implications which leads to the proposition which has to be defined.

As an example assume that one wants to provide a definition for rational numbers which lie in an unit interval. Then the corresponding IDPC \(\text{RatInUnitI}\) is given by:

\[
\forall a \in \mathbb{Q}\ 0 \leq a \rightarrow a \leq 1 \rightarrow \text{RatInUnitI}\ a
\]

which means semantically that to show \(\text{RatInUnitI}\ a\) one has to show that \(0 \leq a\) and \(a \leq 1\).

C. Preliminaries for the formalization in constructive setting

This part is based on the foundation of constructive analysis with witnesses [30]. The most important object in the presented setting is a real number. In \(\text{Minlog}\) reals are built by introducing the typed variables \(\mathbb{N} \Rightarrow \mathbb{Q}, \mathbb{P} \Rightarrow \mathbb{N}\), defining the constructor \(\text{Real Constr} : (\mathbb{N} \Rightarrow \mathbb{Q}) \Rightarrow (\mathbb{P} \Rightarrow \mathbb{N}) \Rightarrow \mathbb{R}\) and introducing the IDPC \(\text{Real}\) i.e. the real \(x\) is given by a regular Cauchy sequence of rationals \((a_n)\) with a given weakly increasing modulus \(M\).That is \(\forall p, m \cdot n \cdot (a_{m+n} - a_m) \leq 2^{-p}\) whenever \(M(p) \leq n, m\) and \(\forall p, q \cdot p \leq q \rightarrow M(p) \leq M(q)\).

Two reals \(x = ((a_n)_n, M)\) and \(y = ((b_n)_n, B)\) are equal, if \(\forall p \in \mathbb{P}\ \left|a_{M(p+1)} - b_{N(p+1)}\right| \leq 2^{-p}\). A real \(x\) is called
non-negative if $\forall p \in \mathbb{P} \; 2^{-b} \leq (a_n)_{M(p)}$. For a real $x$ denote $x \in_p \mathbb{R}^+ \; (\text{alternately } 0 < x)$ if $2^{-p} \leq (a_n)_{M(p+1)}$ for some $p$.

**Definition 1 (Cont):** An uniformly continuous function $f : I \rightarrow \mathbb{R}$ on a compact interval $I$ with rational endpoints is given by

1) an approximating map $h_f(I \cap \mathbb{Q}) \times \mathbb{N} \rightarrow \mathbb{Q}$ and a map $\alpha_f : \mathbb{P} \rightarrow \mathbb{N}$ such that $(h_f(a_n), n)_n$ is a Cauchy sequence with modulus $\alpha_f$.

2) a modulus $\omega_f : \mathbb{P} \rightarrow \mathbb{P}$ of continuity, which satisfies

$$|a - b| \leq 2^{-\omega_f(p)+1} \rightarrow |h_f(a, n) - h_f(b, n)| \leq 2^p$$

with $\alpha_f$ and $\omega_f$ weakly increasing.

The equivalent realisation of continuous functions in $\mathbb{R}$ is done via introducing of the type $\mathbb{P}$ and the IDPC $\mathbb{P}$, stating that $f$ is a proper continuous function. Firstly the basic constructors of $\mathbb{P}$ are:

- $h : \mathbb{Q} \Rightarrow \mathbb{N} \Rightarrow \mathbb{Q}$ (approximation mapping)
- $\alpha : \mathbb{P} \Rightarrow \mathbb{N}$ (modulus of convergence)
- $\omega : \mathbb{P} \Rightarrow \mathbb{P}$ (modulus of continuity)

and finally the constructor $\mathbb{P} : \mathbb{Q} \Rightarrow \mathbb{P} \Rightarrow h \Rightarrow \alpha \Rightarrow \omega \Rightarrow \mathbb{P}$. The property that the mapping of type $\mathbb{P}$ is truly a continuous function in the presented sense can be encoded via the following IDPC $\mathbb{P}$: $\forall(a, b, h, \alpha, \omega) :$

$$(\forall a(0 \leq a \rightarrow a \leq b \rightarrow \text{cauchy}(h(a))) \rightarrow$$

$$(\forall b, k, n(a)(0 \leq a \rightarrow a \leq b \rightarrow a_0 \leq b \rightarrow b \leq b_0 \rightarrow a \rightarrow n \rightarrow |a - b| \leq 2^{-p}) \rightarrow |h(a - h b)| \leq 2^{-p})$$

$$(\forall p, q(p \leq q \rightarrow a \rightarrow p \leq q)(q(p \leq q \rightarrow \omega p \leq \omega q)) \rightarrow \mathbb{P}.$$ 

As an example consider the representation of the function $f(x) = x^2$ on the interval $[0, 2]$:

$\mathbb{P}(\text{cont} \theta 2 \; \left(\begin{array}{c} [a, n] \Rightarrow a \Rightarrow (p \Rightarrow 0) \Rightarrow (p \Rightarrow p+3) \end{array}\right) \Rightarrow \text{ContConstr} \Rightarrow \mathbb{Q} \Rightarrow \mathbb{P} \Rightarrow h \Rightarrow \alpha \Rightarrow \omega \Rightarrow \mathbb{P}.$

The definition of the vector-valued function is provided similarly:

**Definition 2 (ContVec):** An uniformly continuous vector-valued function $f : B \rightarrow \mathbb{R}^n$ on a compact ball $B(c, R) \subset \mathbb{R}^n$ with rational center $c$ and rational radius $R$ is given by

1) an approximating map $h_f(B \cap \mathbb{Q}^n) \times \mathbb{N} \rightarrow \mathbb{Q}$ and a map $\alpha_f : \mathbb{P} \rightarrow \mathbb{N}$ such that $(h_f(a_n), n)_n$ is a Cauchy sequence with modulus $\alpha_f$.

2) a modulus $\omega_f : \mathbb{P} \rightarrow \mathbb{P}$ of continuity, which satisfies

$$\|a - b\|_1 \leq 2^{\omega_f(p)+1} \rightarrow \|h_f(a, n) - h_f(b, n)\|_1 \leq 2^p$$

with $\alpha_f$ and $\omega_f$ weakly increasing.

**Remark 3:** The choice of the norm $\|\cdot\|_1$ influences the values of moduli $\alpha_f$ and $\omega_f$. Usually, one would prefer to use either 1-norm or \infty-norm for computational purpose, since for rational vectors the values of the norm are also rationals, which is not the case, e.g., for the Euclidean norm.

**Definition 4 (Applic):** The application of an uniformly continuous function $f : I \rightarrow \mathbb{R}$ to a real $x := (a_n, (M))$ in $I$ is defined to be a Cauchy sequence $h_f(a_n, (M))$ with modulus $\max(\alpha_f(p+2), M(\omega_f(p+2)-1))$. We denote this real by $f(x)$.

The following theorem is necessary to perform computations with the real numbers.

**Theorem 5 (RealApprox):** $\forall x, p \geq 0(\|a - x\|_1 \leq 2^{-p})$.

At this point an easy example of the program extraction in $\mathbb{R}$ is provided. Consider the following goal as an equivalent to the theorem statement above:

$$\forall x, p \Rightarrow \text{Real}(\text{abs} \rightarrow \text{abs}(a-x) \leq (1|\text{#2}|*p))$$

After unfolding the definition of $x = ((\alpha_n), M)$ and assuming universally quantified variables $(\alpha_n), M$ and the hypothesis Real $x$ the new goal is

$$\exists a \Rightarrow \text{abs}(a-(\text{RealConstr} \rightarrow (\alpha_n), (M))) \leq (1|\text{#2}|*p).$$

Here the existential quantifier can be eliminated by the term (ex-intro $\text{pt}$ “$(\alpha_n)$”). The rest of the proof uses usual techniques to prove that the $\|a_n - ((\alpha_n), M)\|_1 \leq 2^p$. Once the existential proof of the theorem is done, this theorem can be animated. In case of the animation of $\text{RealApprox}$ one obtains a program constant with the following computation rule:

$c\text{RealApprox} \rightarrow \{x0, p\}$

$\{f \times 0 : (\text{abs}(\text{as2}),(\text{as2}))(\text{as2})\}$ which is an extracted program from the constructive proof of the theorem $\text{RealApprox}$. Moreover, one can use the extracted program in the proof of the following theorem and generate new program:

**Theorem 6 (RealVecApprox):** $\forall x \in \mathbb{R}^n, p \exists e \in \mathbb{Q}^n \Rightarrow \text{abs}(e - x) \leq 2^{-p}$.

**Proof:** Proceed by induction on $n$. For $n = 1$ the existence holds by the theorem $\text{RealApprox}$ and is realized by $c\text{RealApprox}$. For $n = 2$ and for any vector real number $x = (x_1, x_2)$ generate a vector $e = (e_1, e_2)$ with $|e_1 - x_1| \leq 2^{-(p+1)}, |e_2 - x_2| \leq 2^{-(p+1)}$ again by the Theorem $\text{RealApprox}$. Clearly $|e_1 - x_1| + |e_2 - x_2| \leq 2^{-p}$. For the induction step assume that for the vector $x' = (x_1', ..., x_n')$ there exists a vector $e'$ with $\|e' - x'\| \leq 2^{-(p+1)}$ and for the real number $x_{n+1}$ there exists a rational number $e_{n+1}$ again by $\text{RealApprox}$ such that $|e_{n+1} - x_{n+1}| \leq 2^{-(p+1)}$. Then for the vector $x = (x_1, ..., x_{n+1})$ the vector $e = (e_1, ..., e_{n+1})$ satisfies the desired condition and is a $2^{-p}$ approximation to a real vector $x$. 

Thus we are able to perform correct computations with vector-valued functions, since the program constant $\text{Application}$ is capable to generate new real number vectors $f(x)$ correctly.

**Definition 7 (Time derivative):** Let $\varphi_1 : [T_1, T_2] \rightarrow \mathbb{R}^n, \varphi_2 : [T_1, T_2] \times B \rightarrow \mathbb{R}^n$ be continuous in sense of Definition 2 and $0 \leq T_1 < T_2$. The function $\varphi_2$ is called partial derivative w.r.t. the time of $\varphi_1$ with the modulus of differentiability $\delta_{\varphi_1} : \mathbb{P} \rightarrow \mathbb{P}$ if for $t_1, t_2 \in [T_1, T_2]$ with $t_1 < t_2$.

$$\forall p \Rightarrow t_2 \leq t_1 + 2^{-\delta_{\varphi_1}(p)} \rightarrow$$

$$|\varphi_1(t_2) - \varphi_1(t_1) - \varphi_2(t_1, x)(t_2 - t_1)| \leq 2^{-p}(t_2 - t_1)$$

We denote the function $\varphi_2$ as $\varphi_1$ and call the function $\varphi_1$ differentiable on $[T_1, T_2]$.

The next steps allow to generate a uniform partition of a rational interval such that the functions can be evaluated at
discrete values.

**Definition 8 (Partition):** Let \( a, b \) be rational numbers such that \( a < b \). A list \( P = a_0, \ldots, a_n \) of rationals is a partition of the interval \([a, b]\), if \( a = a_0 \leq a_1 \leq \cdots \leq a_n = b \).

**Theorem 9 (RatLeAbsBound):** \( \forall a \in \mathbb{Q} \ \exists P \in \mathbb{P} \ |a| \leq 2^p \).

The corresponding extracted program of type \( \text{rat} \Rightarrow \text{pos} \) is denoted as \( \text{cRatLeAbsBound} \).

**Definition 10:** The program constant \( \text{UnifP} \) of type \( \text{rat} \Rightarrow \text{rat} \Rightarrow \text{rat} \Rightarrow (\text{list} \ \text{rat}) \) is recursively defined as follows:

- \( \text{UnifP} \) \( \text{Zero} \ a \ b \rightarrow b \):
- \( \text{UnifP} \) \( \text{Succ} \ n \ a \ b \rightarrow \text{UnifP} \ n \ a \ ((a+b)/2) \) where \( :+\) is list concatenation.

**Definition 11:** The program constant \( \text{EulerMap} \) of type \( \text{rat} \Rightarrow \text{rat} \Rightarrow \text{rat} \Rightarrow (\text{list} \ \text{rat}) \) is recursively defined as:

\[
\begin{align*}
\forall b, p \ \text{Partition} \ 0 \ b \ 2^{-p} \ \text{UnifP} \ b \ p \\
\end{align*}
\]

\( \text{UnifP} \) is indeed a differential equation (properly stated in our constructive framework):  \( \dot{x} = f(t, x), \quad x(0) = x_0. \)  (2)

Although the proof is standard, we provide it here for completeness.

**Theorem 14 (CauchyEulerApproxSolCorr):** Let \( f : B \rightarrow \mathbb{R}^n \) with \( B(c, R) \subset \mathbb{Q}^n \) be an uniformly continuous vector-valued function and \((0, x_0) \in B\). Assume that for the set \( B' \) given by \(|t| \leq t_0, \|x - x_0\| \leq x_b \) is in \( B\). Assume there exists a rational number \( C > 0 \) such that \( \|f(t, x)\| \leq C \) for \((t, x) \in B'\). Then for every \( p \in \mathbb{P} \) there exist an approximate solution \( \varphi : [0, T] \rightarrow \mathbb{Q}^n \) up to the error \( 2^{-p} \) in the sense that

\[
\|\varphi(t) - f(t, \varphi(t))\|_1 \leq 2^{-p}
\]

for all \( t \in [0, T] \) with \( T = \min \left\{ t_0, \frac{1}{2C} \right\} \) where \( \varphi(t) \) is defined.

**Proof:** For any \( p \in \mathbb{P} \) obtain a number \( q \in \mathbb{P} \) by the continuity such that:

\[
\|f(t, x) - f(t, y)\|_1 \leq 2^{-p+1},
\]

whenever \( \|x - y\|_1, |t_1 - t_2| \leq 2^{-q} \). Generate a partition \( P = c_0, \ldots, c_n \) by applying the program constant \( \text{UnifP} \) of the interval \([0, T]\) such that \( \max\{|c_{i+1} - c_i| : i < n\} \leq \min(2^{-q}, 2^{-p+1}) \).

Recursively construct the approximate solution \( \varphi \) via the program constant \( \text{EulerMap} \). Note that the approximate solution satisfies for \( i = 1, \ldots, n \):

\[
\varphi(t) = b_i + (t - c_i)s_{i-1} \quad \text{for} \quad c_{i-1} \leq t \leq c_i
\]

where \( s_{i-1} \) are calculated such that \( \|s_i\|_1 \leq C \) and \( \|s_{i-1} - f(b_i, c_i)\|_1 \leq 2^{-(p+1)} \) by the program constant \( \text{cRealVecApprox} \).

Observe that for \( c_{i-1} < t < c_i \)

\[
\|\varphi(t) - b_{i-1}\|_1 \leq |t - c_{i-1}|\|s_{i-1}\|_1
\]

\[
\leq 2^{-(p+1)} + 2^{-(p+1)} \leq 2^{-p}
\]

and finally,

\[
\|\varphi(t) - f(t, \varphi(t))\|_1 \leq 2^{-p}
\]

**Remark 15:** For existence and uniqueness we rely on the pen-and-paper version [31] and admit its correctness for now. Let \( f : B \rightarrow \mathbb{R}^n \) be continuous and satisfy the Lipshitz condition w.r.t. its second argument, i.e. there exists a constant \( L \) such that \( \|f(t, x_1) - x_2\| \leq L\|x_1 - x_2\| \) for any \( x_1, x_2 \in B \). Assume that the assumption of the Theorem 14 holds. Then we can construct a unique exact solution \( \psi : [0, T] \rightarrow \mathbb{R}^n \) to (2) such that \( \psi(0) = x_0 \). Moreover let \( \bar{\varphi} : [0, T] \rightarrow \mathbb{R}^n \) be an approximate solution to (2).

**Remark 16:** The extracted program \( \varphi \) is of type \( \text{ContVec} \) and fulfills the assumptions of Definition 2 (ContVec). The respective modulus of continuity \( \omega \) and modulus of convergence \( \alpha \) are derived by the proof system automatically. Moreover \( \varphi(t) \) is admissible, i.e. \( (t, \varphi(t)) \in B \) and piecewise continuous differentiable.
B. Program extraction

The direct extraction yields the invocation of the program constant EulerMapFast, which is not usable for practical computation. Thus we define new program constant EulerMapFast of the type contVec=>(ratVec pair ratVec)=>rat=>pos=>(ratVec pair ratVec) recursively as:

- EulerMapFast Zero rt d p \rightarrow rt
- EulerMapFast (Succ n) rt d p \rightarrow EulerMapFast n ((\text{rht rt}) \text{ pair})
  \begin{align*}
  \text{pair ratVec}
  \end{align*}

where \text{pair} is a constructor for a tuple, \text{rt} is a variable of type \text{(ratVec pair ratVec)} and \text{rht} accesses the right element of the tuple.

Next we showed the equivalence of outputs of the program constants EulerMap f a n d p and EulerMapFast n (Inhab rat) pair \alpha d p, where (Inhab ratVec) is an arbitrary inhabitant of rational vectors. Observe that EulerMapFast mimics a usual for loop and there are no more redundant recursive calls as in EulerMap. However EulerMap yields easier goals for the reasoning i.e. we use EulerMapFast for a computational version of the Theorem 14 where the program equivalence guarantees the correctness. Additionally we evaluate all elementary operations (+, −, ·, /, floor) on rational numbers externally in Scheme due to the fact that the operations which are defined for decidable/discrete predicates (rational numbers in this case) should not be affected by some algorithmic uncertainty. Furthermore, Scheme cancels the greatest common divisor automatically, thus we do not have to be concerned about it during the theorem proving. As for the last optimization we implemented the so-called compression of real numbers. That is, for any real \( x = ((a_n)_n,M) \) there is a new real \( y = ((b_n)_n,p+2) \) with \( (b_n)_n := \frac{\text{floor}(a_n(p+2^n))}{2^n} \) with \( x = y \). This allows us to control the length of the denominator and makes the Cauchy sequences less complex. Applying this optimization allows performing the computation more efficiently which is shown experimentally in the next section.

V. CASE STUDY

In the following we compare our results with the existing approaches in proof assistants on the following basic but not need to consider the notion of the stability of the Euler method, since our approach runs an "ideal" Euler method efficiently. Moreover, we stay within Minlog’s verification kernel except for some basic computations on rationals.

Example 2: Integrating the circle

\begin{align*}
  \dot{x} &= -y, \quad x(0) = 1 \\
  \dot{y} &= x, \quad y(0) = 0
\end{align*}

We use the following representation:

- \( h_f(a_1, a_2, a_3, n) = (-a_2, a_1) \),
- \( \alpha(p) = 0 \),
- \( \omega(p) = p \),
- \( C = 2, \quad L = 1 \)

For two correct decimal digits of the integration from 0 to \( 2\pi \) the resulting rational vector is \( \left( \frac{167790121}{22267}, \frac{16777216}{134217728} \right) \) and its timing is approximately 30 min which corresponds to approximately 26 000 verified evaluations of \( f \). Note that our assumptions are valid for any \( t \), thus we are allowed to run the algorithm successively and obtain the continuation of the solution.

Overall we conclude that our extracted program achieved relatively good performance. Additionally, observe that we are able to obtain a verified result at any desired precision in principle. Please note that extracted programs in Minlog allow the evaluation of verified high-level functions directly. Clearly the procedure of program extraction is not limited to the optimization presented in Section IV and the running time can be further reduced drastically. A detailed analysis is beyond the scope of this work and subject of current investigations. Note further that while the analysis of trajectories is often essential for the controller design and its formal verification, the controller acting online may be much simpler and may not need to determine the trajectory.

VI. CONCLUSION

We formalized the Cauchy-Euler approximation method for the solution of the initial value problem for the ordinary differential equation in Minlog. We performed the program extraction in the exact real number arithmetic and showed how the extracted program can be optimized in terms of efficiency and maintaining a certain high level of abstraction. The presented results might be fruitful for the reasoning on higher-order numerical schemes like Runge-Kutta. Furthermore the notion of system trajectory plays a crucial role in the stability of the equilibrium of nonlinear systems where the notion of uniform asymptotic stability is of a separate interest, due to the arising of special initial value problem from the comparison principle [32]. Thus the presented concepts may also be viewed as a part of preliminary work for the development of formalized nonlinear control theory.

REFERENCES

[1] B. A. Wichmann, “A note on the use of floating point in critical systems,” The Computer Journal, vol. 35, no. 1, pp. 41–44, 1992.
[2] D. Delmas, E. Goubault, S. Putot, J. Souyris, K. Tekkal, and F. Védrine, “Towards an industrial use of fluctuat on safety-critical avionics software,” in International
Workshop on Formal Methods for Industrial Critical Systems. Springer, 2009, pp. 53–69.

[3] A. E. Goodloe, C. Muñoz, F. Kirchner, and L. Corrêa, “Verification of numerical programs: From real numbers to floating point numbers,” in NASA Formal Methods Symposium. Springer, 2013, pp. 441–446.

[4] P. Tsipras and M. Mesbahi, “Toward an algorithmic control theory,” Journal of Guidance, Control, and Dynamics, vol. 40, no. 2, pp. 194–196, 2017.

[5] O. Hasan and S. Tahar, “Formal verification methods,” in Encyclopedia of Information Science and Technology, Third Edition. IGI Global, 2015, pp. 7162–7170.

[6] G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. Le Roux, A. Mahboubi, R. O’Connor, S. O. Biha et al., “A machine-checked proof of the odd order theorem,” in International Conference on Interactive Theorem Proving. Springer, 2013, pp. 163–179.

[7] C.-C. Lee and J. Friedman, “Requirements modeling and automated requirements-based test generation,” SAE International Journal of Aerospace, vol. 6, no. 2013-01-2237, pp. 607–615, 2013.

[8] M. Berz and K. Makino, Verified Integration of ODEs and Flows Using Differential Algebraic Methods on High-Order Taylor Models. Kulwer Academic Publishers, 1998, pp. 361–369.

[9] A. Rauh and H. Aschemann, Interval Methods for Verification and Implementation of Robust Controllers. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 201–211.

[10] S. Boldo, F. Clément, J.-C. Filliâtre, M. Mayero, G. Melquiond, and P. Weis, “Wave equation numerical resolution: a comprehensive mechanized proof of a C program,” Journal of Automated Reasoning, vol. 50, no. 4, pp. 423–456, 2013.

[11] P. Cuqu, F. Kirchner, N. Kosmatov, V. Prevost, J. Signoles, and B. Yakobowski, “Frama-c,” in International Conference on Software Engineering and Formal Methods. Springer, 2012, pp. 233–247.

[12] F. Immler and C. Traut, “The flow of ODEs: Formalization of variational equation and Poincaré map,” Journal of Automated Reasoning, vol. 62, no. 2, pp. 215–236, 2019.

[13] T. Nipkow, L. Paulson, and M. Wenzel, Isabelle/HOL — A Proof Assistant for Higher-Order Logic, ser. LNCS. Springer, 2002, pp. 2283.

[14] F. Immler, “A verified ODE solver and the Lorenz attractor,” Journal of Automated Reasoning, vol. 61, no. 1–4, pp. 73–111, 2018.

[15] E. Makarov and B. Spitters, “The Picard algorithm for ordinary differential equations in Coq,” in International Conference on Interactive Theorem Proving. Springer, 2013, pp. 463–468.

[16] U. Berger, K. Miyamoto, H. Schwichtenberg, and M. Seisenberger, “Minlog - a tool for program extraction supporting algebras and coalgebras,” in International Conference on Algebra and Coalgebra in Computer Science. Springer, 2011, pp. 393–399.

[17] L. Zou, J. Lv, S. Wang, N. Zhan, T. Tang, L. Yuan, and Y. Liu, “Verifying chinese train control system under a combined scenario by theorem proving,” in Proc. Working Conf. Verified Software: Theories, Tools, and Experiments. Springer, 2013, pp. 262–280.

[18] F. Bobot, J.-C. Filliâtre, C. Marché, G. Melquiond, and A. Paskevich, “The why3 platform,” LRI, CNRS & Univ. Paris-Sud & INRIA Saclay, version 0.64 edition, 2011.

[19] D. Araiza-Ilán, K. Eder, and A. Richards, “Formal verification of control systems’ properties with theorem proving,” in Proc. 2014 UKACC Int. Conf. Control, 2014, pp. 244–249.

[20] ——, “Verification of control systems implemented in simulink with assertion checks and theorem proving: A case study,” in Proc. 2015 European Control Conf. (ECC), 2015, pp. 2670–2675.

[21] S. Gao, “Descriptive control theory: A proposal,” arXiv preprint arXiv:1409.3560, 2014.

[22] Coq, “The Coq proof assistant reference manual,” 2004, Version 8.0. [Online]. Available: http://coq.inria.fr

[23] J. Harrison, “HOL Light: A tutorial introduction,” in International Conference on Formal Methods in Computer-Aided Design. Springer, 1996, pp. 265–269.

[24] L. de Moura, S. Kong, J. Avigad, F. Van Doorn, and J. von Raumer, “The lean theorem prover (system description),” in International Conference on Automated Deduction. Springer, 2015, pp. 378–388.

[25] A. Anand and R. Knepper, “ROS-Coq: Robots powered by constructive reals,” in International Conference on Interactive Theorem Proving. Springer, 2015, pp. 34–50.

[26] R. O’Connor, “Certified exact transcendental real number computation in Coq,” in International Conference on Theorem Proving in Higher Order Logics. Springer, 2008, pp. 246–261.

[27] A. Platzer, “Logics of dynamical systems,” in Proc. 2012 IEEE/ACM Symp. Logic Computer Science (LICS). IEEE Computer Society, 2012, pp. 13–24.

[28] A. Tarski, “A decision method for elementary algebra and geometry,” in Quantifier Elimination and Cylindrical Algebraic Decomposition. Springer, 1998, pp. 24–84.

[29] B. Bohrer, V. Rahli, I. Vukotic, M. Völ, and A. Platzer, “Formally verified differential dynamic logic,” in Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, ser. CPP 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 208–221.

[30] H. Schwichtenberg, “Constructive analysis with witnesses,” Proof Technology and Computation. Natio Science Series, pp. 323–354, 2006.

[31] W. Hurewicz, Lectures on ordinary differential equations. Courier Corporation, 1990.

[32] H. Khalil, Nonlinear Systems. Prentice-Hall. 2nd edition, 1996, pp. 139–140.