A Formally Verified Motion Planner for Autonomous Vehicles

Albert Rizaldi[1⋆], Fabian Immler[2⋆⋆], Bastian Schürmann[1⋆], and Matthias Althoff[1]

[1] Institut für Informatik, Technische Universität München, Munich, Germany
{rizaldi, bastian.schuermann, althoff}@in.tum.de
[2] Computer Science Department, Carnegie Mellon University, Pittsburgh, USA
immler@cmu.edu

Abstract. Autonomous vehicles are safety-critical cyber-physical systems. To ensure their correctness, we use a proof assistant to prove safety properties deductively. This paper presents a formally verified motion planner based on manoeuvre automata in Isabelle/HOL. Two general properties which we ensure are numerical soundness (the absence of floating-point errors) and logical correctness (satisfying a plan specified in linear temporal logic). From these two properties, we obtain a motion planner whose correctness only depends on the validity of the models of the ego vehicle and its environment.

Keywords: motion primitives, manoeuvre automata, motion planning, theorem proving, linear temporal logic, reachability analysis, autonomous vehicles.

1 Introduction

Autonomous vehicles' planning and control are hard. Not only are they required to consider complex vehicle dynamics, but they must also deal with possibly unknown and dynamically changing environments. To tackle these complexities, most symbolic motion planners abstract continuous systems by discrete representations in either an *environment-driven* [6,12] or a *controller-driven* manner [16,14]. The former partitions the environment into cells, such as triangles or squares, while the latter partitions the controller into several primitives, such as `turn-left` or `turn-right`. Which discretisation is preferred for autonomous vehicles?

⋆ This work is partially supported by the DFG Graduiertenkolleg 1840 (PUMA) and the European Comission project UnCoVerCPS under grant number 643921.

⋆⋆ Supported by DFG Koselleck grant NI 491/16-1. Moreover, this material is based upon work supported by the Air Force Office of Scientific Research under grant number FA9550-18-1-0120. Any opinions, finding, and conclusion or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

Environment-driven discretisation is preferred when *1)* we have static, *a priori* known, and geometrically complex environments; or *2)* one has to handle expressive specifications, such as those expressed in Linear Temporal Logic (LTL). However, environment-driven discretisation usually works only for systems with simple dynamics [5]. On the contrary, controller-driven discretisation is preferred when we have dynamic, possibly unknown, and geometrically simple environments. Controllers designed with this discretisation can handle complex dynamics and navigate the environment by chaining a series of well-tested motion primitives [14]. However, specification languages for this discretisation in the literature, such as in [10], are very close to the implementation level; often, we want to specify *what* to achieve rather than *how* to achieve it.

Most vehicle models for autonomous vehicles are complex, making controller-driven discretisation a natural choice. In this work, we shall use manoeuvre automata-based motion planners [35], where each motion primitive is encoded as a state in our manoeuvre automaton. However, autonomous vehicles operate in dynamic and possibly unknown environments, where they could benefit from specification languages such as LTL — usually associated with environment-driven discretisation. This work aims to combine the advantages of both discretisation strategies by interpreting LTL over manoeuvre automata. To the best of our knowledge, this paper is the first work to tackle this challenge.

To prove correctness of our motion planner, we use the generic theorem prover Isabelle [28], as opposed to the specialised theorem prover KeYmaera X [15], which is tailored for proving the correctness of hybrid systems. This choice is motivated by eliminating numerical errors in computations with real numbers, which is largely ignored in motion planning [30]. Isabelle's libraries of approximation and affine arithmetic allow us to eliminate these errors [19,20]. Our contributions are as follows[3]:

- We provide a formally verified construction of manoeuvre automata (Sect. 3). More precisely, we interface Isabelle with MATLAB for solving optimisation problems and use the formalised algorithm for continuous reachability analysis [22]; both optimisation and reachability analysis are needed for constructing manoeuvre automata formally.
- We show how to eliminate numerical errors for functions involving real numbers (Sect. 4). To this end, we provide a verified translation between the representation of floating-points used by Isabelle and that of IEEE-754 used by MATLAB. Additionally, we extend the work in [22] to handle the trigonometric functions sine and cosine required in this work.
- We show how to plan autonomous vehicles' motions with temporal logic and manoeuvre automata (Sect. 5). More precisely, we interpret LTL over manoeuvre automata and formally perform satisfiability checking — as opposed to model checking — in order to find a sequence of manoeuvres which is guaranteed to satisfy a plan formalised in LTL.

[3] The formalisation is in `https://gitlab.lrz.de/ga96tej/manoeuvre-automata`.

2 Preliminaries

Notations used in this paper closely resemble Isabelle/HOL's syntax. Function application is always written in an uncurried form: instead of writing $f\ x\ y$ as in the λ-calculus, we always write $f(x, y)$. We write $t :: \tau$ to indicate that term t has type τ. Types used in this paper could either be a base type such as $\mathbb{R}$ for real numbers, or constructed via type constructors such as $\alpha\ \mathit{list}$ and $\alpha\ \mathit{set}$ for list of type α and set of type α, respectively. Another type constructor is the function type; a function from type α to β is written as (has the type of) $\alpha \Rightarrow \beta$. We use '$\Longrightarrow$' and '$\longrightarrow$' to denote deduction (inference) and implication, respectively. The set of all objects of type α is $\mathit{UNIV} :: \alpha$. Isabelle[4] also supports the **case** construct as in functional programming:

$$\textbf{case}\ t\ \textbf{of}\ \mathit{pat}_1 \Rightarrow t_1 \mid \ \ldots\ \mid \mathit{pat}_n \Rightarrow t_n\ .$$

One of the most frequently used data structures in this work are *affine forms*. An affine form A is defined by a sequence $(A_i)_{i\in\mathbb{N}}$ with only finitely many nonzero elements. We write A_i to refer to the i-th element of the affine form A. An affine form is interpreted for a valuation $\varepsilon : \mathbb{N} \to [-1, 1]$ as:

$$[\![A]\!]_\varepsilon := A_0 + \sum_i \varepsilon_i \cdot A_i\ .$$

We could also think of ε as a vector taken from an interval vector $[-\mathbf{1}; \mathbf{1}]$, where $\mathbf{1}$ is a vector of ones. One calls the terms ε_i noise symbols, A_0 the centre, and the remaining A_i generators. The idea is that noise symbols are shared between affine forms and that they are treated symbolically: the sum of two affine forms is given by the pointwise sum of their generators, and multiplication with a constant factor is also done componentwise:

$$\begin{aligned}[\![(A + B)]\!]_\varepsilon &:= (A_0 + B_0) + \sum_i \varepsilon_i \cdot (A_i + B_i)\ ,\\ [\![(k \cdot A)]\!]_\varepsilon &:= k \cdot A_0 + \sum_i \varepsilon_i \cdot (k \cdot A_i)\ .\end{aligned}$$

For $A_0, A_i :: \mathbb{R}^n$, the affine form A is a data structure to represent a specific type of set $\mathcal{Z} :: \mathbb{R}^n\ \mathit{set}$ (see the notation of type constructor for sets) called *zonotope* — a special class of polytopes. By defining a function *range* which represents all possible valuations of an affine form, the relationship between an affine form A and a zonotope $\mathcal{Z}$ is formalised as $\mathit{range}(A) = \mathcal{Z}$. Figure 1 provides the graphical illustration of the set of all points belonging to a zonotope.

If we represent an affine form concretely by a pair of its centre c and a list of its generators gs, then the *Minkowski sum* of two affine forms $A = (c, gs)$ and $A' = (c', gs')$ is defined as:

$$\begin{aligned}\mathit{msum}\,(A, A') &= (c + c',\ \mathit{msum\text{-}gens}\,(A, A'))\ ,\\ \mathit{msum\text{-}gens}\,(A, A') &= gs\ @\ gs'\ ,\end{aligned}$$

[4] From now on, 'Isabelle' refers to 'Isabelle/HOL' for simplicity.

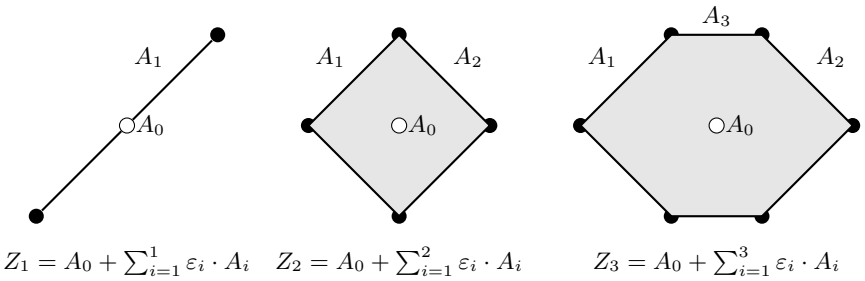

Fig. 1. Three zonotopes with $A_0 = (0, 0), A_1 = (1, 1), A_2 = (1, -1)$, and $A_3 = (1, 0)$. Black circles represent the extreme points of each zonotope.

where function @ denotes list concatenation. Figure 1 provides graphical illustrations of the Minkowski sum: $Z_2 = \mathit{msum}\,(Z_1, Z_2')$, $Z_3 = \mathit{msum}\,(Z_2, Z_3')$ where $Z_2' = \mathbf{0} + A_2$ and $Z_3' = \mathbf{0} + A_3$.

3 Constructing Manoeuvre Automata

A manoeuvre automaton (MA) [14] is an automaton whose states represent manoeuvres (motion primitives) which an autonomous system could execute. For helicopters, these could be standard manoeuvres such as `hover` and `land`, or more aggressive movements such as `hammerhead` and `loop`. For autonomous vehicles, these could be basic manoeuvres such as `turn-left` and `turn-right`, or more ambitious manoeuvres such as `hard-left` and `hard-right`. A transition between two states in an MA means that the system can execute those two manoeuvres successively.

Definition 1. *We define a manoeuvre automaton as a tuple* $MA = (M, \mathit{jump}, \mathit{ode})$ *where*

- M *is a predefined type for manoeuvre labels;*
- $\mathit{jump} :: (M \times M)\,\mathit{set}$ *is the transition relation between manoeuvre labels; and*
- $\mathit{ode}(m) :: \mathbb{R} \times \mathbb{R}^n \Rightarrow \mathbb{R}^n$ *is the corresponding ordinary differential equation (ODE) for manoeuvre* m*.*

If we assume that the $\mathit{ode}(m)$ has the general form of

$$\dot{x} = f(x, u_m) \ , \tag{1}$$

then the $\mathit{ode}(m)$ represents a fixed system model f — such as a point-mass or as a kinematic single-track model for autonomous vehicles [3] — with a fixed input trajectory u_m for manoeuvre m. For an initial state x_{init} and a final state x_{final}, a controller must choose a trajectory $u_m \in \mathcal{U}_m$ which steers x_{init} to x_{final}. We refrain from discussing the design of such controllers, as this work focusses more on the verification aspect; interested readers can consult the work in [35].

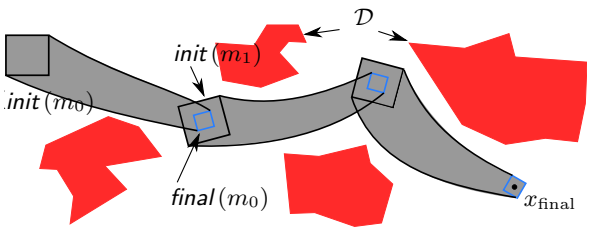

Fig. 2. Ensuring the safety of a path from an MA.

For safety verification purposes, it is paramount to compute the *reachable set* of a manoeuvre m — denoted by *reach* (m). This set represents the set of all states x which could be reached by the system f in (1) from an initial set denoted by *init* (m) with trajectory u_m. A manoeuvre m is safe with respect to a given unsafe set $\mathcal{D}$ if and only if *reach* (m) does not intersect with the unsafe set: *reach* $(m) \cap \mathcal{D} = \emptyset$ (see Fig. 2). A formally verified computation of reachable sets of continuous systems with the theorem prover Isabelle has been previously researched by one of the authors in [22] and we shall use it here.

How can we incorporate the reachable set of each manoeuvre to ensure the safety of a path? A safe path from an MA is a series of manoeuvres $m = m_0, m_1, \ldots, m_n$ which: *a)* respects the transition relation *jump* in Def. 1, i.e., $(m_i, m_{i+1}) \in$ *jump* for $0 \leq i < n$; *b)* ensures that the reachable set of each manoeuvre does not intersect with an unsafe set; and *c)* for every chain (m_i, m_{i+1}) in the series, the final set of m_i — denoted by *final* (m_i) — must be contained by the initial set of m_{i+1}, i.e., *final* $(m_i) \subseteq$ *init* (m_{i+1}) (see [35]). Figure 2 illustrates these requirements of ensuring the safety of a path from an MA.

The first technical challenge for formally constructing manoeuvre automata, as proposed in this work, is how to interface the controller design in [35] implemented in MATLAB and the reachability analysis in [22] formalised in Isabelle. Figure 3 illustrates how we interface Isabelle and MATLAB by using the C programming language as a *lingua franca*. Functions programmed in MATLAB are callable from C by using the MATLAB API. Isabelle, on the other hand, can call functions in Standard ML (SML) directly but not those in C. Fortunately, there is a Foreign Function Interface (FFI) between SML and C which enables us to call functions in C and, hence, MATLAB indirectly. Therefore, we need to provide the corresponding wrapper for each MATLAB function required by Isabelle at the SML and C levels.

The second technical challenge is to bridge the different types of floating-point representation between Isabelle and MATLAB. Isabelle uses arbitrary precision floating-point numbers ($m \cdot 2^e$ for potentially unbounded $m, e \in \mathbb{Z}$) and MATLAB uses IEEE-754 floating point-numbers (with fixed precision). How to obtain a formally correct conversion between arbitrary precision floating-point numbers (as used in Isabelle/HOL) and IEEE floating-point numbers (as used in MATLAB) is discussed in the next section.

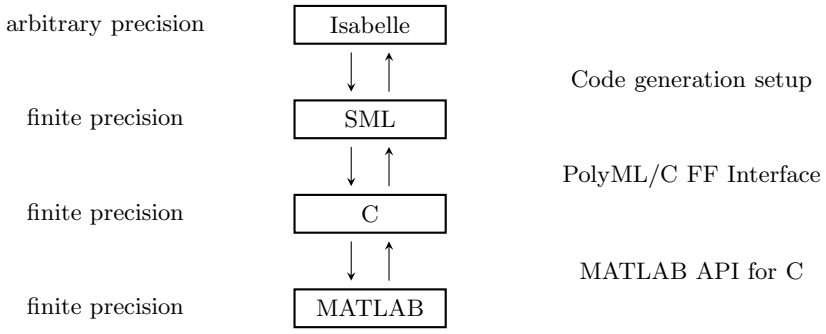

Fig. 3. Block diagram for interfacing Isabelle and MATLAB.

4 Affine Arithmetic and Floating-Point Numbers

This section considers rounding errors when using finite precision floating-point numbers to ensure soundness of our proofs. To achieve this, we use rigorous numerics, which encloses real numbers by sets. This means, for example, the function $\times$ :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* is "lifted" to a new function $\otimes$ :: *real set* $\Rightarrow$ *real set* $\Rightarrow$ *real set* with the correctness theorem $\forall x, y.\ \ x \in X \wedge y \in Y \longrightarrow x \times y \in X \otimes Y$. The first problem entailed by this decision is how to choose the proper data structure to represent the abstract type *real set*. Following the decision made in our previous work [20], we use affine arithmetic [13] for this purpose. There are other approaches such as intervals [27] and Taylor models [7] whose discussion is out of the scope of this paper. The second problem is how to approximate functions operating on reals with functions operating on sets correctly. Previous work in [20] has covered affine approximation of arithmetic functions such as addition, multiplication, subtraction, and division, but not trigonometric functions. For this particular work, we need affine approximations of trigonometric functions such as sine and cosine occuring in model f in (1) (the specific model can be found in (5), Sect. 5).

4.1 Affine approximation of trigonometric functions

To simplify formal proofs, modularity and abstraction are important. As a basis for all operations that follow, we use a generic linear operation that involves round-off operations and also adds a noise symbol for further uncertainties (this is also discussed by Stolfi and de Figueredo [13]). The idea is to define a generic linear operation *affine-unop*$(\alpha, \beta, \delta, X)$ that encloses the linear function $x \mapsto \alpha \cdot x + \beta$ with an uncertainty of δ for every valuation $\varepsilon \in \mathbb{N} \to [-1, 1]$:

$$|\alpha \cdot [\![X]\!]_\varepsilon + \beta - [\![\textit{affine-unop}(\alpha, \beta, \delta, X)]\!]_\varepsilon| \leq \delta \ .$$

The motivation behind *affine-unop*$(\alpha, \beta, \delta, X)$ is that $\alpha \cdot x + \beta$ approximates some (possibly nonlinear) function f up to an error δ, i.e.,

$$|f(x) - (\alpha \cdot x + \beta)| \leq \delta \ , \tag{2}$$

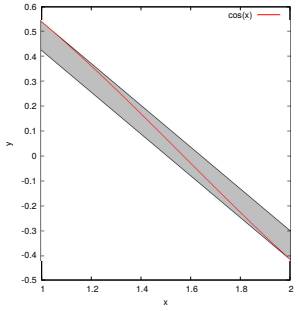

Fig. 4. Min-range approximation of $\cos[1, 2]$.

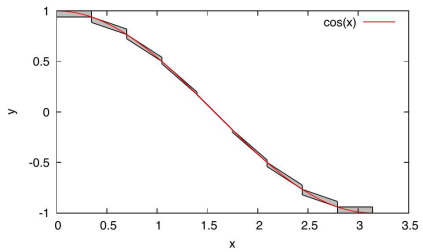

Fig. 5. Min-range approximations (nine subdivisions) of $\cos[0, \pi]$

up to a certain interval $x \in [l, u]$. There are various degrees of freedom for linearising a non-linear function f such that (2) holds. In this work, we use the *min-range approximation* [13,34] for the sake of ease of implementation and verification; other techniques such as interval approximation [13,34] and first-order Tchebychev approximation [13,34] exist.

The idea behind the min-range approximation is to maximize the slope of the enclosure while fixing the range of the approximation. Consider Fig. 4, which illustrates a min-range approximation of cosine on the interval $[1, 2]$; it does not exceed the interval $[\cos(2), \cos(1)]$. Any smaller slope would be just as safe, but the slope could not be chosen to be larger. The following theorem guides us to find suitable values of α, β, and δ.

Proposition 1 (Min-range approximation).

$$\forall x \in [l, u].\ |f(x) - (\alpha \cdot x + \beta)| \leq \delta$$

if the following conditions are satisfied:

1. $\forall y \in [l, u].\ \alpha \leq f'(y)$;
2. $\delta \geq \frac{f(u) - f(l) - \alpha \cdot (u - l)}{2} + |(f(l) + f(u) - \alpha \cdot (l + u))/2 - \beta|$.

Parameter α needs to be a lower bound on the derivative while parameters β and δ need to be chosen such that they account for the error of the linear function centred between $f(u)$ and $f(l)$ as well as for the error that β makes with respect to the centre (the second summand on the right of the inequality bounding δ). This is a slight generalisation of what is demanded in the literature [13,34], where one assumes a convex function f. This ensures that the derivative f' attains its maximum at one of the endpoints of the interval. Something that is not mentioned in the statement of the lemma should be noted: the approximation using α, β is close to the optimal approximation only if the function f is increasing on $[l, u]$ (otherwise the theorem still holds, but δ is unnecessarily large). However, a similar approximation lemma can be easily obtained for the case when f is decreasing.

Trigonometric Functions. The trigonometric functions sine and cosine pose the problem that they are not monotonic. This can be alleviated in two steps (similar to the treatment of periodic functions in [2]). The first step, range reduction, exploits periodicity to reduce the argument to the range $[0, 2\pi]$. Range reduction (shifting the argument x by $-2\pi \cdot \lfloor \frac{x}{2\pi} \rfloor$) is computed using interval arithmetic. The second step is a case distinction if the argument is contained in the decreasing part $[0, \pi]$ or the monotone part $[\pi, 2\pi]$ (for cosine). It is possible that this distinction cannot be decided (if e.g., the argument interval straddles π), but then the only valid min-range approximation is the interval approximation (with 1 as upper bound). A series of such computed min-range approximations is shown in Fig. 5.

4.2 From Isabelle's to IEEE-754's floating-point representation

Affine arithmetic in Isabelle is implemented with arbitrary-precision floating-point numbers, hereafter denoted by float_∞. Software floating point numbers (the formalisation in Isabelle originates from Obua's work [29]) are the subset of real numbers that can be represented by two arbitrary-precision integers: mantissa (or significand) m and exponent e. Mantissa and exponent together represent the real number $m \cdot 2^e$:

$$\mathit{float}_\infty := \{m \cdot 2^e \mid m, e \in \mathbb{Z}\} \ .$$

Arbitrary-precision floating-point numbers are convenient for formal reasoning because arithmetic operations can be carried out without round-off errors. For efficiency, however, we do use explicit round-off operations overapproximately to reduce the size of the mantissa. The explicit separation of operation and rounding helps keeping the formalisation modular.

The representation used by MATLAB is IEEE-754 floating-point numbers. A specification of floating-point numbers (with a fixed precision of 52 bits for the mantissa and 11 bits for the exponent of double-precision floating point numbers) according to the IEEE-754 standard was formalized in Isabelle by Yu [37].[5] We denote IEEE-754 floating-point numbers with $\mathit{float}_{\mathit{ieee}}$. They are represented by triples $(s, e, f) \in \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ to represent sign s, exponent e, and fraction f. There are special representations for special values like infinity or NaN (Not-a-Number); everything else represents *finite* numbers. A predicate $\mathit{is\text{-}finite}$ encodes whether a triple represents a finite number. Finite IEEE floating point numbers can be normal ($e \neq 0$) or denormal ($e = 0$) and are interpreted as a real number differently using $\mathit{to\text{-}real}$:

$$\mathit{to\text{-}real}(s, e, f) = \begin{cases} (-1)^s \cdot 2^{-1022} \cdot (f \cdot 2^{-52}) & \text{if } e = 0 \ , \\ (-1)^s \cdot 2^{e-1023} \cdot (1 + f \cdot 2^{-52}) & \text{if } e \neq 0 \ . \end{cases}$$

[5] Yu's formalisation was inspired by Harrison's [18] extensive formalisation in HOL Light. More work on floating-point numbers in theorem provers has been done in the comprehensive formalisation of floating-point numbers by Boldo and Melquiond [9] in Coq as well as early efforts in ACL2 [26].

We provide functions *of-ieee* and *to-ieee* which convert between a subset of arbitrary-precision floating-point numbers and IEEE floating-point numbers. The bijection is guarded by *is-valid* (to ensure that the arbitrary precision floating-point number is actually of suitable finite precision) and *is-finite* (to exclude special values).

$$\textit{is-finite}(s, e, f) \implies \textit{of-ieee}(s, e, f) = \textit{to-real}(s, e, f) \ ,$$

$$\textit{is-valid}(m \cdot 2^e) \implies \textit{to-real}(\textit{to-ieee}(m, e)) = m \cdot 2^e \ .$$

We implemented this (based on work by Fabian Hellauer, which we gratefully acknowledge here) using the IEEE-754 formalisation in the archive of formal proofs [38]. Note that since Isabelle's floating-point representation can have arbitrary precision, we have to ensure that the floating-point numbers used in Isabelle's theories are guaranteed to have at most 53-bit precision (i.e., *is-valid* holds) to be able to pass them down to SML, C, and MATLAB.

5 Motion Planning with Manoeuvre Automata

5.1 Interpreting LTL over manoeuvre automata

Definition 2 (Linear Temporal Logic for MA). *If AP is the type for all atomic propositions, then we can create a new compound data type*

$$\textbf{datatype}\ \mathit{atom} \ = \ AP^+ \mid AP^- \ ,$$

where we label an atomic proposition with either a positive or negative sign. The syntax of LTL for manoeuvre automata is defined by the following grammar:

$$\phi \ ::= \ \textit{true} \mid \pi \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \mathsf{X}\,\phi \mid \phi_1 \,\mathsf{U}\, \phi_2 \ , \tag{3}$$

where π :: atom. Constant false*, logical operators disjunction and implication, and temporal operators* F *and* G *are defined as usual [11].*

Atomic propositions in path planning with LTL are used to represent objects of interest. For example, atomic propositions in our work could be defined as follows:

$$\textbf{datatype}\ AP \ = \textit{left-boundary} \mid \textit{right-boundary} \mid \textit{obstacle} \mid \textit{goal}$$

Definition 3 (Semantics of LTL for MA over finite-length traces). *Suppose that the state space for the model* ode(m) *in Def. 1 is of type $\mathbb{R}^n$, and there is an interpretation function $[\![\_]\!] :: AP \Rightarrow \mathbb{R}^n$ set. Additionally, for a finite sequence of sets $\sigma = \mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_n$, we denote the j-th suffix of σ by $\sigma[j..] := \mathcal{A}_j, \ldots, \mathcal{A}_n$ for $0 \leq j \leq n$. We can define a semantics of LTL for MA over a finite sequence*

of sets $\sigma = \mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_n$*, where* $\mathcal{A}_i :: \mathbb{R}^n$ *set for* $0 \leq i \leq n$*, as follows:*

$$
\begin{aligned}
&\sigma \models \mathit{true} \\
&\sigma \models \pi^+ && \iff && \mathcal{A}_0 \subseteq [\![\pi]\!] \\
&\sigma \models \pi^- && \iff && \mathcal{A}_0 \cap [\![\pi]\!] = \emptyset \\
&\sigma \models \neg\phi && \iff && \sigma \not\models \phi \\
&\sigma \models \phi_1 \wedge \phi_2 && \iff && \sigma \models \phi_1 \text{ and } \sigma \models \phi_2 \\
&\sigma \models \mathsf{X}\,\phi && \iff && \text{if } \sigma[1..] \text{is defined then } \sigma[1..] \models \phi \\
&\sigma \models \phi_1 \,\mathsf{U}\, \phi_2 && \iff && \exists j.\ \sigma[j..] \models \phi_2 \ \wedge\ \forall i. 0 \leq i < j \longrightarrow \sigma[i..] \models \phi_1 \ .
\end{aligned}
$$

Comparison with standard LTL. The differences with standard LTL's syntax and semantics primarily lie in the additional sign for each atomic proposition and their denotations. To illustrate these differences more concretely, consider the formalisation of reach-avoid plans in standard LTL. Fainekos et al. [11] formalised these plans with $\neg\mathit{obstacle}\ \mathsf{U}\ \mathit{goal}$; this is fine if we interpret LTL over a single trajectory. For an interpretation over a set of trajectories, we can lift the denotation for atomic propositions used by Fainekos et al. [11] into $\sigma \models \pi :: AP \iff \mathcal{A}_0 \subseteq [\![\pi]\!]$ (see [33]). This denotation implies $\sigma \models \neg\mathit{obstacle}$ if and only if $\mathcal{A}_0 \not\subseteq [\![\mathit{obstacle}]\!]$ and, if we assume further that $\mathcal{A}_0 \cap [\![\mathit{obstacle}]\!] \neq \emptyset$, then there could be a trajectory which visits the obstacle before it reaches the goal. This means the safety of σ cannot be guaranteed anymore.

The syntax and semantics in Defs. 2 and 3 provide a solution to this problem. Each atomic proposition can be labelled either with a positive or negative sign, and the root cause of the unsafety in the previous argument is due to the additional assumption $\mathcal{A}_0 \cap [\![\mathit{obstacle}]\!] \neq \emptyset$. The semantics solves this problem by enforcing that all negatively labelled atomic propositions have the denotation that all trajectories in $\mathcal{A}_0$ cannot be located at $[\![\pi]\!]$, i.e., $\mathcal{A}_0 \cap [\![\pi]\!] = \emptyset$. Positively labelled atomic propositions, meanwhile, have the obvious denotation that all trajectories in the initial set $\mathcal{A}_0$ must also be located inside $[\![\pi]\!]$, i.e., $\mathcal{A}_0 \subseteq [\![\pi]\!]$. In case $\mathcal{A}_0 \not\subseteq [\![\pi]\!]$ and $\mathcal{A}_0 \cap [\![\pi]\!] \neq \emptyset$, there might be a trajectory which always stays in $[\![\pi]\!]$ or lies outside of $[\![\pi]\!]$ completely, but this should not justify $\sigma \models \pi^+$ or $\sigma \models \pi^-$ because we choose soundness over completeness.

Checking zonotope inclusion and intersection freedom. The semantics in Def. 3 does not stipulate any concrete type of sets. To demonstrate our approach, we use zonotopes in this work to check $\sigma \models \pi^+$ (using an inclusion check) and $\sigma \models \pi^-$ (checking for intersection freedom) in $\mathbb{R}^2$ since higher dimensions are not required in this work. We define the function $\mathit{zono\text{-}contain2D}(\mathit{prec}, Z, Z')$ to check whether zonotope $\mathit{range}(Z)$ is a subset of zonotope $\mathit{range}(Z')$. This is performed[6] by first enumerating all extreme points of zonotope $\mathit{range}(Z)$ — via the function $\mathit{extreme\text{-}pts}(Z)$ — initially (see Fig. 1) and then checking whether each of these extreme points belongs to the zonotope $\mathit{range}(Z_2)$ as partially done in [21].

[6] For high-dimensional zonotopes, please consult the technique described in CORA [1].

Theorem 1. *By defining* $\mathit{zono\text{-}contain2D}(\mathit{prec}, Z, Z')$ *as:*

$$\begin{aligned}\mathit{zono\text{-}contain2D}(\mathit{prec}, Z, Z') \;:=\; &\textbf{case}\ \mathit{extreme\text{-}pts}(Z)\ \textbf{of}\\ &[]\ \Rightarrow\ Z_0 \in_{\mathit{zono}} Z'\\ &|\,\mathit{ps}\ \Rightarrow\ \forall p.\ p \in \mathit{set}(\mathit{ps}) \longrightarrow p \in_{\mathit{zono}} Z'\ ,\end{aligned}$$

we have the correctness condition:

$$\mathit{zono\text{-}contain2D}(\mathit{prec}, Z, Z') \implies \mathit{range}(Z) \subseteq \mathit{range}(Z')\ ,$$

for any precision prec *and any two zonotopes* Z, Z' *of type* $\mathbb{R}^2$*.*

We also define the function $\mathit{collision\text{-}freedom2D}(\mathit{prec}, Z, Z')$ to check whether zonotope $\mathit{range}(Z)$ does *not* intersect with $\mathit{range}Z'$ based on [17] which we proved formally in Isabelle too.

Theorem 2. *Suppose that affine forms* Z *and* Z' *have the centres of* Z_0 *and* Z_0'*, respectively, and* $Z_0' - Z_1'$ *denotes the vector difference of* Z_0' *and* Z_1'*, then*

$$\underbrace{Z_0' - Z_1' \notin \mathit{range}(\mathit{msum\text{-}gens}(Z, Z'))}_{:=\ \mathit{collision\text{-}freedom2D}(\mathit{prec}, Z, Z')} \implies \mathit{range}(Z) \cap \mathit{range}(Z') = \emptyset\ .$$

Note that the two theorems above take the precision $\mathit{prec} :: \mathbb{N}$ into account to ensure numerical soundness.

5.2 Satisfiability checking of LTL over manoeuvre automata

The problem of finding a path in MA which satisfies a plan formalised in LTL can now be stated formally as satisfiability checking.

Definition 4 (Satisfiability checking). *An LTL formula* ϕ *is satisfiable with respect to a manoeuvre automaton* $MA = (M, \mathit{jump}, \mathit{ode})$ *if there is a path* $\tau = m_0, m_1, \ldots m_{n-1}$ *such that* $m_i :: M$ *for all* $0 \leq i < n$ *and*

$$\mathit{reach}(m_0), \mathit{reach}(m_1), \ldots, \mathit{reach}(m_{n-1}) \models \phi\ .$$

Satisfiability checking is a search problem and since *1)* time efficiency is paramount, and *2)* a path satisfying a plan usually has a finite duration, we use a depth-limited search strategy for satisfiability checking. Since each manoeuvre lasts for 1 s and a sensible duration for a plan is supposed to be less than 10 s, the maximum depth is set to be 10. Note that the search strategy can be improved further by using an informed search strategy. However, since our main focus is correctness, we choose a simpler yet sufficient depth-limited search strategy for satisfiability checking.

As an example, we construct an intentionally simple, formally verified manoeuvre automaton with three motion primitives which last for 1 s each:

$$\textbf{datatype}\ M\ =\ \mathit{go\text{-}straight} \mid \mathit{turn\text{-}left} \mid \mathit{turn\text{-}right}\ , \tag{4}$$

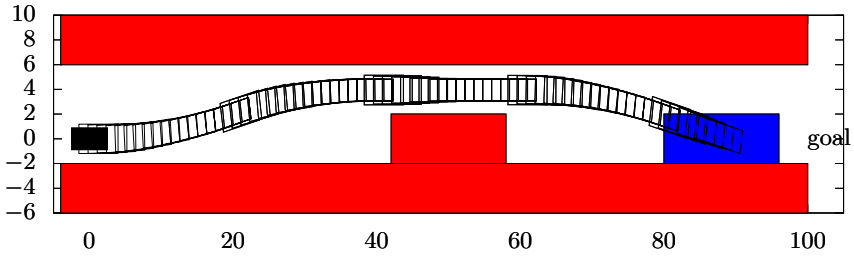

Fig. 6. Example of reach-avoid scenario. The vehicle is represented as the solid black rectangle. Red-coloured rectangles are the objects the vehicle has to avoid. The blue-coloured rectangle is the area which the vehicle has to reach eventually.

where any two manoeuvres can be composed, i.e., $\mathit{jump} := \mathit{UNIV} :: M \times M$. Note that the duration for each motion primitive need not to be the same; some primitives could last for, e.g., 0.1 s and others could last for 5 s. We use the following kinematic model of autonomous vehicles:

$$\dot{v} = a; \quad \dot{\Psi} = b; \quad \dot{x} = v \cdot \cos(\Psi); \quad \dot{y} = v \cdot \sin(\Psi) \ . \tag{5}$$

State variables v and Ψ are speed and orientation, respectively, while x and y are the positions in Cartesian coordinates. Inputs to the system are a and b, which denote acceleration and normalised steering angle, respectively. The initial set $\mathit{init}\,(m)$ is set to be the same for all manoeuvres:

$$[19.8; 20.2]\,\mathrm{m\,s^{-1}} \times [-0.02; 0.02]\,\mathrm{rad} \times [-0.2; 0.2]\,\mathrm{m} \times [-0.2; 0.2]\,\mathrm{m} \ .$$

Meanwhile, the final states are (20, 0, 20, 0) for *go-straight*, (20.2, 0.2, 19.87, 0.2) for *turn-left*, and (20.2, −0.2, 19.87, −0.2) for *turn-right*. We use the controller design in [35] to obtain a set of trajectories for each manoeuvre and use the verified implementation in [22] to compute the reachable sets for each time.

According to the third requirement to ensure the safety of a path from an MA in Sect. 3, we must ensure the enclosure property $\mathit{final}\,(m_1) \subseteq \mathit{init}\,(m_2)$ holds for any two manoeuvres $(m_1, m_2) \in \mathit{jump}$. However, the concrete numbers above show that $\mathit{final}\,(\mathit{go\text{-}straight}) \not\subseteq \mathit{init}\,(\mathit{go\text{-}straight})$. This does not mean we cannot compose two *go-straight* primitives. To achieve this, the initial set can be shifted in position and orientation — due to position and orientation invariance in (5).

We consider the reach-avoid scenario for autonomous vehicles (Fig. 6) for motion planning. The road is divided into two four-meter-wide lanes and bounded by left and right boundaries. There is also a 16 m × 4 m-rectangle located at $(50, 0)$ which serves as an obstacle in our scenario. The autonomous vehicle has the length and width of 5 m and 1.75 m, respectively. It is located initially at $(0, 0)$ and must reach the goal represented by a 16 m × 4 m rectangle which is located at $(80, 0)$.

The reach-avoid plan is formalised with the following LTL formula:

$$\phi \ := \ (\mathit{left\text{-}boundary}^- \ \wedge \ \mathit{right\text{-}boundary}^- \ \wedge \ \mathit{obstacle}^-) \ \mathsf{U} \ \mathit{goal}^+ \ .$$

After performing satisfiability checking, the search returned the following plan as shown in Fig. 6:

$$\tau \;:=\; \textit{turn-left}\;,\; \textit{turn-right}\;,\; \textit{go-straight}\;,\; \textit{turn-right}\;,\; \textit{go-straight}\;.$$

Regarding the search strategy for satisfiability checking, there are two properties we proved: termination and soundness. The former is proved with the aid of the function package in Isabelle [23] by specifying a measure function which decreases after each recursive call. Meanwhile, the latter is ensured due to the following two facts: *1)* we use the formalised LTL monitoring function from our previous work [32] to check whether current nodes satisfy the LTL formula, and *2)* we interpret each atomic proposition over-approximatively either due to inherent uncertainty or numerical round-offs.

Two remarks worth mentioning here. Firstly, note that the main scientific dimension considered in this work is the correctness of a motion planner achieved with the aid of a theorem prover. Hence, we prioritise correctness over other dimensions such as coverage, efficiency, and scalability. The example provided in this section should be perceived as an evidence that the formalisation in Isabelle is implementable (code generation); this section by no means is an evaluation of the coverage of our framework which we plan to do in future with other scenarios in [3]. Secondly, readers might question the fidelity of the model in (5). However, Schürmann et al. [36] have provided a framework such that a relatively simple model like ours with added uncertainties from a higher fidelty model or a real vehicle could adequately ensure the safety of a plan in a real vehicle.

6 Related Work and Conclusions

Fainekos et al. [12] and Plaku et al. [30] use satisfiability checking (or falsification) of temporal logic for finding a path which satisfies a plan formalised in (a fragment of) LTL. Fainekos et al. [12] expanded and contracted objects which must be avoided and reached, respectively, in order to have a robust interpretation of LTL. Plaku et al. [30] ignore the issue of numerical soundness when checking whether a path satisfies an LTL formula. Our approach, meanwhile, uses sets (zonotopes) as the main data structure which means we can handle robustness and numerical soundness simultaneously.

Interpreting LTL formulae over a set of trajectories has also been studied by Roehm et al. [33]. The difference between our semantics is in the way we treat the negation operator. In their work, the negation operator is allowed for formulae without any temporal operators only. Our approach, however, does not have this restriction — hence ours is more expressive — but it comes with an additional requirement of labelling each atomic proposition with a positive or negative sign.

Mitsch et al. [25] use the theorem prover KeYmaera X [15] to prove safety properties of autonomous vehicles. The main difference to our work is the approach to formal reasoning. Theirs is *proof-theoretic*: *a)* they specify the physical

model of autonomous vehicles with hybrid programs and the property with differential dynamic logic [31]; then *b)* they use the proof system's inference rules to deduce that the hybrid program indeed satisfies the specified property. As pointed out by Anand and Knepper [4], KeYmaera X does not consider the possibility of round-off errors in floating-point numbers. This issue has been addressed by Bohrer et al. [8] where they introduce a framework called VeriPhy.

Our approach is *model-theoretic*: *1)* we model autonomous vehicles with manoeuvre automata in which each state (manoeuvre) is assigned with reachable sets of the physical behaviour; *2)* we specify the property in a modified LTL which takes the reachable sets into account; and *3)* we enumerate all possible paths in the manoeuvre automaton and find a path which satisfies the property according to the predefined semantics of the modified LTL. The role of the Isabelle theorem prover in our work is to prove that each step is implemented correctly. Compared to VeriPhy, we use affine arithmetic and VeriPhy uses interval arithmetic — a special case of affine arithmetic. However, our approach needs to trust the code generation setup provided by Isabelle, whereas VeriPhy uses a sound compilation technique to generate code in CakeML [24].

Anand and Knepper [4] use the Coq theorem prover to implement a framework to specify the physical model and controller of robots for the Robot Operating System (ROS). Compared to our formalisation, theirs is closer to the implementation level; ours assumes that the optimal controller can be implemented correctly in the hardware. However, their implementation assumes that the high-level plan is given, whereas we derive a high-level plan and a low-level controller. Both works guarantee numerical soundness, but with a different technique; theirs uses constructive reals, whereas we use floating-point numbers.

Belta et al. [5] have outlined that the challenge for symbolic motion planning and control is to tie the *top-down approaches*, which use temporal logic on rather abstract models, and *bottom-up approaches*, whose aim is to construct manoeuvre automata effectively for formal analysis. We solve this challenge by adapting the syntax and semantics of LTL for manoeuvre automata. The main finding for this work is that *reachability analysis* is the key ingredient to solve this problem. It allows us to compute the reachable sets of each motion primitive and subsequently to define the satisfaction relation of motion primitives with formulae in LTL. We also address the challenge of formal verification of cyber-physical systems, where numerical soundness is largely ignored. By using a generic theorem prover such as Isabelle, we can guarantee both mathematical correctness and numerical soundness.

References

1. Althoff, M.: An introduction to CORA 2015. In: Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems (2015)
2. Althoff, M., Grebenyuk, D.: Implementation of interval arithmetic in CORA 2016. In: Proc. of the 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems. pp. 91–105 (2016)

3. Althoff, M., Koschi, M., Manzinger, S.: CommonRoad: Composable benchmarks for motion planning on roads. In: Proc. of the IEEE Intelligent Vehicles Symposium. pp. 719–726 (2017)
4. Anand, A., Knepper, R.A.: Roscoq: Robots powered by constructive reals. In: Proc. of the 6th International Conference on Interactive Theorem Proving. pp. 34–50 (2015)
5. Belta, C., Bicchi, A., Egerstedt, M., Frazzoli, E., Klavins, E., Pappas, G.J.: Symbolic planning and control of robot motion [grand challenges of robotics]. IEEE Robotics Automation Magazine 14(1), 61–70 (2007)
6. Belta, C., Isler, V., Pappas, G.J.: Discrete abstractions for robot motion planning and control in polygonal environments. IEEE Transactions on Robotics 21(5), 864–874 (2005)
7. 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)
8. Bohrer, B., Tan, Y.K., Mitsch, S., Myreen, M., Platzer, A.: Veriphy: Verified controller executables from verified cyber-physical system models. In: Proc. of the ACM SIGPLAN Conference on Programming Language Design and Implementation (2018), *(Accepted)*
9. Boldo, S., Melquiond, G.: Flocq: A unified library for proving floating-point algorithms in Coq. In: Proc. of the IEEE Computer Arithmetic Symposium. pp. 243–252 (2011)
10. Egerstedt, M.B., Brockett, R.W.: Feedback can reduce the specification complexity of motor programs. IEEE Transactions on Automatic Control 48(2), 213–223 (2003)
11. Fainekos, G.E., Kress-Gazit, H., Pappas, G.J.: Temporal logic motion planning for mobile robots. In: Proc. of the IEEE International Conference on Robotics and Automation. pp. 2020–2025 (2005)
12. Fainekos, G.E., Girard, A., Kress-Gazit, H., Pappas, G.J.: Temporal logic motion planning for dynamic robots. Automatica 45(2), 343 – 352 (2009)
13. de Figueiredo, L., Stolfi, J.: Affine arithmetic: Concepts and applications. Numerical Algorithms 37(1-4), 147–158 (2004)
14. Frazzoli, E., Dahleh, M.A., Feron, E.: Maneuver-based motion planning for nonlinear systems with symmetries. IEEE Transactions on Robotics 21(6), 1077–1091 (2005)
15. Fulton, N., Mitsch, S., Quesel, J.D., Völp, M., Platzer, A.: KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In: Proc. of the International Conference on Automated Deduction. pp. 527–538 (2015)
16. Gavrilets, V., Mettler, B., Feron, E.: Human-inspired control logic for automated maneuvering of miniature helicopter. Journal of Guidance Control and Dynamics 27(5), 752–759 (2004)
17. Guibas, L.J., Nguyen, A., Zhang, L.: Zonotopes as bounding volumes. In: Proc. of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 803–812 (2003)
18. Harrison, J.: Floating-point verification using theorem proving. In: Proc. of the 6th International Conference on Formal Methods for the Design of Computer, Communication, and Software Systems. pp. 211–242 (2006)
19. Hölzl, J.: Proving inequalities over reals with computation in Isabelle/HOL. In: Proc. of the ACM International Workshop on Programming Languages for Mechanized Mathematics Systems. pp. 38–45 (2009)

20. Immler, F.: Formally verified computation of enclosures of solutions of ordinary differential equations. In: Proc. of the 6th International Symposium of NASA Formal Methods. pp. 113–127 (2014)
21. Immler, F.: A verified algorithm for geometric zonotope/hyperplane intersection. In: Proc. of International Conference on Certified Programs and Proofs. pp. 129–136 (2015)
22. Immler, F.: Verified reachability analysis of continuous systems. In: Proc. of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 37–51 (2015)
23. Krauss, A.: Automating recursive definitions and termination proofs in higher-order logic. Ph.D. thesis, Technical University Munich (2009)
24. Kumar, R., Myreen, M.O., Norrish, M., Owens, S.: Cakeml: A verified implementation of ml. In: Proc. of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 179–191 (2014)
25. Mitsch, S., Ghorbal, K., Vogelbacher, D., Platzer, A.: Formal verification of obstacle avoidance and navigation of ground robots. The International Journal of Robotics Research 36(12), 1312–1340 (2017)
26. Moore, J.S., Lynch, T., Kaufmann, M.: A mechanically checked proof of the correctness of the kernel of the AMD5K86 floating-point division algorithm. IEEE Transactions on Computers 47(9), 913–926 (1996)
27. Moore, R.E.: Methods and applications of interval analysis. SIAM (1979)
28. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, vol. 2283. Springer (2002)
29. Obua, S.: Flyspeck II: The Basic Linear Programs. Ph.D. thesis, Technische Universität München, München (2008)
30. Plaku, E., Kavraki, L.E., Vardi, M.Y.: Falsification of LTL safety properties in hybrid systems. International Journal on Software Tools for Technology Transfer 15(4), 305–320 (2013)
31. Platzer, A.: Differential dynamic logic for hybrid systems. Journal of Automated Reasoning 41(2), 143–189 (2008)
32. Rizaldi, A., Keinholz, J., Huber, M., Feldle, J., Immler, F., Althoff, M., Hilgendorf, E., Nipkow, T.: Formalising traffic rules for autonomous vehicles involving multiple lanes in Isabelle/HOL. In: Proc. of the 13th International Conference on integrated Formal Methods. pp. 50 – 66 (2017)
33. Roehm, H., Oehlerking, J., Heinz, T., Althoff, M.: STL model checking of continuous and hybrid systems. In: Proc. of 14th International Symposium on Automated Technology for Verification and Analysis. pp. 412–427 (2016)
34. Rump, S.M., Kashiwagi, M.: Implementation and improvements of affine arithmetic. Nonlinear Theory and Its Applications, IEICE 6(3), 341–359 (2015)
35. Schürmann, B., Althoff, M.: Convex interpolation control with formal guarantees for disturbed and constrained nonlinear systems. In: Proc. Hybrid Systems: Computation and Control. pp. 121–130 (2017)
36. Schürmann, B., Heß, D., Eilbrecht, J., Stursberg, O., Köster, F., Althoff, M.: Ensuring drivability of planned motions using formal methods. In: Proc. of the Intelligent Transportation Systems Conference. pp. 1661–1668 (2017)
37. Yu, L.: A formal model of ieee floating point arithmetic. Archive of Formal Proofs (Jul 2013), `http://isa-afp.org/entries/IEEE_Floating_Point.html`, Formal proof development
38. Yu, L.: A formal model of ieee floating point arithmetic. Archive of Formal Proofs (Sep 2017), `http://devel.isa-afp.org/entries/IEEE_Floating_Point.html`, Formal proof development