Explaining the Behaviour of Hybrid Systems with PDDL+ Planning

Diego Aineto[1] , **Eva Onaindia**[1] , **Miquel Ramírez**[2] , **Enrico Scala**[3] and **Ivan Serina**[3]
[1]VRAIN, Universitat Politècnica de València
[2]University of Melbourne
[3]Università degli Studi di Brescia
{dieaigar,onaindia}@upv.es, miquel.ramirez@unimelb.edu.au, {enrico.scala,ivan.serina}@unibs.it

Abstract

The aim of this work is to explain the observed behaviour of a *hybrid system* (HS). The explanation problem is cast as finding a trajectory of the HS that matches some observations. By using the formalism of *hybrid automata* (HA), we characterize the explanations as the language of a network of HA that comprises one automaton for the HS and another one for the observations, thus restricting the behaviour of the HS exclusively to trajectories consistent with the observations. We observe that this problem corresponds to a reachability problem in model-checking, but that state-of-the-art model checkers struggle to find concrete trajectories. To overcome this issue we provide a formal mapping from HA to PDDL+ and rely on off-the-shelf automated planners. An experimental analysis over domains with piece-wise constant, linear and nonlinear dynamics reveals that the proposed PDDL+ approach is much more efficient than solving directly the explanation problem with model-checking solvers.

1 Introduction

A *hybrid system* (HS) is a dynamical system that exhibits a discrete and continuous behaviour, and captures the control of continuously evolving physical activities typical in automated manufacturing, chemical engineering and robotics systems. A *hybrid trajectory* is a particular execution of the HS that interleaves discrete and continuous behaviour. We aim to solve the problem of finding an explanation, i.e., a hybrid trajectory, for some observations of an HS. We will call this the HS explanation (HSE hereinafter) problem. To this end, we build on the formalism of *hybrid automata* (HA) to formulate the problem and on *planning* technology to solve it.

The formalism of HA [Henzinger, 2000] is extensively used in Model Checking (MC) of hybrid systems, specifically for *verification of safety properties*. The safety verification problem is traditionally formulated as a reachability analysis aimed at certifying that a trajectory that violates the property does not exist, thereby a correspondence between this problem and our HSE problem can be found. A common approach is to compute a conservative over-approximation of the reachable state space that is used to guarantee that the HS is safe if none of the states defined as unsafe is reachable. The issue with MC tools that use over-approximation [Henzinger *et al.*, 1997; Frehse, 2008] is that they are generally unable to produce a concrete trajectory, i.e., a counter-example, for an HS that is deemed unsafe. Exceptionally, tools such as dREACH [Kong *et al.*, 2015] and HyComp [Cimatti *et al.*, 2015], both built on top of Satisfiability Modulo Theories (SMT) solvers, are able to generate counterexamples. Another mechanism to directly seek a trajectory that violates a safety condition in an HS is *falsification* [Plaku *et al.*, 2009]. Falsification tools [Annpureddy *et al.*, 2011; Zhang *et al.*, 2018] find the inputs required to steer the system towards violating the property and use optimization techniques guided by robustness metrics. They generally assume that the system dynamics are governed by a black-box model whose behaviour is only observed from the input signals and their corresponding outputs via a simulator [Zutshi *et al.*, 2014; Corso *et al.*, 2020]. Simulation-based techniques for falsification, in addition, heavily rely on urgent discrete transitions, i.e., a discrete transition is immediately taken once the condition is satisfied, which makes them unsuitable for the non-deterministic behaviour in our benchmarks.

We characterize the language of explanations through a network of HA in parallel composition, where one automaton describes the HS and the other accepts the given observations. This composition effectively restricts the trajectories of the HS to those that are consistent with the observations. We then formalize the HSE problem as an optimization problem that looks over the language of explanations to find the most suitable one. In principle, this problem can be solved as a model checking problem but the aforementioned approaches followed by state-of-the-art MC tools struggle to find concrete trajectories, or assume limited knowledge of the systems dynamics, or only handle simple dynamics. To overcome this limitation, we further provide a formal mapping from HA to PDDL+ [Fox and Long, 2006], which enables the use of off-the-shelf automated planners. AI Planning heuristics have in the past proven very effective for finding trajectories [Wehrle and Helmert, 2009].

The connection between HA and PDDL+ has been mainly explored as the translation from PDDL+ planning to HA to underpin the formal interpretation of PDDL+ semantics with the HA formalism [Fox and Long, 2006; Bogomolov *et al.*,

2015]. Mapping PDDL+ into HA enables solving a hybrid planning problem using model-checking tools. Another research advocates encoding hybrid planning problems with SMT [Bryce *et al.*, 2015; Cashmore *et al.*, 2020] as a way to overcome the general limitation of PDDL+ planners to address nonlinear changes. We exploit the reverse mapping, encoding the network of HA as PDDL+ planning.

Our algorithmic solution to the HSE problem comes down to a plan recognition (PR) problem in an HS as we need to infer the trajectory (plan) that fulfills the observations of the HS (agent) in a continuous space. Unlike approaches to PR that use planners [Ramírez and Geffner, 2009], our aim is not goal prediction and our observations are not sub-sequences of actions as it is commonly the case in the PR approaches. We focus on finding the best sequence of states traversed by a system that explains some observations [Aineto *et al.*, 2020].

Our main contributions are: (a) a crisp formulation of the HSE problem that draws on the HA formalism; (b) a planning-based approximation to the HSE problem that exploits PDDL+ models and algorithms to scale up to large state spaces; and (c) an experimental analysis that includes a comparison between our planning-based approach and MC tools.

2 Preliminaries

A hybrid automaton is formally defined as an FSM, where states (*locations*) prescribe constraints over a set of variables, and transitions are governed by ordinary differential equations. We adopt the HA formulation used in [Doyen *et al.*, 2018] with explicit input variables. Firstly, we introduce some notation: X is a set of variables; $\mathsf{Constr}(X)$ is the set of constraints over X defined by the Backus Naur rule $\phi ::= \theta < 0 \mid \theta \leq 0 \mid \theta = 0 \mid \theta > 0 \mid \theta \geq 0 \mid \phi \wedge \psi$, being θ a polynomial expression over X; $[\![\phi]\!]$ refers to all valuations of ϕ over X, and $[\![\phi]\!]_v$ to a particular valuation v.

The tuple $H = \langle \mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, U, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$ defines a hybrid automaton, where:

- Loc is a set of *discrete variables* called *locations* of H.
- Lab is a set of *labels*.
- $\mathsf{Edg} \subseteq \mathsf{Loc} \times \mathsf{Loc}$ is a set the discrete transitions.
- $X = \{x_1, \ldots, x_n\}$, set of real-valued *state* variables.
- $U = \{u_1, \ldots, u_m\}$, set of real-valued *input* variables.
- $\mathsf{Init} : \mathsf{Loc} \rightarrow \mathsf{Constr}(X)$ states the possible valuations of X when H starts in location l, i.e., H can start in l with an initial valuation $v \in [\![\mathsf{Init}(l)]\!]$.
- $\mathsf{Inv} : \mathsf{Loc} \rightarrow \mathsf{Constr}(X \cup U)$ is the *invariant* of location l, which constrains the possible valuations of X and U when H is at l. H can stay in l as long as the valuation $v \in [\![\mathsf{Inv}(l)]\!]$.
- $\mathsf{Flow} : \mathsf{Loc} \rightarrow \mathsf{Constr}(\dot{X} \cup X \cup U)$ specifies what differential constraints apply on the behaviour of X when H is at l.
- $\mathsf{Jump} : \mathsf{Edg} \rightarrow \mathsf{Constr}(X^+ \cup X \cup U)$ gives the jump condition of edge e, often as a conjunction of a *guard*, $\mathsf{Guard} : \mathsf{Edg} \rightarrow \mathsf{Constr}(X \cup U)$, and a *reset* constraint, $\mathsf{Reset}(e) : \mathsf{Edg} \rightarrow \mathsf{Constr}(X^+)$, where variables in X^+ refer to the updated values of X after edge e is traversed.
- $\mathsf{Final} : \mathsf{Loc} \rightarrow \mathsf{Constr}(X)$ is the final condition of location l. H can only finish in l if the final valuation $v \in [\![\mathsf{Final}(l)]\!]$.

The semantics of H is the transition system $[\![H]\!] = \langle S, S_0, S_f, \Sigma, \rightarrow \rangle$; a *state* is a pair $q = (l, v)$, where $l \in \mathsf{Loc}$ and v is the valuation over X in l; S is the state space; S_0 is the set of possible initial states; S_f is the set of final states; $\Sigma = \mathsf{Lab} \cup \mathbb{R}^{\geq 0}$, where $\mathbb{R}^{\geq 0}$ denotes duration; and $\rightarrow$ is the transition relation that contains all tuples $((l, v), \sigma, (l', v'))$, which belong to any of these two types:

- **discrete transition**: $\sigma \in \mathsf{Lab}$, $e = (l, \sigma, l') \in \mathsf{Edg}$, and $(v, v') \in [\![\mathsf{Jump}(e)]\!]$. These transitions have no duration and are instantaneous.
- **continuous transition**: $\sigma \in \mathbb{R}^{\geq 0}$, $l = l'$, and $\mathsf{Flow}(l)$ determines the evolution of X, which changes from valuation v to valuation v' while remaining in location l as long as the valuation satisfies $[\![\mathsf{Inv}(l)]\!]$. The duration of continuous transitions is given by σ.

A *trajectory* is a run of $[\![H]\!]$ defined as $\tau = q_0 \sigma_1 q_1 \sigma_2 \ldots \sigma_n q_n$ or, equivalently, as $\tau = q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_n} q_n$, where $q_0 \in S_0$, and (q_{i-1}, σ_i, q_i) is a transition from $\rightarrow$, $\forall i \ 1 \leq i \leq n$. We define $d(q_i)$ as the sum of the duration for every transition between q_0 and q_i, and it represents the time at which state q_i is reached. If $q_n \in S_f$, we say that τ is a *trajectory accepted* by H. The set of all trajectories accepted by H is the language of H, denoted as $\mathcal{L}(H)$.

PDDL+ (Planning Domain Definition Language) is a high-level language to formulate planning instances that require reasoning over a mixture of discrete and continuous variables [Fox and Long, 2006]. A PDDL+ instance is given by the tuple $\Pi = \langle V, F, A, P, E, I, G \rangle$, where V and F are numeric and Boolean state variables, respectively, I is the initial state, and G is the goal condition. PDDL+ explicitly differentiates three types of state transitions, namely, actions A, events E and processes P. **An action** $a \in A$ is a pair $\langle \mathrm{pre}_a, \mathrm{eff}_a \rangle$ where pre_a is a formula containing both propositional and numeric conditions, and eff_a is a set of propositional and numeric assignments that cause *instantaneous changes*. The execution of an action a is decided by the planner as long as pre_a holds in the current state. **An event** $e \in E$ is structured as an action but its execution is triggered by the satisfaction of its preconditions rather than being decided by the planner. **A process** $p \in P$ is a pair $\langle \mathrm{pre}_p, \mathrm{eff}_p \rangle$, where pre_p has the same structure as pre_a, and eff_p only contains *continuous effects*, which are time-dependent effects of the form $\dot{x} = \xi$ (ξ represents the first derivative of x w.r.t. time).

Actions are in control of the agents while P and E describe what happens in the world when some condition is met. Processes last for as long as their conditions are met. For example, heating up a tank is a process which increases its temperature until a max value (e.g. $t = 100°\mathbf{C}$) is reached. Events correspond to instantaneous transitions that happen the instant their conditions are met, usually transforming the state in such a way that the conditions are no longer met. Events are uncontrollable; in the previous example, we might consider an event when the tank hits $t = 100°\mathbf{C}$, at which time the tank turns off.

We consider PDDL+ problems extended with a set C of global constraints (propositional formulae as the ones used in actions' preconditions) [Scala *et al.*, 2016]. C constrains state trajectories to those that always satisfy the formula in C.

A PDDL+ plan is a set of timed actions, i.e., pairs from $\mathbb{R}^{\geq 0} \times A$ plus the ending time $t_e \in \mathbb{R}^{\geq 0}$. Informally speaking, a plan is said to be valid iff *i)* the induced trajectory of states conforming to prescribed events, processes and global constraints, and obtained by starting from the initial state, is such that all actions have their precondition satisfied when they are applied, *ii)* the plan satisfies the goal at time t_e.

3 The HS Explanation Problem

The HSE problem is about finding a trajectory of an HS that matches some observations. This problem is formulated through two HA: one automaton models the behaviour of the HS, the other one tracks the run of the HS and checks when the produced trajectory matches the observations. We will refer to the first one as the **HS automaton**, and to the second one as the **monitor automaton**. Let $H = \langle \mathsf{Loc}, \mathsf{Lab}, \mathsf{Edg}, X, \mathsf{Init}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Jump}, \mathsf{Final} \rangle$ be the HS automaton:

Definition 3.1 (Observation)**.** *An* observation*, z, on the behaviour of H, $z \in \mathbb{R}^{\geq 0} \times \mathsf{Constr}(X)$, is the pair $\langle t, \phi \rangle$, where t is a* time point*, and ϕ is a constraint over X.*

By defining observations as constraints, ϕ is actually denoting a region of values, and so we can express qualitative uncertainty in the observations and partial observability over the state variables X. From a practical standpoint, sensors are rarely completely accurate and the dynamics that govern some variables in X may not be directly measurable, e.g. the linear acceleration of an aircraft.

Example 3.2. *Consider that the behaviour of an aircraft flying at a fixed altitude is modeled by the hybrid automaton H of Figure 3. H defines three control modes or locations, $\mathsf{Loc} = \{FS, AL, AR\}$, representing, respectively,* 'flying straight', 'adjusting course left' *and* 'adjusting course right'*. Let $X = \{x, y, \theta\}$ be the set of state variables of H, where $(x, y) \in \mathbb{R}^2$ describes the planar position, and θ is the angular orientation. Moreover, during the flying trajectory of the aircraft, its position is observable but θ is not. Initially, the aircraft is at position $(0, 0)$ and pointing at a direction given by the angle $\theta = 4.84$. An observation of the aircraft during flight is defined as*

$$z = \langle t, (c_x - x)^2 + (c_y - y)^2 \leq r^2 \rangle$$

where $(c_x, c_y) \in \mathbb{R}^2$ is the center of a circle of radius $r \in \mathbb{R}$. For instance, at time t=1, the observation is $z_1 = \langle 1, (5.593 - x)^2 + (-19.117 - y)^2 \leq 10^2 \rangle$, meaning that the position of the aircraft is somewhere within a circular region of radius $r = 10$ centered around $(5.593, -19.117)$.

A *sequence of observations* is denoted as $\omega = (z_k)_{k=1}^{K}$, where each z_k is an observation as expressed in definition 3.1. Let ω be an observation sequence of the behaviour of a system that is modeled through an HS automaton H. We define a **monitor automaton**, H^ω, which contains as many locations as the number of observations in ω, aimed at determining when the trajectory produced by H matches the observations in ω. The monitor automaton $H^\omega = \langle \mathsf{Loc}^\omega, \mathsf{Lab}^\omega, \mathsf{Edg}^\omega, X^\omega, U^\omega \ \mathsf{Init}^\omega, \mathsf{Inv}^\omega, \mathsf{Flow}^\omega, \mathsf{Jump}^\omega, \mathsf{Final}^\omega \rangle$ is formalized as:

- $\mathsf{Loc}^\omega \coloneqq \{l_0, \ldots, l_K\}$. H^ω contains one location for each observation $z_k \in \omega$; l_0 is the initial location of H^ω.
- Lab^ω is a set of labels (labels in H^ω are meaningless).
- $\mathsf{Edg}^\omega \coloneqq \{(l_{k-1}, \sigma_k, l_k) \mid 1 \leq k \leq K\}$. Discrete transitions move from one location to the next one.
- $X^\omega \coloneqq \{t\}$. H^ω explicitly models the passage of time through the time variable t for matching purposes.
- $U^\omega \subseteq X$. The input variables of H^ω are the subset of the state variables X of H that are observable.
- $\mathsf{Init}^\omega \coloneqq \{\mathsf{Init}^\omega(l_0) \coloneqq t = 0\}$. Time starts at $t = 0$.
- $\mathsf{Inv}^\omega \coloneqq \{\mathsf{Inv}^\omega(l_k) \coloneqq t \leq t_{k+1} + \Delta t \mid 0 \leq k \leq K - 1\}$. The automaton can remain at location l_k until the last instant allowed to validate observation z_{k+1}. Inv^ω is redudant with Jump^ω and can be omitted.
- $\mathsf{Flow}^\omega = \{\mathsf{Flow}^\omega(l) \coloneqq \dot{t} = 1 \mid l \in \mathsf{Loc}^\omega\}$, meaning that time advances normally.
- $\mathsf{Jump}^\omega = \{\mathsf{Jump}^\omega((l_{k-1}, \sigma_k, l_k)) \coloneqq t_k - \Delta t \leq t \leq t_k + \Delta t \wedge \phi_k \mid (l_{k-1}, \sigma_k, l_k) \in \mathsf{Edg}^\omega\}$. For observation $z_k = \langle t_k, \phi_k \rangle$, the jump condition ensures that there is one time point within a specified interval around t_k where the valuation of the constraint ϕ_k holds.
- $\mathsf{Final}^\omega \coloneqq \{\mathsf{Final}^\omega(l_K) \coloneqq \mathtt{true}\}$.

As specified in Jump, for each observation $z_k = \langle t_k, \phi_k \rangle$, we define a time window $[t_k - \Delta t, t_k + \Delta t]$, being Δt a time precision parameter, as the interval where we accept that the constraint ϕ_k can be matched. H^ω models the passage of time via the variable t, and it is constantly tracking the observable variables of H comprised in U^ω. The task of H^ω is to check that the values of U^ω match the constraint ϕ_k within the interval $[t_k - \Delta t, t_k + \Delta t]$; that is, it checks that $[\![\phi_k]\!]_u$ holds at any point in the interval. More specifically, H^ω awaits in location l_{k-1} for the input values u to match the observation z_k in the specified interval, moment at which H^ω transits to location l_k. The run of $[\![H^\omega]\!]$ produces a trajectory that traverses all the locations in Loc^ω. We will denote this trajectory as τ^ω.

Ultimately, our objective is to build a **network of HA** in parallel composition, $N = \{H, H^\omega\}$, that characterize the set of trajectories of H that explain the observed behaviour given by ω. For the network, we use global-time compositional semantics [Henzinger, 2000] where time elapses for all the HA in the network and for the same duration, and communication is achieved through shared variables, specifically through the variables U^ω.

Definition 3.3 (Synchronized trajectory of N)**.** *A synchronized trajectory of $N = \{H, H^\omega\}$ is the pair $\tau_S = \langle \tau, \tau^\omega \rangle$ such that $\tau = q_0 \sigma_0 \ldots q_n$ and $\tau^\omega = q_0^\omega \sigma_0^\omega \ldots q_m^\omega$ are concurrent local trajectories of H and H^ω, respectively, and have the same duration ($d(q_n) = d(q_m^w)$). A synchronized trajectory $\tau_S = \langle \tau, \tau^\omega \rangle$ is accepted by N, denoted $\tau_S \in \mathcal{L}(H, H^\omega)$, iff $\tau \in \mathcal{L}(H)$ and $\tau^\omega \in \mathcal{L}(H^\omega)$.*

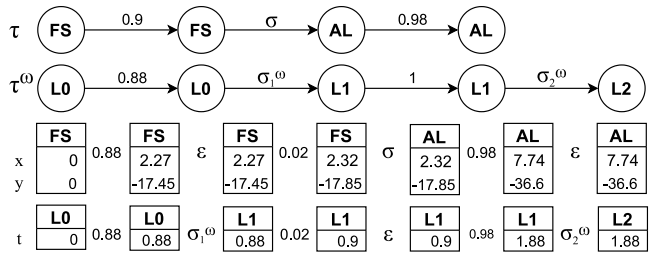

Figure 1: Explanation τ and support τ^ω (above), and their synchronization (below).

Definition 3.4 (Explanation & Support)**.** *$\tau \in \mathcal{L}(H)$ is an **explanation** for an observation sequence ω under H iff there exists a **support** $\tau^\omega \in \mathcal{L}(H^\omega)$ such that $\langle \tau, \tau^\omega \rangle \in \mathcal{L}(H, H^\omega)$. We define the language of explanations as $\mathcal{E}(H, \omega) = \{\tau \mid \langle \tau, \tau^\omega \rangle \in \mathcal{L}(H, H^\omega)\}$.*

Then, an explanation τ is the trajectory that explains the behaviour of the hybrid system[1], and a support τ^ω is the trajectory that determines when τ matches the observations.

Example 3.5. *We extend the Example 3.2 of the aircraft assuming an observation sequence ω of the HS automaton H (Figure 3) given by the following two observations:*

$$z_1 = \langle 1, (5.593 - x)^2 + (-19.117 - y)^2 \leq 10^2 \rangle$$
$$z_2 = \langle 2, (16.791 - x)^2 + (-35.595 - y)^2 \leq 10^2 \rangle$$

Our HSE problem is to find a trajectory of the aircraft modeled by H that explains $\omega = (z_1, z_2)$. We define a monitor automaton H^ω that contains one location for z_1 and one for z_2 using time precision $\Delta t = 0.125$. Fig. 1 shows a solution to the problem; i.e, a synchronized trajectory $\langle \tau, \tau^\omega \rangle$ of the network $N = \{H, H^\omega\}$ where τ is the aircraft explanatory trajectory for ω, and τ^ω determines when τ matches z_1 and z_2. The lower part of Fig. 1 details the synchronization of τ and τ^ω (only the observable variables of H, $U^\omega = \{x, y\}$, and the single variable t of H^ω are shown). Discrete transitions σ_1^ω and σ_2^ω happen at $t_1 = 0.88$ and $t_2 = 1.88$ when τ validates (matches) the observations z_1 and z_2. ϵ transitions are 'stutters' *representing self-loops $((l, v), \epsilon, (l, v))$ that replicate the state of one automaton when the other performs a discrete transition [Alur* et al.*, 1995].*

There may exist multiple trajectories of H^ω that support the validation of the observations by an explanation τ. For instance, in Example 3.5, the trajectory $\tau^\omega = (\mathsf{L0}, 0) \xrightarrow{1.12} (\mathsf{L0}, 1.12) \xrightarrow{\sigma_1^\omega} (\mathsf{L1}, 1.12) \xrightarrow{0.76} (\mathsf{L1}, 1.88) \xrightarrow{\sigma_2^\omega} (\mathsf{L2}, 1.88)$ also synchronizes with τ. This happens because the matching of an observation $< t_k, \phi_k >$ can occur at any point in the continuous time interval $[t_k - \Delta t, t_k + \Delta t]$. We refer to a support composed only of *urgent* discrete transitions, i.e., transitions that are immediately taken once the Jump condition is enabled, as a *greedy support* and denote it as τ_g^ω. The question that arises here is whether using a greedy support to match the observations of an explanation compromises the completeness of an HSE problem. Theorem 3.7 proves this is not the case.

Definition 3.6 (Prefix)**.** *Given two trajectories τ' and τ, τ' is a prefix of τ, $\tau' \subseteq \tau$, if either they are identical, or τ' is a finite sequence and τ begins with τ'.*

Theorem 3.7. *Let $\mathcal{L}_g(H, H^\omega) \subseteq \mathcal{L}(H, H^\omega)$ be the subset of synchronized trajectories formed with a greedy support τ_g^ω, and $\mathcal{E}_g(H, \omega) = \{\tau \mid \langle \tau, \tau_g^\omega \rangle \in \mathcal{L}_g(H, H^\omega)\}$. For every $\tau \in \mathcal{E}(H, \omega)$, there exists a $\tau' \in \mathcal{E}_g(H, \omega)$ such that τ' is a prefix of τ.*

Proof: We show that for any explanation τ in $\mathcal{E}(H, \omega)$, it is possible to construct a greedy trajectory τ_g^ω that supports a prefix $\tau' \subseteq \tau$. Let τ be an explanation for $\omega = (z_k)_{k=1}^K$, $z_k = \langle t_k, \phi_k \rangle$, and $I_k = [a_k, b_k]$, with $t_k - \Delta t \leq a_k \leq b_k \leq t_k + \Delta t$, be all the time points at which τ matches the observation; i.e., $[\![\phi_k]\!]_u$ holds $\forall t \in I_k$. A greedy support τ_g^ω for $\tau' \subseteq \tau$ is built by taking discrete transitions $((l_{k-1}, a_k), \sigma^\omega, (l_k, a_k))$ at time $m_k = \max(m_{k-1}, a_k)$ for $1 \leq k \leq K$. So observation z_k is matched as soon as the matching of z_{k-1} allows within the interval $I_k = [a_k, b_k]$. □

The maximum difference in duration between an explanation τ in $\mathcal{E}(H, \omega)$ and its prefix τ' in $\mathcal{E}_g(H, \omega)$ is $b_K - a_K$, which is bounded by $2\Delta t$. Then, for a sufficiently small Δt there is no significant difference between both languages. Particularly, when $\Delta t = 0$ we have $\mathcal{E}(H, \omega) = \mathcal{E}_g(H, \omega)$ since the explanation τ and its prefix τ' are identical. We can thus assume urgent transitions for H^ω, which effectively narrow the language of the network $N = \{H, H^\omega\}$, while guaranteeing that $\mathcal{E}_g(H, \omega)$ contains an almost complete prefix of any possible explanation in $\mathcal{E}(H, \omega)$.

Definition 3.8 (HS-explanation problem)**.** *An* HSE *problem is given by the tuple $\langle H, \omega \rangle$, where H is the description of an HA, and ω is a sequence of observations.*

The solution to $\langle H, \omega \rangle$ is defined as follows

$$\pi(H, H^\omega) := \min_{\langle q_0 \sigma_1 \dots q_n, q_0^\omega \sigma_1^\omega \dots q_m^\omega \rangle} f(q_0 \dots q_n) \quad (1)$$

$$\text{subject to } (q_i, \sigma_{i+1}, q_{i+1}) \in \rightarrow,\ 0 \leq i < n \quad (2)$$
$$(q_i^\omega, \sigma_{i+1}^\omega, q_{i+1}^\omega) \in \rightarrow^\omega,\ 0 \leq i < m \quad (3)$$
$$q_0 \in S_0, q_0^\omega \in S_0^\omega \quad (4)$$
$$q_n \in S_f, q_m^\omega \in S_f^\omega \quad (5)$$

Problem (1)–(5) is an optimization problem over the language of explanations $\mathcal{L}(H, H^\omega)$. Subproblem (2)–(5) is the underlying reachability problem of searching trajectories that belong to $\mathcal{L}(H, H^\omega)$. Solutions $\pi(H, H^\omega)$ are synchronized trajectories $\tau_S = \langle \tau, \tau^\omega \rangle$ accepted by N, as ensured by constraints (4)–(5) that minimize a user-defined function $f(\cdot)$ that measures the implausibility of an explanation.

Figure 2 illustrates graphically the HSE problem for our aircraft example. A 100-sec trajectory (black line) represents the original trajectory of the aircraft produced by the HS automaton H. In order to generate the observations of the aircraft position, the original trajectory was sampled at 1-sec intervals with different radius, $r \in \{5, 10, 20, 40\}$. The colored lines are the trajectories that explain the observed behaviour of the aircraft under the different values of r.

[1]Our notion of explanation is not to be confused with *planning explanations* [Fox *et al.*, 2017] which also account for the process that generated the trajectory.

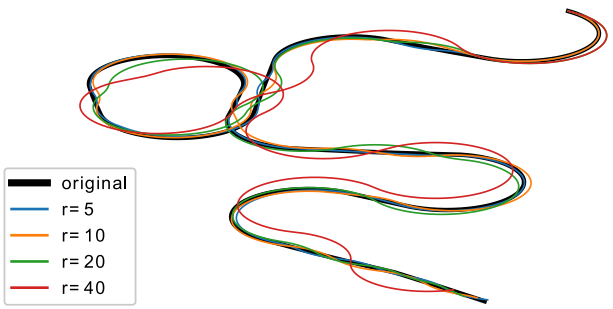

Figure 2: Original trajectory (black) and explanations (colored).

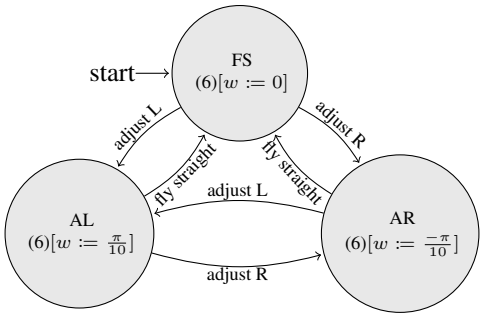

Figure 3: HA model of an aircraft

4 HS Explanation as Planning

We present our proposal to approximate the solution to the HSE problem as for equations (1)–(5) using PDDL+.

Optimization function. As commented in Def. 3.8, the user defines a function to measure the quality of a solution. In our approximation, we use $f(\tau) = |\tau|$, which measures the length or number of discrete transitions of the explanation[2]. Discrete transitions are the ones that truly determine the evolution of the HS as the continuous transitions merely indicate the awaiting of the system in a location. Therefore, we believe minimizing $|\tau|$ is a reasonable proxy for more complicated objective functions. In general, the quality of the solution is subject to the range of values encompassed by the definition of the observations; the smaller the region of values, the more accurate the solution. This can be observed in Fig. 2, where the explanation under $r = 5$ generates a trajectory that basically covers the original flight trajectory.

Mapping the network of HA to PDDL+. The goal is to map the network of HA, $N = \{H, H^\omega\}$, into a PDDL+ planning instance $\Pi(H, H^\omega)$ and solve it with off-the-shelf planners. We will call $\pi_+(H, H^\omega)$ the solution returned by a PDDL+ planner to the HSE problem. A solution $\pi_+(H, H^\omega)$ will contain actions corresponding to discrete transitions of $[\![H]\!]$ such that $f(\tau) = |\tau|$ is minimized, and it will reach a state satisfying the constraints in equation (5).

Let H be any hybrid automaton (either an HS or a monitor automaton). We first describe the mapping of H to a PDDL+ problem $\Pi(H) = \langle V, F, A, P, E, I, G, C\rangle$ that preserves the semantics of $[\![H]\!]$:

- The state variables X of H map directly to numeric variables ($V = X$).
- The locations of H are represented using Boolean variables ($F = \mathsf{Loc}$).
- A discrete transition of H is represented with $a = \langle \mathrm{pre}_a, \mathrm{eff}_a\rangle \in A$, reflecting the non-determinism of H. $\mathsf{Jump}(e)$ is decomposed into $\mathsf{Guard}(e)$, encoded in pre_a, and $\mathsf{Reset}(e)$, encoded in eff_a (change to next location).
- The continuous transitions of H that model the evolution of X as time passes are represented with processes P, $P = \{\langle l, \mathsf{Flow}(l)\rangle \mid l \in \mathsf{Loc}\}$.
- $I \models \mathsf{Init}$, we assume that the initial space S_0 induced by the initial condition Init of H is a singleton.
- $G = l_{final}$. Without loss of generality, we assume there is a single location l_{final} such that $\mathsf{Final}(l_{final}) = \texttt{true}$ and $\mathsf{Final}(l) = \texttt{false}$ for all $l \neq l_{final}$ as any HA can be reduced to this form.
- Inv of H is represented as the global constraints $C = \bigwedge_{l\in\mathsf{Loc}} \neg l \vee \mathsf{Inv}(l)$: as long as H is in location l, $\mathsf{Inv}(l)$ must hold (if specified). As locations are mutually exclusive, only one constraint will be active in a state.

PDDL+ events can be used, instead of actions, to encode urgent discrete transitions. Intuitively, discrete transitions of the HS automaton map into actions that represent the actual control of the agent (e.g., adjusting course left); while discrete transitions of the monitor automaton map into actions (or PDDL+ events if we want a greedy support) that are only executable when the trajectory matches with an observation.

Mapping a network $N = \{H, H^\omega\}$ into a PDDL+ planning problem $\Pi(H, H^\omega)$ is done by composing the elements of $\Pi(H)$ and $\Pi(H^\omega)$ via the union operator for sets X, F, A, P and I, and via the conjunction operator for constraints C and goal condition G. A solution plan is a set of timed instantaneous transitions $\pi_+(H, H^\omega) = (\{\langle t_0, a_0\rangle, \ldots, \langle t_n, a_n\rangle\}, t_e)$ such that $a_i \in A$, $0 \leq i \leq n$, and t_e is the ending time of the plan. $\pi_+(H, H^\omega)$ models the discrete transitions of both H and H^ω and represents a synchronized trajectory of $N = \{H, H^\omega\}$. Extracting the local plans for H and H^ω is a straightforward process that only requires checking whether a_i belongs to Lab or Lab^ω. For instance, a PDDL+ plan for Ex. 3.5 is: $\{\langle 0.88, \texttt{validate\_1}\rangle, \langle 0.90, \texttt{adjust\_left}\rangle, \langle 1.88, \texttt{validate\_2}\rangle\}, t_e)$, which includes i) the local plan for H, $(\{\langle 0.9, \texttt{adjust\_left}\rangle\}, t_e = 1.88)$, representing an explanation τ of the HS, and ii) the local plan for H^ω, $(\{\langle 0.88, \texttt{validate\_1}\rangle, \langle 1.88, \texttt{validate\_2}\rangle\}, t_e = 1.88)$, representing a support τ^w that specifies when the HS matches the two observations. Both local plans can effectively be used to generate a synchronized trajectory to the HSE problem.

5 Empirical Evaluation

We evaluate our approach computationally on three domains with different types of dynamics: piece-wise constant, linear and nonlinear. All domains feature non-deterministic (even arbitrary) discrete mode switches.

[2]The length of a support τ^ω is always bounded by the K observations in ω

	Thermostat				Platoon			Flight		
Size	HC	dR	HSE_P	HSE_P$_e$	dR	HSE_P	HSE_P$_e$	dR	HSE_P	HSE_P$_e$
2	10 (0.1)	10 (0.2)	10 (0.5)	10 (0.5)	0	10 (1.1)	10 (0.6)	10 (0.3)	10 (0.6)	10 (0.5)
4	10 (0.5)	10 (0.5)	10 (0.6)	10 (0.6)	0	10 (1.0)	10 (0.6)	1 (14)	10 (2.8)	10 (0.7)
6	10 (0.7)	8 (44)	10 (0.6)	10 (0.6)	0	10 (0.7)	10 (0.6)	0	10 (1.3)	10 (0.8)
8	10 (1.3)	3 (15)	10 (0.6)	10 (0.6)	0	10 (0.8)	10 (0.6)	0	10 (1.6)	10 (0.8)
10	10 (1.6)	0	10 (0.7)	10 (0.6)	0	10 (4.3)	10 (0.7)	0	10 (17)	10 (0.8)
20	10 (11)	0	10 (1.4)	10 (0.7)	0	10 (3.9)	10 (0.9)	0	7 (74)	10 (1.3)
30	10 (60)	0	10 (3.1)	10 (1.0)	0	8 (67)	10 (1.1)	0	1 (127)	10 (1.6)
40	10 (118)	0	10 (5.2)	10 (1.1)	0	1 (191)	10 (1.4)	0	0	10 (1.9)
50	5 (261)	0	10 (8.2)	10 (1.3)	0	0	10 (1.5)	0	0	10 (2.3)
60	0	0	10 (14)	10 (1.6)	0	0	10 (1.9)	0	0	10 (3.5)
70	0	0	10 (20)	10 (1.8)	0	0	10 (2.2)	0	0	10 (4.3)
80	0	0	10 (27)	10 (1.9)	0	0	10 (2.4)	0	0	10 (5.8)
90	0	0	10 (38)	10 (2.2)	0	0	10 (2.8)	0	0	10 (9.4)
100	0	0	10 (49)	10 (2.5)	0	0	10 (3.1)	0	0	10 (14)

Table 1: Coverage and run-time results. Each entry reports the number of solved problems, and the median time in seconds for each system. dR stands for dReach, HC for HyComp.

Thermostat (constant). This domain uses Henzinger's thermostat [Henzinger, 2000] hybrid automaton. The flows are made piece-wise constant; invariants and guards leave a window of 1ºC where the system may non-deterministically switch modes. Temperature is the only observable variable.

Platoon (linear). This domain [Makhlouf and Kowalewski, 2014] is a staple in the linear dynamics competition for formal verification of HS. The domain depicts a platoon of networked vehicles following each other switching between two discrete modes: a normal operation and a communication-failure mode. We handle a platoon of 4 vehicles for a total of 9 continuous variables: relative distance and velocity to the vehicle ahead, and acceleration of each vehicle save for the leader. The observable variables are the 3 relative distances.

Flight (nonlinear). This models the domain of Ex. 3.5. Assuming a simplified physics model that ignores wind and gravitation, the in-flight dynamics of an aircraft can be described by the following differential equations:

$$\dot{x} = v\ cos\ \theta \qquad \dot{y} = v\ sin\ \theta \qquad \dot{\theta} = w \qquad (6)$$

where v and w are the linear and angular velocity. In our experiments v is made constant, so w is the only control parameter (Figure 3). The flow constraints (6)$[w := a]$ shown in each location are abbreviations of the dynamics of equation (6) with w replaced by the angular velocity a of the respective location. We observe the planar coordinates (x,y).

We computed 10 trajectories for each domain using a simulator, and extracted observation sequences by sampling the trajectories at a fixed rate of 1sec. For each trajectory we scale the number of observations from 2 to 100: the smallest instance asks to find a trajectory explaining 2 observations over 2 secs of execution while the largest 100 observations over 100 secs of execution.

We attempt to solve the HSE problem with 3 configurations: HSE_MC (via **M**odel-**C**hecking), and 2 configurations via **P**lanning, HSE_P and HSE_P$_e$ differing in how discrete transitions of H^ω are encoded: HSE_P uses actions and HSE_P$_e$ uses greedy supports; i.e., events. The comparison with HSE_MC aims at evaluating the usefulness of our mapping into planning against the long-established technique of MC. We tested DREACH and HYCOMP, two MC tools that are able to generate counter-examples. Encoding the HSE problem into the MC tools simply requires defining the unsafe condition as Final^ω.

We tried different PDDL+ planners [Piotrowski *et al.*, 2016; Penna *et al.*, 2009; Cashmore *et al.*, 2020], but focused on ENHSP [Scala *et al.*, 2020] as it was the one giving us the better performance with support for the dynamics of all our domains. We ran ENHSP using the h^{mrp}_{max} heuristic on top of plain A* with a random tie-breaking. ENHSP is a discretization based planner that decouples time-discretization for planning (δ_p) from the one of plan execution (δ_e) [Ramírez *et al.*, 2017]. Keeping δ_e small ensures that i) the generated explanations remain valid and ii) far more states can match the observations. We keep $\delta_p = \delta_e = 0.01$ over all domains when ENHSP run using HSE_P, and set $\delta_p = 0.1$ in Thermostat and Platoon, and $\delta_p = 0.4$ in Flight when ENHSP run with HSE_P$_e$. All experiments run on an Intel Core i5 3.1GHz x 4, time and memory limits set to 300s and 8GB, respectively.

Results. For each domain-configuration and problem size, we show the number of problems solved and the median time over successful runs (shown in parentheses in Table 1). A problem is solved if a solution was found within a time frame of 300s. Solutions found by HSE_P and HSE_P$_e$ were validated in a simulator using the plan found by ENHSP as a controller automaton in parallel composition with H and H^ω. We can observe that HYCOMP and DREACH scale up poorly (no results of Platoon and Flight are shown for HYCOMP as it only supports constant derivatives; 0 indicates no solution found within the time budget). We attribute the poor performance of DREACH to the large size of the variables domain. HSE_P, the configuration that uses actions to match observations in H^ω, shows better performance than the MC tools, but still unable to solve the full suite of problems as computation times quickly ramp up for the linear and nonlinear domains. HSE_P$_e$, however, solves all the problems. Indeed, HSE_P$_e$ would be able to scale to much larger instances. The nonlinear dynamics of the Flight domain are clearly more computationally expensive for the planner as evidenced by the reported times and a higher difficulty to scale up. The quality of the solutions found by both versions of HSE_P are the same for *thermostat* and *platoon* domains. HSE_P slightly outperformed HSE_P$_e$ in the *flight* domain where it found solutions up to 10% shorter in average for problem sizes 8 and 10. The MC tools produced plans of the same size as the best plan obtained between HSE_P and HSE_P$_e$.

6 Conclusions

This paper has shown a novel and general method to explain the observed behaviour of an HS that exhibits different control modes. Our HSE specification uses the HA formalism, which enables already existing model checking tools to tackle the problem. We, however, propose a planning-based approach that exploits the capabilities of heuristic PDDL+ planners to explore the state space and obtain good quality solutions. Our empirical evaluation shows a significant increase in performance with respect to model checking tools. In light of these results, exploring the combination of PDDL+ planning with model checking to tackle falsification problems seems to be a promising line of research.

Acknowledgments

This work is supported by the Spanish MINECO project TIN2017-88476-C2-1-R and partially supported by the EU ICT-48 2020 project TAILOR (No. 952215). Diego Aineto is partially supported by the *FPU16/03184*. Enrico Scala and Ivan Serina are partially supported by Regione Lombardia, Call Hub Ricerca e Innovazione, within the project "MoSoRe@Unibs—Infrastrutture e servizi per la Mobilità Sostenibile e Resiliente", 2020–2022 and by AIPlan4EU, a project funded by EU Horizon 2020 research and innovation programme under GA n. 101016442 (since 2021).

References

[Aineto *et al.*, 2020] Diego Aineto, Sergio Jimenez, and Eva Onaindia. Observation Decoding with Sensor Models: Recognition Tasks via Classical Planning. In *ICAPS*, 2020.

[Alur *et al.*, 1995] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The Algorithmic Analysis of Hybrid Systems. *Theor. Comput. Sci.*, 138(1):3–34, 1995.

[Annpureddy *et al.*, 2011] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *TACAS*, 2011.

[Bogomolov *et al.*, 2015] Sergiy Bogomolov, Daniele Magazzeni, Stefano Minopoli, and Martin Wehrle. PDDL+ planning with hybrid automata: Foundations of translating must behavior. In *ICAPS*, 2015.

[Bryce *et al.*, 2015] Daniel Bryce, Sicun Gao, David J. Musliner, and Robert P. Goldman. SMT-based nonlinear PDDL+ planning. In *AAAI*, 2015.

[Cashmore *et al.*, 2020] Michael Cashmore, Daniele Magazzeni, and Parisa Zehtabi. Planning for hybrid systems via satisfiability modulo theories. *J. Artif. Intell. Res.*, 67:235–283, 2020.

[Cimatti *et al.*, 2015] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. HyComp: An SMT-Based Model Checker for Hybrid Systems. In *TACAS*, 2015.

[Corso *et al.*, 2020] Anthony Corso, Robert J Moss, Mark Koren, Ritchie Lee, and Mykel J Kochenderfer. A survey of algorithms for black-box safety validation. *arXiv preprint arXiv:2005.02979*, 2020.

[Doyen *et al.*, 2018] Laurent Doyen, Goran Frehse, George J. Pappas, and André Platzer. Verification of Hybrid Systems. In *Handbook of Model Checking*, pages 1047–1110. Springer, 2018.

[Fox and Long, 2006] Maria Fox and Derek Long. Modelling mixed discrete-continuous domains for planning. *J. Artif. Intell. Res.*, 27:235–297, 2006.

[Fox *et al.*, 2017] Maria Fox, Derek Long, and Daniele Magazzeni. Explainable planning. *CoRR*, abs/1709.10256, 2017.

[Frehse, 2008] Goran Frehse. PHAVer: algorithmic verification of hybrid systems past hytech. *Int. J. Softw. Tools Technol. Transf.*, 10(3):263–279, 2008.

[Henzinger *et al.*, 1997] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model checker for hybrid systems. *Int. J. Softw. Tools Technol. Transf.*, 1(1-2):110–122, 1997.

[Henzinger, 2000] Thomas A. Henzinger. The Theory of Hybrid Automata. In *Verification of Digital and Hybrid Systems*, pages 265–292. Springer Berlin Heidelberg, 2000.

[Kong *et al.*, 2015] Soonho Kong, Sicun Gao, Wei Chen, and Edmund M. Clarke. dReach: δ-Reachability Analysis for Hybrid Systems. In *TACAS*, 2015.

[Makhlouf and Kowalewski, 2014] Ibtissem Ben Makhlouf and Stefan Kowalewski. Networked cooperative platoon of vehicles for testing methods and verification tools. In *ARCH@ CPSWeek*, 2014.

[Penna *et al.*, 2009] Giuseppe Della Penna, Daniele Magazzeni, Fabio Mercorio, and Benedetto Intrigila. UPMurphi: A Tool for Universal Planning on PDDL+ Problems. In *ICAPS*, 2009.

[Piotrowski *et al.*, 2016] Wiktor Mateusz Piotrowski, Maria Fox, Derek Long, Daniele Magazzeni, and Fabio Mercorio. Heuristic planning for hybrid systems. In *AAAI*, 2016.

[Plaku *et al.*, 2009] Erion Plaku, Lydia E. Kavraki, and Moshe Y. Vardi. Hybrid systems: from verification to falsification by combining motion planning and discrete search. *Formal Methods Syst. Des.*, 34(2):157–182, 2009.

[Ramírez and Geffner, 2009] Miquel Ramírez and Hector Geffner. Plan Recognition as Planning. In *IJCAI*, 2009.

[Ramírez *et al.*, 2017] Miquel Ramírez, Enrico Scala, Patrik Haslum, and Sylvie Thiébaux. Numerical integration and dynamic discretization in heuristic search planning over hybrid domains. *CoRR*, abs/1703.04232, 2017.

[Scala *et al.*, 2016] Enrico Scala, Patrik Haslum, Sylvie Thiébaux, and Miquel Ramírez. Interval-based relaxation for general numeric planning. In *ECAI*, 2016.

[Scala *et al.*, 2020] Enrico Scala, Alessandro Saetti, Ivan Serina, and Alfonso Emilio Gerevini. Search-guidance mechanisms for numeric planning through subgoaling relaxation. In *ICAPS*, 2020.

[Wehrle and Helmert, 2009] Martin Wehrle and Malte Helmert. The Causal Graph Revisited for Directed Model Checking. In *SAS*, 2009.

[Zhang *et al.*, 2018] Zhenya Zhang, Gidon Ernst, Sean Sedwards, Paolo Arcaini, and Ichiro Hasuo. Two-layered falsification of hybrid systems guided by monte carlo tree search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2894–2905, 2018.

[Zutshi *et al.*, 2014] Aditya Zutshi, Jyotirmoy V. Deshmukh, Sriram Sankaranarayanan, and James Kapinski. Multiple shooting, CEGAR-based falsification for hybrid systems. In *EMSOFT*, 2014.