Heuristic Planning for Hybrid Dynamical Systems with Constraint Logic Programming

Shaun Mathew[1], Mikhail Soutchanski[1]

[1] *Toronto Metropolitan University, 245 Church St, ENG281, Toronto, ON, M5B 2K3, Canada,*
https://www.cs.torontomu.ca/mes

Abstract

We explore how planning for near optimal behaviors of mixed discrete-continuous systems can be done by deductive reasoning. For reasoning to be efficient, it must be properly controlled. It is surprising and mathematically non-trivial fact that this control can be achieved in a domain-independent way for a large class of planning domains with non-linear continuous processes. Moreover, in contrast to other planners for hybrid dynamical systems that use the single-model approaches and have to instantiate all actions and fluents before search starts, our planner is lifted, namely it works directly with action schemas. Following Constraint Logic Programming framework, it delegates solution of the optimization problem with respect to numerical and temporal constraints to an external numerical optimization solver. We discuss that our theorem-proving based planner for hybrid systems achieves performance competitive with the state-of-the-art approaches to temporal numerical planning on several well-known benchmarks. We argue that our deductive approach may have potential to compete with existing operation research techniques when solving large-scale optimization problems for relational hybrid systems.

1. Introduction

Hybrid systems, sometimes referred to as hybrid dynamical systems, are heterogeneous systems that consist of components with discrete and continuous behavior. The discrete component transitions between states depending on instantaneous control actions or exogenous events, while the continuous variables within a state change over time according to ordinary differential equations (ODE). Each discrete state can be provided with its own system of ODEs. In a general case, transitions between states can switch between different continuous flows and can reset some of the parameters. In the literature, hybrid automata are the most popular method of modelling hybrid systems [1, 32, 59]. Hybrid automata generalize ordinary finite-state automata, since the states of the former include numeric variables whose evolution is modeled by ODEs.

There are examples of practical hybrid systems where the transitions between the states are parameterized, and the states themselves are no longer atomic, but have relational structure, as in data bases. A prominent example of parameterized hybrid systems is articulated complex robots with autonomous motions where one has to solve the problem of integration of task and motion planning (TAMP), e.g., see [79, 63, 94, 74, 41, 30]. Other practical examples of the control problems for relational hybrid systems include the Unit Commitment problem in the large-scale

IPS-RCRA-SPIRIT 2023: 11th Italian Workshop on Planning and Scheduling, 30th RCRA Workshop on Experimental evaluation of algorithms for solving problems with combinatorial explosion, and SPIRIT Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy. November 7-9th, 2023, Rome, Italy [13]

electric power grids where power generators have to be properly scheduled when they turn on/off to fulfill daily electricity demands with minimal cost while respecting non-linear constraints on electricity flows [20, 76, 23, 60], and the control of streetlights to compute a new timing sequence that eliminates unexpected traffic jams in a region of a large city [96].

The most common and popular approach to the analysis of hybrid automata is based on model checking [26, 32]. Given a hybrid automaton and a safety property, reachability analysis determines if the system can ever enter unsafe states at any point of its run-time. It is well-known that in the general case the reachability problem for hybrid systems is undecidable [2, 48, 32]. Despite this fundamental conceptual obstacle, there is a lot of effort in developing sound (but incomplete) specialized tools for reachability analysis [1, 26], as well as research on approximate algorithms [40], and work into exploring robust classes of hybrid systems [81].

As an alternative to model-checking, there are theorem-proving techniques [87]. Cordell Green proposed to consider problem solving and the planning problem as fundamentally a reasoning problem that can be solved using a theorem-proving approach in Situation Calculus (SC) [44]. However, because he did not anticipate any control over unrestricted resolution, his program did not work well [47, 55, 54]. Subsequent planning research moved away from theorem proving to specialized representations such as STRIPS and ADL. Almost all modern research in automated (common-sense level) AI planning focuses on search with domain-independent heuristics in a single model of a discrete transition system [16, 42]. There were occasional proposals to revive Green's planning as a deductive reasoning approach and implement the automated planners in PROLOG, e.g., see a detailed discussion in [92], but they were not supported with competitive implementations using domain independent heuristics.

Despite this history, there are advantages to theorem-proving based planning over the single-model approach. The latter requires both the Closed World Assumption (CWA) and the Domain Closure Assumption (DCA). The main reason for relying on the (unrealistic) DCA in the single-model approach is the need for instantiating the action schemas and fluents before search for a plan can start. As a consequence, the single-model approach to planning sometimes has issues with scalability as the number of objects in the domain increases, e.g., see [65, 67, 56, 29]. It turns out, even if an implementation relies on DCA, there is no need to instantiate action schemas in advance, if one employs PROLOG to solve planning problems described in the Situation Calculus [91]. This deductive approach is lifted in the sense that it works directly with action schemas and determines at run time what are the next possible (ground) actions.

Our research vision is that efficient search for a near optimal plan for relational hybrid systems can be accomplished within the deductive approach using a domain independent heuristic. This paper describes a preliminary version of a lifted Non-linEArr Temporal (NEAT) planner that makes a step towards this long-term vision. To simplify implementation this version relies on both CWA and DCA, and a more general version is left to future work. We keep our presentation rather informal to make it more accessible to a larger audience. In the next section, we provide a short review of temporal SC and introduce a simple example. Also, we mention the previous work on Constraint Logic Programming (CLP). Then, we describe our methodology and summarize the experimental results collected from running NEAT and the state-of-the-art planners for hybrid systems on a few standard benchmarks. In conclusion, we propose our hypothesis why our approach can potentially become competitive with the Mixed Integer Non-Linear Programming (MINLP) approach to control of hybrid systems that is popular in Operation Research (OR) [62].

2. Background

Situation Calculus (SC) was introduced in [68, 69], and later refined by Reiter [82, 84] who introduced the Basic Action Theory (BAT) as a solution to the frame problem in first-order logic (FOL). Unlike the notion of state that has a complicated meaning, SC has situation, i.e., a sequence of actions, which provides a concise symbolic representation that is a convenient proxy for the state [78]. A BAT consists of the following axioms.

1. The foundational axioms for situations characterizing situations as a tree with the initial situation S_0 as the root [84].
2. The successor state axioms (SSA) for fluents [82]. For each (atemporal) fluent they characterize what actions make it true, and what actions make it false. Informally speaking, SC representation is fluent centered, in contrast to PDDL that is action-centered [46].
3. The action precondition axioms (PA) are formulated using the special predicate $poss(A, S)$: when action A is possible in situation S. There is one PA for each domain specific action A. Usually, the preconditions are conjunctions of fluent literals and numerical constraints.
4. The unique name axioms (UNA) for actions and objects.
5. A FOL theory that describes what holds in the initial situation, but in most benchmark planning instances this is simply the conjunction of the fluents wrt the initial situation S_0.

Notice a BAT does not include state constraints that characterize the invariants of the system. Syntactically, state constraints are $\forall s$-quantified sentences with s as the only situation variable. They are assumed to be (manually) compiled offline into the initial theory [58, 78, 70, 84], since in a general case the presence of state constraints leads to the conceptual problems [58, 19].

We implement BATs in PROLOG, see [84, 57] for details and examples. PROLOG facilitates a natural implementation of situation terms and BATs thanks to its semantics [97, 61]. In particular, the initial situation S_0 is represented as the empty list $[\,]$, and the situation $do(A, S)$ that results from executing action A in a previous situation S is represented as the list $[A \mid S]$, with the most recent action A in the head of the list. In the sequel, we provide examples of the axioms both in PROLOG and in FOL to facilitate understanding for the readers who are not familiar with PROLOG. In FOL formulas, all free (lower case) variables are implicitly $\forall$-quantified at front, while in PROLOG the variables start with an upper-case letter.

Reiter's book [84] includes his sequential temporal SC [83] that represents durative actions (and processes extended in time) with two instantaneous initiating and terminating actions and with a fluent that defines a durative action (a process, respectively). For example, to represent the motion process from a location X to another location Y during a time interval occupied by situation S, one can introduce predicate fluent $moving(X, Y, S)$ and consider a SSA for this fluent assuming that an instantaneous action $beginMoving(X, Y, T)$ makes this fluent true, if it is executed at time T, and another instantaneous action $endMoving(X, Y, T)$ makes this fluent false, in the case if in S there is ongoing process of moving from X to Y.

```
moving(X,Y, [A | S]) :- A=beginMoving(X,Y,T).
moving(X,Y,[A|S]) :- moving(X,Y,S), not A=endMoving(X,Y,T).
```

$$\forall x, y, a, s.\, moving(x, y, do(a, s)) \leftrightarrow \exists t(a = beginMoving(x, y, t)) \vee moving(x, y, s) \wedge \neg\exists t(a = endMoving(x, y, t)).$$

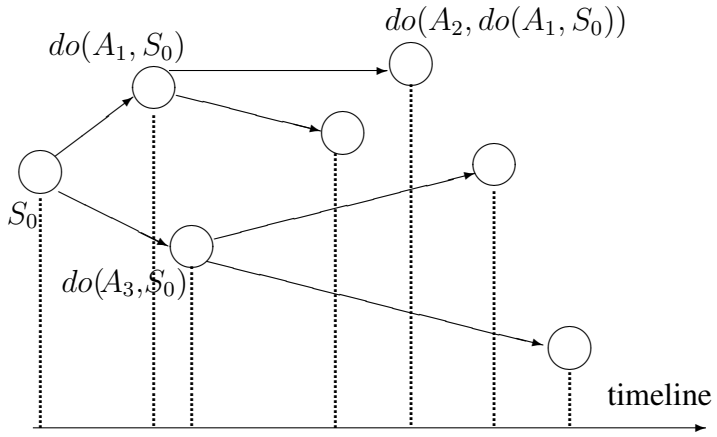

Figure 1: Let the left most circle represent the initial situation S_0. Suppose that actions A_1, A_3 are possible in S_0, and they result in situations $do(A_1, S_0)$ and $do(A_3, S_0)$. Also, shown is situation $do(A_2, do(A_1, S_0))$ that results from doing A_2 in $do(A_1, S_0)$. To avoid clutter other actions and the resulting situations are not named, and branching rightwards is omitted. Note each circle maps to a point on the timeline representing the unique moment when situation starts. Each situation s may last for an interval from the moment it starts until an action a occurs and produces $do(a, s)$.

Reiter requires all action functions to have the last argument time, but he considers only the usual *atemporal* fluents that do not mention time as an argument.

To illustrate his representation informally imagine the single horizontal time-line under the situation tree (with the root as the left-most node) that is branching rightwards. The moment of time T when situation S starts is characterized using the special predicate $start(S, T)$. This predicate maps S to the unique moment of time T when an instantaneous action resulted in S. Each situation can last over an interval of time or just an instant, depending on when the next action is executed. For each domain specific instantaneous action $actionName(T)$, there is an atomic statement $time(actionName(T), T)$ relating the variable T with the moment of time when $actionName(T)$ occurs. There are two domain independent temporal axioms:

```
start([], 0).            /*Let the initial situation start at 0 */
start([A|S],T) :- time(A,T).  /*Start time of situation [A|S] */
```

The last axiom is saying that if action A is executed at moment T in situation S, then this results in a new situation $[A \mid S]$ that starts at the same moment of time T. In FOL, Reiter uses the function symbols $start(do(a, s))$ that represents the moment of time when situation $do(a, s)$ starts, and $time(a)$ that represents the moment when an instantaneous action a is executed. These two functions are related using the following simple axioms:

$time(A(\bar{x}, t)) = t.$ /* for arbitrary action function $A(\bar{x}, t)$*/

$\forall a, s, t.\ start(do(a, s)) = time(a).$

Reiter's temporal BAT is limited, because he does not allow fluents to change over time within situation, in between actions. But as we know, both in science and engineering, physical quantities depend on time explicitly. The recently developed Hybrid Temporal Situation Calculus (HTSC) considers temporal fluents in addition to atemporal fluents [9, 10]. In HTSC, there are both atemporal fluents (as in [84])), and also *temporal fluents* that have time argument in addition

to situation argument. To simplify our presentation we consider next the example formulated in PROLOG. Also, to keep our presentation coherent we collected all FOL axioms for the following example in a separate Appendix at the end of this paper. The readers who are not familiar with PROLOG can consult those FOL axioms. In this example, we use the special symbols $= and $=< to implement the equality or inequality constraints on the variables. These constraints are simply collected in a separate data structure at run time while the program is executed. They are taken into account only when the related (linear or non-linear) optimization problem has to be solved for the given numerical objective.

Example. Consider a finite number of balls that can be dropped and that can elastically bounce from the floor. There are the actions $drop(B, Time)$ and $catch(B, Time)$ that can be executed by the agent. When a ball B hits the floor, a natural event $bounce(B, Time)$ occurs. When the ball B rises to the top-most point of its trajectory, another natural event $atPeak(B, Time)$ occurs, and then the ball starts falling down. The atemporal fluent $falling(B, S)$ means the ball B is falling down and accelerating under the Earth gravity. The atemporal fluent $flying(B, S)$ means the ball B bounced, it is flying up in situation S and decelerating due to gravity. We assume the vertical axis is oriented downwards, i.e., if a ball is falling down, then its speed is positive and increases. But when the ball bounces, its speed is negative and decreases. For simplicity we assume all balls are perfectly elastic, i.e., there is no damping. In addition, we consider temporal fluent $dist(B, D, T, S)$ that characterizes the ball's height D at the moment of time T within S, and temporal fluent $vel(B, V, T, S)$ that characterizes instantaneous velocity V at the moment T within the time interval when S lasts. These temporal fluents describe time dependent change within situation S, in between two occurrences of the agent actions and/or the natural events. (Subsequently, we do not distinguish between actions and events; both of them are represented with action terms.) The following are the precondition axioms (PAs) and an instance specific atomic statements.

```
ball(b1).  ball(b2).    epsilon(E) :- E is (1/100). /* approximation*/
time( drop(Ball,T), T).       time( catch(Ball,T), T).
time( bounce(Ball, T), T).    time( atPeak(Ball, T), T).
natural(bounce(B,T)).   natural(atPeak(B,T)). /* Two Nature actions */
agent(drop(B,T)).       agent(catch(B, T)).   /* Two Agent actions */

poss(drop(B,Time), S) :- ball(B), not falling(B,S),
    not flying(B,S),start(S, T), T $=< Time.
poss(catch(B,Time), S) :- ball(B), start(S,T), T $=< Time,
    ( falling(B,S); flying(B,S) ).  /*can catch if flying or falling*/
poss(bounce(B,Time), S) :- ball(B), falling(B, S), epsilon(E),
    dist(B,Distance,Time,S), Distance $= 0,
    vel(B,Velocity,Time,S), Velocity $>= E, start(S,T), T $=< Time.
poss(atPeak(B,Time), S) :- ball(B), flying(B, S), epsilon(E),
    dist(B,Distance,Time, S), Distance $>= E,
    vel(B,Velocity,Time, S), Velocity $=0, start(S,T), T $=< Time.
```

The first axiom is saying the agent action $drop(B, Time)$ is possible in S at the moment of time $Time$, if a ball B is neither falling, nor flying in S, and the moment of time T when situation

S started is less than or equal to $Time$. We add the temporal constraint $T\$=<Time$ to a special data structure that represents a constraint store to be evaluated later at run time when the planner checks whether the goal logical conditions are satisfied. (In particular, this means that scheduling, i.e., assignment of time moments to actions, happens once the plan skeleton has been already computed.) The last axiom is saying that a natural event $atPeak(B, T)$ can occur in S at the moment $Time$ if B is flying in S so that it reached its highest point at which its velocity is 0, but its hight is positive, and time T when S started is less than or equal to $Time$. This PA adds several more numerical constraints on the variables to the constraint store. Our planner makes sure that natural events must be executed as soon as they are possible: nature cannot wait. This is simply implemented by making sure that the planner chooses first a natural action, if there is one possible, else it can choose a possible agent action. In other words, the planner cannot do any agent actions, as long as there are possible natural actions. Implicitly, this imposes a limit on the hybrid domains where this planner can work, since it is easy to fabricate an artificial counter-example with repeatedly possible natural actions such that the planner can never do a required agent action that would solve a problem. Thus, this version is an incomplete planner.

Since we are dealing with hybrid systems, we have to use a language that can capture their semantics, and it was argued that HTSC serves this purpose well [9, 10]. In addition to all the usual axioms in BAT, HTSC introduces State Evolution Axioms (SEA) for *temporal* fluents. These SEA describe how numerical values of temporal fluents evolve across time, inside situation, similar to the idea of continuous state evolution from hybrid automata. Recall that in HTSC one can model temporal change in between actions/events using domain specific temporal fluents.

In our example, we have three possible contexts, one where the ball is at rest, one where it is falling down, and one where the ball is flying up after it bounced from the floor. An agent may drop the ball, initiating the falling process, where the temporal fluents, velocity and distance, evolve according to the standard equations of motion. Upon contact with the floor, i.e. when distance is zero, the ball bounces and enters a different flying context with its own set of the motion equations for velocity and distance. The following SSA characterize how the actions can switch between different contexts.

```
falling(B, [A|S]):- A=drop(B,T). /* Action drop makes falling true */
falling(B, [A|S]):-A=atPeak(B,T). /*Action atPeak makes falling true*/
falling(B, [A|S]):- not A=catch(B,T), not A=bounce(B,T), falling(B,S).

flying(B, [A|S]):- A=bounce(B,T).  /*Action bounce makes flying true*/
flying(B, [A|S]):- not A catch(B,T), not A=atPeak(B,T), flying(B,S).
```

In the third, persistence (default) rule, if the ball is falling in situation S, it will be also falling in the situation $[A \mid S]$ that results from executing any action A in situation S, unless A is either bouncing or catching action for this ball B. The last persistence (inertia) rule has similar purpose.

Note that when actions occur, the temporal change can be either continuous, or there might be jumps or resets in the values of temporal fluents. To describe these smooth transitions or jumps at the moments when new situation starts, we use auxiliary fluents $init\_dist(B, D, S)$ and $init\_vel(B, V, S)$. The SSA for the former is saying that the height of the ball changes continuously, no matter what actions or events happen. However, the velocity of the ball resets to 0, when the agent catches the ball. When the ball bounces, its velocity jumps to the quantity

with the opposite sign. All other actions with any other balls have no effect on these physical quantities at the moment when new situation starts.

```
init_dist(B,Dist,[Act | S]) :- time(Act,T), dist(B,Dist,T,S).

init_vel(b1,0,[ ]).      init_vel(b2,0,[ ]).
init_vel(B,Vel, [Act | S]) :- Act=catch(B,T), Vel $= 0.

init_vel(B,NewVel, [Act | S]) :- Act=bounce(B,T), time(Act,T),
    vel(B, OldVel, T, S), NewVel $= -1 * OldVel.

init_vel(B,Vel,[Act | S]) :- time(Act,T), vel(B,Vel,T,S),
    not Act = catch(B, _), not Act = bounce(B, _).
```

In a general case, it is assumed there are finitely many mutually exclusive context conditions.If all of them are false, then the temporal fluent does not change, as for the ball that is at rest.

The evolutions of temporal fluents across time are described with the State Evolution Axioms (SEA), see details and other examples in [9, 10]. Each SEA characterizes how temporal fluent changes with time within a context determined by atemporal fluents. Each temporal fluent evolves from its initial value, determined by the corresponding $init$ atemporal fluent, at the moment when situation starts. To simplify our example let us assume that the balls move along straight lines instead of physically correct quadratic trajectories. Then, the equation for both height and velocity are linear wrt time. (Recall that the acceleration due to gravity is 9.81)

```
vel(b1, 0, 0, []).      vel(b2, 0, 0, []).
vel(B, V,T, S) :- not falling(B, S), not flying(B, S), V $= 0.
vel(B, V,T, S) :- falling(B, S),  /* 2nd context: ball is falling */
    init_vel(B, OldVel, S), start(S, T1), T $>= T1,
    V $= OldVel + 9.81*(T - T1). /* OldVel >0, ball B accelerates */
vel(B, V,T, S) :- flying(B, S),  /* 3rd context: ball is flying */
    init_vel(B, OldVel, S), start(S, T1), T $>= T1,
    V $= OldVel + 9.81*(T - T1). /* OldVel < 0, ball B decelerates */

dist(b1,100,0,[]). dist(b2,150,0,[]). /*NB: different initial heights*/
dist(B, D,T, S) :- not falling(B, S), not flying(B, S),
    init_dist(B,D,S). /*the ball at rest remains where it was before*/
dist(B, D,T, S) :- falling(B, S),   /* 2nd context: ball is falling */
    init_dist(B, OldDist, S), start(S,T1), T $>= T1,
   D $= OldDist - 0.5*9.81*(T-T1). /*D decreases if B falls down*/
dist(B, D,T, S) :- flying(B, S),   /* 3rd context: ball is flying */
    init_dist(B, OldDist, S), start(S,T1), T $>= T1,
   D $= OldDist + 0.5*9.81*(T-T1). /*D increases if B moves up*/
```

Notice that SEA introduce new equality constraints on the physical quantities represented as the arguments of temporal fluents. These constraints are collected in our constraint store. They can be evaluated when the planner checks whether a goal state was reached or not. Thanks to our modeling simplification, all the constraints on the variables are linear. Therefore, if the planning instance has an associated metric that is also a linear function of its arguments, then optimization reduces to solving the linear programming problem.

As an illustration, we solved the following planning problem for the two balls: find *the earliest moment of time* such that each ball reached its peak at least once, both balls are falling, the velocities of the two balls are equal, and their heights are also equal. Checking these goal conditions reduces to the linear programming problem that can be solved using the external solver interfaced with our PROLOG program. We solved this planning instance with an uninformed iterative deepening depth-first search (DFS) planner [57] using a computer with an Intel(R) Core(TM) i7-11700K CPU running at 3.60GHz. The planner and a problem instance have been loaded, compiled and run within ECLiPSe Constraint Logic Programming System, Version 7.0 #63 (x86_64_linux), released on April 24, 2022. This system is equipped with the EPLEX library [88] that supports external Linear Programming solvers from within ECLiPSe. Specifically, we used a CLP/CBC solver from the COIN-OR open-source project (eplex_osi). The program found a correct 8 step plan in 0.18 seconds: *[drop(b2, 0), bounce(b2, 30.581039755351682), drop(b1, 50.9683995922528), atPeak(b2, 61.162079510703364), catch(b2, 71.35575942915392), bounce(b1, 71.35575942915392), drop(b2, 91.743119266055047), atPeak(b1, 91.743119266055047)]*. Notice that this plan must be clever since the two balls had different initial heights.[1] The EPLEX library is designed to interface with linear programming solvers only. For this reason, the trajectory of the bouncing ball was approximated as a linear function of time, since otherwise a different non-linear programming external solver would be required, and EPLEX could not be used.

Note that a simple DFS planner from [57] can be adapted to solve this planning problem for bouncing balls because the goal state is encoded with the rule that calls (through EPLEX) an external linear programming solver to minimize the total time taken by the plan actions, but everything else in search is the same as in the case of solving planning problems for a BAT. However, it is clear that for solving more complex planning problems, uninformed depth-first search would not work. In the next section, we present a greedy best-first search (GBFS) planner guided with a domain independent numerical heuristic that is more suitable for hybrid systems, and that can work in the cases if temporal fluents change according to arbitrary non-linear functions of time.

In a general case, each context in HTSC corresponds to its own system of ODEs, and therefore, each context determines its own initial value problem (IVP) for the temporal fluents [93].

The constraints of a planning problem written in the language of HTSC can be formulated using the Constraint Logic Programming (CLP) framework [27, 28, 51]. They are implemented as a PROLOG program, where constraints are usually included into the precondition axioms for actions, the state evolution axioms for temporal fluents and the problem objective function.

We would like to emphasize that we are aware of previous research that employed CLP in the analysis of hybrid systems, e.g., see the papers [95, 45, 49] and the references there. That earlier research focused on timed automata, or timed push-down automata that take as input infinite stream of signals, or worked only with hybrid automata where continuous change is linear over time, but the previous papers did not consider near-optimal planning over relational non-linear hybrid systems, as we do here.

[1]The source code of this example is available at https://www.cs.torontomu.ca/~mes/publications/

3. Methodology

The *NEAT* planner is a deductive planner that does heuristic search over situations in a situation tree, rather than over state space. It is a planner somewhat similar to the planner that is described in [92], but the planner described there is based on progression, and it can do A^* search, while *NEAT* solves the projection problem using regression, and it does GBFS rather than A^* search. Our *NEAT* planner implemented in PROLOG builds on an implementation [91], and it reasons only about the (logical) atemporal fluents. In order to reason about numerical constraints on temporal fluents, we use an external efficient Non-Linear Programming (NLP) solver, and hence we rely on the semantics of CLP [51, 64].

Since EPLEX library provides an interface to only linear programming solvers, but we explored the benchmarks with non-linear functions, the *NEAT* planner required a different library AMPLEX that is described in [66]. The AMPLEX interface has been developed similarly to EPLEX [88]. More specifically, AMPLEX collects inequality and equality constraints as strings and passes them to the well-known AMPL software [35, 3]. AMPL software does preprocessing of the input and produces a request for solving a non-linear programming problem (NLP) to a specialized mathematical programming solver. In our experiments, AMPLEX together with AMPL delegate the NLP problems from the planner to the state-of-the-art proprietary NLP solver *Knitro* developed by Artelys [18, 4]. Note that the *NEAT* planner has to solve a NLP problem on every step of its planning process. Namely, whenever the *NEAT* planner selects the next action to execute, it has to check whether the goal state is reached, and this involves sending the corresponding NLP problem to the external solver.

Note that when *NEAT* produces a tentative sequence of actions, all their object arguments are instantiated, but (continuous) time arguments are not. Similarly, the values of domain specific temporal fluents and other continuous change related variables participating in the constraints remain to be determined. Given a tentative plan skeleton with uninstantiated values for the action timestamps, the external NLP solver finds a near-optimal solution (if it exists) that satisfies all constraints. Thanks to this approach, we do not need to discretize time or any other physical quantities. Their values are computed by an external NLP solver as a result of solving an optimization problem. In the current version of *NEAT*, we use only one metric related to finding the shortest total time for reaching a goal state, but other objectives can be easily accommodated as well. Since our planner utilizes PROLOG's built-in resolution and substitution algorithms to instantiate possible actions at run time, it can work without producing first a (potentially large) instantiated transition system before searching for a reachable sequence of actions. Thus, we can do *lifted planning* that works at the level of action schemas. Notice that our approach significantly differs from the Answer Set Planning [6, 90], because we do not need to perform grounding of fluents and actions in advance. (However, we can do conformant planning without CWA similarly to [34, 8]. Exploring this in the context of hybrid systems is future work.)

Our *NEAT* planner is based on the assumption that there is proper alignment between the numerical goal conditions and the logical goal conditions. Namely, we assume that if the numerical goal conditions are true, then the system reached the goal state and the logical goal conditions are satisfied. This assumption holds for the most of the benchmark domains. Therefore, in the practical domains where the numerical and logical conditions are well aligned, solving an optimization problem is sufficient for solving the planning problem.

To guide search we employ a novel NLP heuristic that computes an approximation of the objective function for the original planning problem. This heuristic is non-trivial, and it builds on advanced mathematical research in the area of optimal control. More specifically, our heuristic function follows the approach proposed by Revaz Gamkrelidze to establish existence of solutions to an extended class of optimal control problems [39]. Gamkrelidze called his approach to control as “sliding modes" or “sliding regimes". This approach is also known as convexification, or as relaxed controls, generalized controls, or as extended controls because all individual changes slide together as a bundle. Gamkrelidze described his approach in the textbook [38] that is based on his lectures at the Tbilisi State University (in Georgia). Later, it was understood that his approach is essentially equivalent to an approach independently developed by L.C. Young and E.J. McShane, e.g., see L.Cesari’s notes to the first chapter in [24]. It turned out, that Gamkrelidze’s approach has strong kinship with intuition that motivated [14].[2] The interested readers can find a related detailed discussion in [5]. Informally, our heuristic evaluates each action in terms of its contribution to the task of reaching the goal state as soon as possible. If the tentatively chosen action is the right thing to do, then the system can reach the goal sooner, rather than later. The heuristic imagines that all processes can slide altogether and assigns weights to their effects. Then, the heuristic computes the values of all temporal fluents in several discrete moments of time, and thereby it produces a NLP that is delegated to an external NLP solver. It returns an approximation to the minimal time (required to reach the goal state) that determines priority of a chosen action in the priority queue. Since we assume that the numerical and logical goal conditions are properly aligned, by evaluating the minimal time required to reach the numerical goal conditions, our heuristic estimates how well each of the next possible actions contributes to reaching the goal state sooner.

In contrast to the previous hybrid system planners, our domain independent heuristic does not impose any limitations on the type of non-linear functions that can be considered. Therefore, *NEAT* can do informed best-first greedy search even if temporal fluents change according to non-polynomial functions such as $sin(x), cos(x), log(x), e^x, \sqrt{x}$, etc. Our preliminary experiments with PDDL+ benchmarks show that our approach is comparable with existing model-based and SAT-based approaches to temporal numeric planning. This is discussed in the next section. Recall that the PDDL [46], the Planning Domain Description Language, was created to standardize input to the automated AI planners, PDDL 2.1 enhanced PDDL with numerical fluents and durative actions [36], and PDDL+ is an extension to deal with mixed discrete-continuous domains such as hybrid systems [37, 10].

Initially, we had developed an automated translator in Python from PDDL+ benchmarks [52] to HTSC following the semantics in [10]. Later, we have changed our implementation of HTSC and realized that there are less than 10 popular PDDL+ benchmarks that are somewhat weak and limited. Therefore, we decided to abandon automatic translation. Nevertheless, our HTSC implementations of the PDDL+ domains and planning instances are faithful and direct translations of the original PDDL+ benchmarks with a few exceptions that we explain below. No additional

[2]Nikolay Bogolyubov (also transliterated as Bogoliubov or Bogoliouboff) published [14] when he was 20, soon after completing his PhD in Kyiv University in 1928. His paper impressed Leonida Tonelli, an Italian mathematician who contributed to the calculus of variations, so that Bogolyubov was awarded the Bologna Academy of Sciences Prize and the degree of doctor of mathematics in 1930. See the details in the Wikipedia article https://en.wikipedia.org/wiki/Nikolay_Bogolyubov

information was encoded in the process of manual translation of the domain into HTSC, and hence there is no performance advantage gained over an automatic translation of the domain. Moreover, it is well-known that PROLOG has a built-in order of executing clauses from the top to the bottom, and this applies to the precondition axioms in the aforementioned DFS planner. However, in the case of *NEAT*, this order has no effect on the performance of our planner, since when a situation is removed from the priority queue, we compute *all* actions that are possible in that situation, compute heuristic values for *all* successors, and insert them into the priority queue. Therefore, only heuristic priorities of situations determine which situation will be expanded next, not the order of clauses in our program, and not the order of inserting situations into the priority queue. Similar remarks apply to computing the values of temporal fluents inside the heuristic function: rearranging the SEAs in the program differently would have no effect on run time.

There are the following limitation of the current version of *NEAT*. First, the PDDL+ benchmarks use the construct $\#t$ [10, 46] to formulate ODEs, and the implementations of PDDL+ planners do pre-processing of a few built-in simple ODEs to match their syntactic expressions with the corresponding functions of time. Instead, we opted to solve each benchmark ODE manually and encode the resulting function directly in our SEAs for temporal fluents. We leave the task of solving a few classes of ODEs (that can be symbolically solved using algebraic techniques as in [80]) to future work. Second, the current version of *NEAT* does not implement the PDDL+ "overall" construct that imposes global numerical constraints that must hold over an interval of time while a durative action is being executed. This is not a significant limitation for the benchmarks that we considered, since we encoded the required constraints in the PAs of the agent actions. A domain independent implementation of global numerical constraints is left to future.

4. Experimental Results

We perform direct comparisons between our planner, *NEAT* [66], and the existing state-of-the-art non-linear temporal numeric planners for which open source implementations are readily available, namely, *DiNo* [77], *ENHSP* [86, 85] and *SMTPlan+* [21, 22]. These planners demonstrate markedly better performance than their counterparts on a number of benchmarks [52]. We also discuss an indirect comparison between *NEAT* and two other planners, *dReach* [53, 17] and *CASP* [6]. Both *DiNo* and *ENHSP* are model-based planners that discretize time and reduce the temporal numeric planning problem to a numeric planning problem (without time). Both use different heuristics to efficiently search the grounded transition graph. We run default configurations and used the 2018 release of ENHSP. *SMTPlan+* reduces the temporal numeric planning problem to a Satisfiability Modulo Theories (SMT) problem [7] and does not discretize time. No domain-independent heuristic is used apart from computational tricks already built into the underlying SMT solver Z3. *dReach* transforms the hybrid automaton representation of the planning problem into an SMT. *CASP* is a planner that reduces the temporal numeric planning problem into an answer set programming problem (ASP). The authors did not develop an automated translator from PDDL+ to ASP, and hence we were unable to perform a direct computational comparison. There are a few other well-known PDDL+ planners, e.g., [75, 15, 33, 31], that were not included into this experimental work because they were previously discussed and compared with one of *DiNo*, *ENHSP*, *SMTPlan+*, or because their source code was not readily available.

In each domain, the planners are given 30min to solve the instance and 1 GB of memory. The tests were run on a Intel(R) Core(TM) i7-3770 CPU with 3.40GHz clock speed. In terms of metrics, we focused on (1) execution time (in seconds) which describes how long the planner takes to find a plan, (2) coverage which describes how many instances were successfully solved before timing out or running out of memory, (3) plan duration (in time units) which is a measure of plan optimality, namely, how long the produced plan takes to execute from start to finish, and sometimes, (4) plan length which is the number of steps in the plan.

For direct comparisons between *DiNo*, *ENHSP* and *SMTPlan+*, we ran our tests on the standard set of PDDL+ benchmarks [52]: Linear Generator, Car, and Nonlinear Solar Rover. These benchmark domains are extensively discussed in [52, 77, 22].

The Linear Generator domain consists of a single generator with an initial fuel level and several refueling tanks of fixed volume that can be used only once. A satisfying plan must ensure that the generator has sufficient fuel to run for 1000 time units at a rate of 1 liter per time unit by scheduling refueling actions. We generate 50 instances of the domain with decreasing initial levels of fuel by units of 20 in the generator and an increasing number of tanks by units of 1, starting from 980 units of fuel and a single tank, respectively.

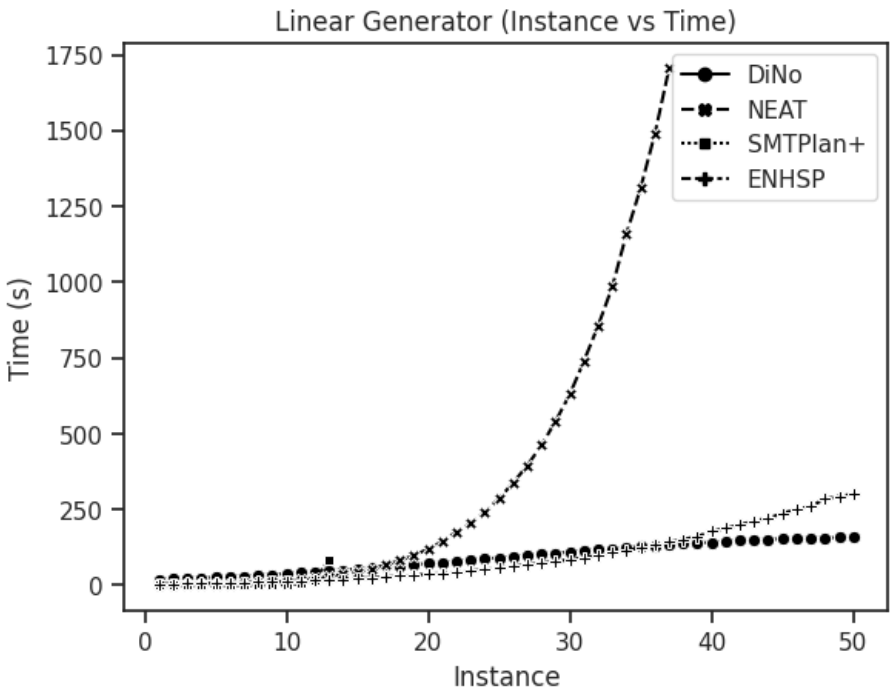

Figure 2: Graph of planner execution times on the linear generator domain

On the Linear Generator domain, our planner achieves better performance than *SMTPlan+* solving 37/50 benchmark instances, while *SMTPlan+* solved only 13 out of 50 instances. *NEAT* is outcompeted by *DiNo* ($\epsilon = 0.1$, $T = 10000$) and *ENHSP* ($\epsilon = 1$) that solved 50/50 of the planning instances, albeit with a coarse time discretization. Looking at the data collected in

Instance	1	5	10	15	20	25	30	35	40	45	50
DiNo	19.60	25.76	37.81	54.01	71.83	90.76	109.81	126.80	140.82	152.64	159.90
ENHSP	3.60	6.89	12.33	22.13	36.30	57.12	84.80	123.48	178.02	237.27	300.50
SMTPlan+	0.03	0.07	1.82	-	-	-	-	-	-	-	-
NEAT	0.57	2.96	13.91	44.27	119.63	285.67	630.29	1312.26	-	-	-

Table 1
Linear Generator Results - Instance to Execution Time (sec)

[6], *CASP* is only able to solve 8/50 benchmark instances. According to [21], *dReach* performs poorly and it is outcompeted by *SMTPlan+* in every benchmark. Analyzing Figure 2, we see that *DiNo* and *ENHSP* scale linearly as the number of tanks increases. It should be noted that manual intervention with the discretization factor was required for *ENHSP* to produce a plan. The linear performance can be attributed to *DiNo's* symmetry-breaking techniques in the model-checker. In symmetry-breaking, an order upon the actions is imposed - greatly reducing the number of possible actions in a state and hence this significantly reduces the search space. *SMTPlan+* also uses symmetry-breaking techniques in the SMT solver, but it solves few instances. Likely, due to grounding, the planner is unable to scale up. By contrast, the performance of *NEAT* is somewhere in the middle between the non-discretizing and discretizing planners in terms of coverage, but it exhibits exponential growth, unlike *DiNo*, when the number of refueling tanks increases. The reason for this is twofold. First of all, the current implementation of *NEAT* does not employ any symmetry-breaking techniques, and therefore it has to reason about the growing sequence of actions in each situation as the number of tanks increases. Secondly, the current implementation of the *Poss* predicate is simple, and it does not scale up as the sequence of actions gets longer.

We use Cashmore's variant of the Car domain found in [21], where a car must travel a specified distance of at least 30 units in as short a time as possible without overheating the engine that happens if the car exceeds a velocity threshold. This is done by accelerating or decelerating the car in a timely manner. We generate 50 instances of the domain with varying upper and lower bounds for integer acceleration and deceleration, starting at 1 and -1, respectively. While this domain can be trivially solved with a single accelerate action followed by a single decelerate action, solving the problem optimally requires scheduling multiple accelerate actions, and then gradually decelerating to arrive at a goal distance of 30 units with zero velocity at the shortest possible time. We note that there is another non-linear version of this domain with an additional wind resistance process that has a non-linear effect on the velocity of the car. We report here the results collected from a simplified version found in [21], without the wind resistance process.

The HTSC axioms for the Car domain are provided in [10]. Since the SEAs require there are only finitely many temporal change axioms, but potentially there are infinitely many accelerations, we take advantage of the upper and lower limits on acceleration that are present in each PDDL+ instance. Namely, we encode in each instance that there are finitely many integer acceleration values available and enforce that there might be only one moving process at a time with a specific acceleration. Whenever the planner does an accelerate or a decelerate action, a natural action must occur next that ends the moving process with the previous acceleration, and then another natural action must occur that starts a new moving process with the new acceleration value. The precondition axioms make sure that only this sequence of natural actions is possible. This is similar to the distinction between the agent and natural actions in the bouncing ball example.

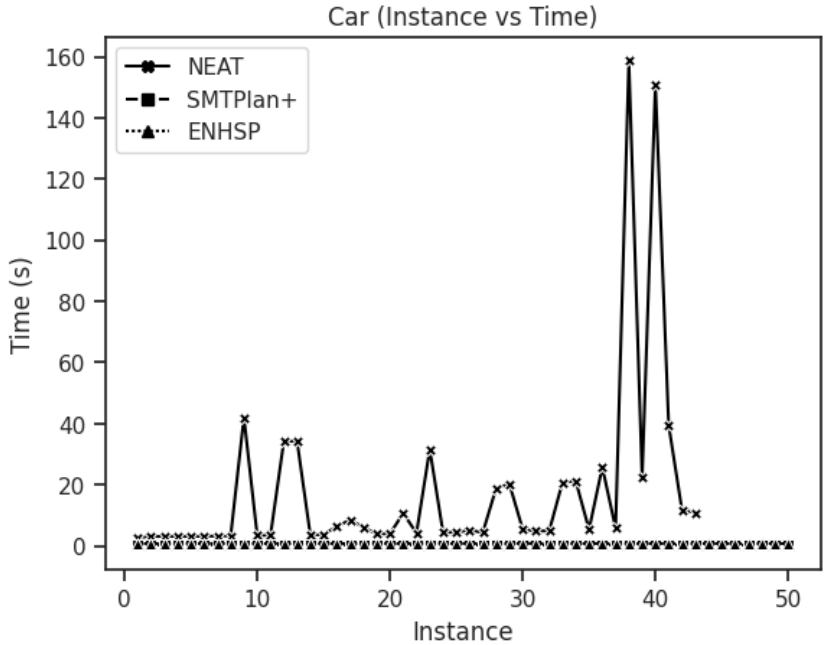

Figure 3: Graph of planner execution times on the car domain

Instance	1	5	10	15	20	25	30	35	40	45	50
DiNo ($\epsilon = 0.1$)	-	-	-	-	-	-	-	-	-	-	-
ENHSP ($\epsilon = 0.1$)	-	-	-	-	-	-	-	-	-	-	-
ENHSP ($\epsilon = 1$)	0.71	0.84	0.75	0.75	0.79	0.72	0.88	0.72	0.82	0.73	0.74
SMTPlan+	0.065	0.065	0.065	0.065	0.065	0.065	0.065	0.065	0.065	0.065	0.065
NEAT	2.78	3.13	3.38	3.72	3.95	4.48	5.44	5.36	150.86	-	-

Table 2
The results for the Car domain - Instance to Execution Time (sec)

In this second benchmarking domain, Car, the discretizing planners, *DiNo* ($\epsilon = 0.1$, $T = 10000$) and *ENHSP*, do not perform well and aren't able to solve any of the benchmark instances with a discretization factor of 0.1. ENHSP ($\epsilon = 1$) solved all 50 instances but with a coarse discretization 1. *SMTPlan+* performs very well and is able to solve all instances of the benchmark. However, *SMTPlan+* produces the same short plan to trivially satisfy all instances, while longer plans could be used to reach the goal in a shorter amount of time. More specifically, *SMTPlan+* finds a plan that accelerates the car once at the beginning, and then one time unit later decreases acceleration, so that the car travels with a constant velocity of 1 per time unit before it reaches the required distance of 30 units, taking in total 32 time units. Both *ENHSP* and *NEAT* find plans of shorter duration, and *NEAT* finds the fastest plans. Looking at the asymptotic behavior of *CASP* on the Car domain, we see that the time to solve remains constant. Therefore, we believe it would have been able to solve all instances of the Car domain. However, no comments about the optimality of the plans were made by the authors. Our planner can solve 43/50 instances, but produces plans of better quality in terms of optimality than all other planners. Namely, it

finds a plan that does several accelerate actions consecutively before it does several decelerate actions. This produces plans that have a total duration of less than 9 time units. From Figure 3, we see that *SMTPlan+'s* time to solve remains constant, which is expected since *SMTPlan+* first evaluates if all shorter plans are feasible solutions before attempting longer ones. This is at the cost of optimality, unlike *NEAT*, which exhibits very slow linear growth, but times out at instance 44 because the heuristic evaluation function resulted in *NEAT* choosing an erroneous and excessively long action sequence. *ENHSP* with a coarse discretization also has a time to solve that remains constant, this is because its heuristic results in it choosing the same plan regardless of the planning instance even if more optimal plans are available. It should be noted that when tested with an upper bound 15 on the plan's length, *NEAT* is able to solve all instances of the domain in constant time like *SMTPlan+* and *ENHSP*.

Instance	1	5	10	15	20	25	30	35	40	45	50
DiNo	26.75	64.39	112.18	157.43	205.48	252.78	300.95	352.44	398.26	443.26	485.44
NEAT	4.41	6.95	8.53	4.75	4.31	4.02	4.24	3.31	3.96	4.20	4.28

Table 3
Non-linear Solar Rover Results - Instance to Execution Time (sec)

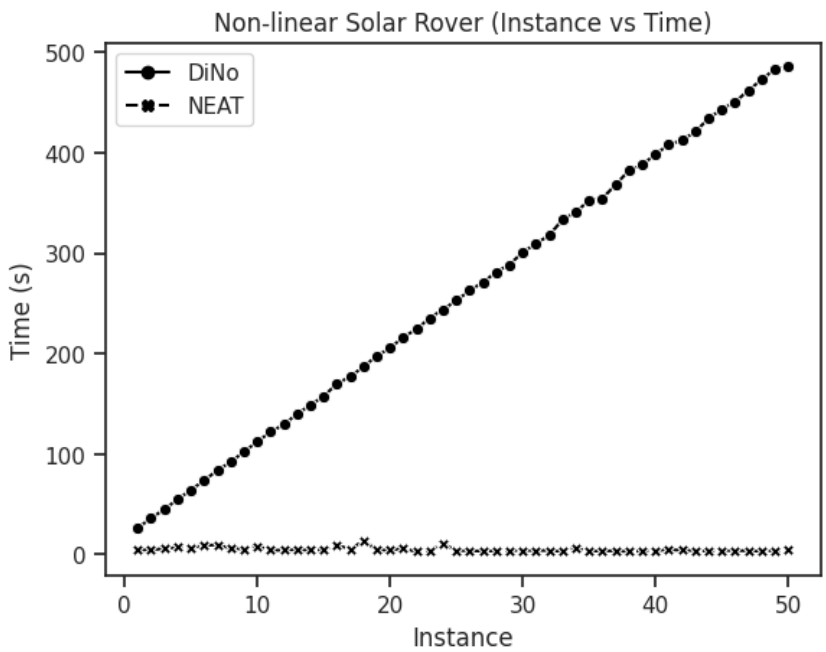

Figure 4: Graph of planner execution times on the non-linear solar rover domain

In the last domain, a non-linear Solar Rover, a rover must wait for the sun to rise in order to charge its main solar battery or drain backup batteries to increase the charge of the main battery. Once a certain level of charge is reached for the main battery, the rover is able to complete its mission and send data without depleting its energy reserves. The solar charging process is governed by an ODE that is dependent on the initial state of charge of the battery, and thus

optimal plans should include draining the backup batteries, in order to reach the required charge level as quickly as possible.

We generated 50 instances with different times for when the sun rises, initially starting at 50 time units and increasing by increments of 50 up to 2500 time units. Our planner can solve all instances. Out of the discretizing planners, only *DiNo* ($\epsilon = 0.1$, $T = 10000$) can solve all instances, but it doesn't produce optimal plans. *SMTPlan+* is unable to solve any of the instances because it is unable to model continuous change that isn't polynomial over time during charging process and throws an error on the linear variant of the benchmark. ENHSP was incapable of compiling the planning domain file. No benchmark data are available for *dReach* and *CASP*. Looking at Figure 4, we see that as the instance number increases, *DiNo* takes a longer amount of time to solve each problem instance, increasing linearly with the instance number, unlike *NEAT* which takes a constant amount of time to solve the problem regardless of the time horizon.

From these benchmarks, we see that our *NEAT* planner performs well in domains where there are several objects, competing well with *SMTPlan+*, *dReach* and *CASP*. Our planner compares well with *DiNo* and *ENHSP* in the domains where plan optimality was important. Finally, note that this version of the *NEAT* planner was implemented in a few months by a Master student [66]. In a recent simplified version of *NEAT*, the total code size is relatively small. Namely, the planner and heuristic together have less than 50 PROLOG rules and make about 150 calls to predicates in total, i.e, the program is short and elegant.

5. Discussion and Future Work

Recently, we learned about research related to timeline-based planning, e.g., see [25, 12]. Since the underlying framework is very different from HTSC, it may take some time to research the conceptual differences between *NEAT* and timeline-based planning, and to understand if there are common computational intuitions behind the heuristics.

Since our planner checks goal conditions for each new situation visited, it returns a correct plan for the benchmarks that we explored. It is doable to prove formally that our planner is sound under the reasonable assumptions. Completeness cannot be attained in general, due to undecidability of the reachability problem in non-trivial hybrid systems. It is interesting and important question how closely and under what conditions our planner can approximate optimal plans on realistic benchmarks.

Since only one NLP solver *Knitro* was explored in [66], the natural question is how the performance of the planner will be affected by using alternative NLP solvers. Moreover, the planner submits incrementally growing NLPs to the external solver, and if the solver can warm-start in the same region that has been already constructed from the previous set of constraints, then the time to answer repeatedly growing NLP queries could be significantly reduced. We are exploring both these directions, and they remain important directions of the near-term research.

Since our approach does not require grounding before planning can start, it is our hypothesis that our approach can scale up to compute near-optimal plans for larger relational hybrid systems. This hypothesis can be verified by exploring the unit commitment problem instances with a few 100s of objects. There are several large benchmark instances for the unit commitment problem. There is a related experimental research on developing more informative and less limited

benchmarks, and we hope that [73] will serve as the starting point to address this challenge.

We are aware that the Operations Research (OR) community usually solves the large-scale optimization problems for hybrid systems by modelling the problem as a Mixed Integer Linear Programming problem (MILP) [62, 100], and then using approximations of feasible solutions to find an optimum. Often, MILP is chosen over Mixed Integer Non-Linear Programming (MINLP) [11, 98], since MILP scales up better than MINLP. However, because the MILP formulation fails to capture the underlying non-linear dynamics, the solutions are not optimal.

It is our hypothesis that our approach can achieve better performance on such domains. Our reasoning is as follows. We do *not* discretize time in contrast to MILP-based methods which allows us to accurately represent the continuous evolution of the system. Additionally, we represent faithfully the non-linear dynamics of the system which results in a better objective function cost. Lastly, to deal with combinatorics of a large search space MILP methods employ somewhat crude approximation heuristics based on the problem's underlying numerical structure that does not preserve the meaning of the original hybrid system, and they disregard the semantics of the underlying dynamical system. In contrast, a theorem-based proving approach like *NEAT* captures well this semantics thanks to CLP.

In conclusion, we remind that there were previous publications about potential relative advantages of Constraint Programming (CP), and specifically CLP, over OR modelling and solution methods, e.g., see [43, 89], as well as proposals to integrate advantages of logic based methods with OR [50, 71, 72, 99]. However, the previous work considered CP and CLP broadly, while we focus specifically on using CLP for near-optimal planning in relational hybrid systems.

Acknowledgements. Thanks to the Natural Sciences and Engineering Research Council (NSERC) of Canada and the Faculty of Science (TMU) for providing partial funding of research. Thanks to the reviewers for useful comments to a preliminary version of this paper.

6. Appendix: Bouncing Ball Example

1. Precondition Axioms

   a) Action $drop$

$$\forall s \forall t \forall b.\ poss(drop(b,t),s) \leftrightarrow ball(b) \wedge \neg falling(b,s) \wedge \neg flying(b,s) \wedge t \geq start(s).$$

   b) Action $catch$

$$\forall s \forall t \forall b.\ poss(catch(b,t),s) \leftrightarrow ball(b) \wedge (falling(b,s) \vee flying(b,s)) \wedge t \geq start(s).$$

   c) Action $bounce$

$$\forall s \forall t \forall b.\ poss(bounce(b,t),s) \leftrightarrow ball(b) \wedge distance(b,t,s) = 0 \wedge \\ velocity(b,t,s) \geq \epsilon \wedge falling(b,s) \wedge t \geq start(s).$$

   d) Action $atPeak$

$$\forall s \forall t \forall b.\ poss(atPeak(b,t),s) \leftrightarrow ball(b) \wedge distance(b,t,s) \geq 0 \wedge \\ velocity(b,t,s) = 0 \wedge flying(b,s) \wedge t \geq start(s).$$

2. Successor State Axioms for Relational Atemporal Fluents

a) Atemporal fluent $falling$

$$(\forall a \forall s \forall b).\ falling(b, do(a, s)) \leftrightarrow \exists t(a = drop(b, t)) \vee \exists t(a = atPeak(b, t)) \vee (\neg \exists t(a = catch(b, t)) \wedge \neg \exists t(a = bounce(b, t)) \wedge falling(b, s))$$

b) Atemporal fluent $flying$

$$(\forall a \forall s \forall b).\ flying(b, do(a, s)) \leftrightarrow \exists t(a = flying(b, t)) \vee (\neg \exists t(a = catch(b, t)) \wedge \neg \exists t(a = atPeak(b, t)) \wedge flying(b, s))$$

3. Successor State Axioms for Initial Values

a) Fluent $init_{vel}$

$$(\forall s \forall y \forall a \forall b).\ init_{vel}(b, do(a, s)) = y \leftrightarrow \exists y_0.y_0 = velocity(time(a), s) \wedge \exists t(a = catch(b, t) \wedge y = 0) \vee \exists t(a = bounce(b, t) \wedge y = -y_0) \vee (\neg \exists t(a = bounce(b, t)) \wedge \neg \exists t(a = catch(b, t)) \wedge y = y_0).$$

b) Fluent $init_{dist}$

$$(\forall s \forall y \forall a \forall b) init_{dist}(b, do(a, s)) = y \leftrightarrow distance(time(a), s) = y$$

4. State Evolution Axioms for Temporal Functional Fluents

a) velocity

$$(\forall s \forall t \forall b).velocity(b, t, s) = y \leftrightarrow \exists y_0.y_0 = init_{vel}(b, s) \wedge (\neg falling(b, s) \wedge \neg flying(b, s) \wedge y = y_0) \vee (falling(b, s) \wedge y = y_0 + \int_{start(s)}^{t} 9.81 dx) \vee (flying(b, s) \wedge y = y_0 + \int_{start(s)}^{t} 9.81 dx).$$

b) distance

$$(\forall s \forall t \forall b).distance(b, t, s) = y \leftrightarrow \exists y_0.y_0 = init_{dist}(b, s) \wedge (\neg falling(b, s) \wedge \neg flying(b, s) \wedge y = y_0) \vee \big(falling(b, s) \wedge y = y_0 - \int_{start(s)}^{t} (9.81 \cdot x)\ dx\big) \vee \big(flying(b, s) \wedge y = y_0 + \int_{start(s)}^{t} (9.81 \cdot x)\ dx\big).$$

5. Foundational Axioms for Situations and Time (from Chapters 4 and 7 of [84]).

$\forall a_1 \forall a_2 \forall s_1 \forall s_2.do(a_1, s_1) = do(a_2, s_2) \rightarrow a_1 = a_2 \wedge s_1 = s_2$

$\forall s.\neg(s \sqsubset S_0)$

$\forall a \forall s \forall s'.s \sqsubset do(a, s') \leftrightarrow s \sqsubseteq s'$, where $s \sqsubseteq s'$ means $(s \sqsubset s' \vee s = s')$

$\forall P.(P(S_0) \wedge \forall a \forall s(P(s) \rightarrow P(do(a, s)))) \rightarrow \forall s P(s)$

$\forall a, s'.do(a, s') \sqsubseteq s \rightarrow (poss(a, s') \wedge start(s') \leq time(a)) \wedge \forall a' \big(poss(a', s) \wedge natural(a') \wedge a \neq a' \rightarrow time(a') \leq time(a)\big)$

$\forall a.\ start(do(a, s)) = time(a)$

$start(S_0) = 0$

6. Domain Specific Axioms for the Bouncing Ball Example

$\forall t, \forall b.time(drop(b, t)) = t$ $\quad$ $\forall t, \forall b.time(catch(b, t)) = t$

$\forall t, \forall b.time(bounce(b, t)) = t$ $\quad$ $\forall t, \forall b.time(atPeak(b, t)) = t.$

$\forall t \forall b.natural(atPeak(b, t))$ $\quad$ $\forall t \forall b.natural(bounce(b, t)).$

$\forall t \forall b.agent(drop(b, t))$ $\quad$ $\forall t \forall b.agent(catch(b, t)).$

7. Initial Theory

$ball(b1)$	$ball(b2)$
$velocity(b1, 0, S_0) = 0$	$velocity(b2, 0, S_0) = 0$
$distance(b1, 0, S_0) = 100$	$distance(b2, 0, S_0) = 150$

References

[1] Erika Abraham, Johanna Nellen, and Stefan Schupp. Techniques and tools for hybrid systems reachability analysis. *NSV 2017: 10th International Workshop on Numerical Software Verification 2017*, Apr 2017.

[2] 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.

[3] AMPL Optimization Inc. A Mathematical Programming Language. https://ampl.com, 2023.

[4] Artelys. Knitro User Manual. https://www.artelys.com/docs/knitro/, 2022. Accessed: 2023-08-31.

[5] E. R. Avakov and G. G. Magaril-Il'yaev. Gamkrelidze Convexification and Bogolyubov's Theorem. *Mathematical Notes*, 107:539–551, 2020.

[6] Marcello Balduccini, Daniele Magazzeni, Marco Maratea, and Emily Leblanc. CASP solutions for planning in hybrid domains. *Theory Pract. Log. Program.*, 17(4):591–633, 2017.

[7] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability - 2nd Edition*, volume 336 of *Frontiers in AI and Applications*, pages 1267–1329. IOS Press, 2021.

[8] Vitaliy Batusov. Deterministic Planning in Incompletely Known Domains with Local Effects, Master Thesis. Technical report, TMU, Toronto Metropolitan (formerly Ryerson) University, Department of Copmputer Science, 2014.

[9] Vitaliy Batusov, Giuseppe De Giacomo, and Mikhail Soutchanski. Hybrid Temporal Situation Calculus. *arXiv*, 1807.04861, 2018.

[10] Vitaliy Batusov and Mikhail Soutchanski. A Logical Semantics for PDDL+. In *the 29th Intern. Conf. on Automated Planning and Scheduling ICAPS*, pages 40–48, 2019.

[11] Pietro Belotti, Christian Kirches, Sven Leyffer, Jeff T. Linderoth, James R. Luedtke, and Ashutosh Mahajan. Mixed-integer nonlinear optimization. *Acta Numer.*, 22:1–131, 2013.

[12] Riccardo De Benedictis and Amedeo Cesta. Lifted heuristics for timeline-based planning. In *24th European Conference on Artificial Intelligence (ECAI)*, pages 2330–2337, 2020.

[13] Riccardo De Benedictis, Nicola Gatti, Marco Maratea, Aniello Murano, Enrico Scala, Luciano Serafini, Ivan Serina, Elisa Tosello, Alessandro Umbrico, and Mauro Vallati. Preface to the Italian Workshop on Planning and Scheduling, RCRA Workshop on Experimental evaluation of algorithms for solving problems with combinatorial explosion, and SPIRIT Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy (IPS-RCRA-SPIRIT 2023). In *Proceedings of the Italian Workshop on Planning and Scheduling, RCRA Workshop on Experimental evaluation of algorithms for solving problems with combinato-*

rial explosion, and SPIRIT Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy (IPS-RCRA-SPIRIT 2023) co-located with 22th International Conference of the Italian Association for Artificial Intelligence (AI IA 2023)*, 2023.

[14] Nikolay Bogolyubov. Sur quelques méthodes nouvelles dans le calcul des variations. *Annali di Matematica Pura ed Applicata*, 7:249–271, 1929.

[15] Sergiy Bogomolov, Daniele Magazzeni, Stefano Minopoli, and Martin Wehrle. PDDL+ planning with hybrid automata: Foundations of translating must behavior. In *the 25th Intern. Conf. on Automated Planning and Scheduling, ICAPS 2015*, pages 42–46, 2015.

[16] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[17] Daniel Bryce, Sicun Gao, David J. Musliner, and Robert P. Goldman. SMT-based nonlinear PDDL+ planning. In B. Bonet and S. Koenig, editors, *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 3247–3253. AAAI Press, 2015.

[18] Richard H. Byrd, Jorge Nocedal, and Richard A. Waltz. *Knitro: An Integrated Package for Nonlinear Optimization*, pages 35–59. Springer, 2006.

[19] Diego Calvanese, Giuseppe De Giacomo, and Mikhail Soutchanski. On the undecidability of the situation calculus extended with description logic ontologies. In *the 24h International Joint Conference on Artificial Intelligence, IJCAI-2015*, pages 2840–2846, 2015.

[20] Joshua Campion, Chris J. Dent, Maria Fox, Derek Long, and Daniele Magazzeni. Challenge: Modelling unit commitment as a planning problem. In *23rd Intern. Conf. on Automated Planning and Scheduling, ICAPS 2013*, 2013.

[21] Michael Cashmore, Maria Fox, Derek Long, and Daniele Magazzeni. A compilation of the full PDDL+ language into SMT. In *the 26th International Conference on Automated Planning and Scheduling, ICAPS 2016*, pages 79–87, 2016.

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

[23] Anya Castillo, Carl Laird, César A. Silva-Monroy, Jean-Paul Watson, and Richard P. O'Neill. The Unit Commitment Problem With AC Optimal Power Flow Constraints. *IEEE Transactions on Power Systems*, 31(6):4853–4866, 2016.

[24] Lamberto Cesari. *Optimization–Theory and Applications: Problems with Ordinary Differential Equations*. Springer, 2012.

[25] Amedeo Cesta, Alberto Finzi, Simone Fratini, Andrea Orlandini, and Enrico Tronci. Analyzing flexible timeline-based plans. In *19th European Conference on Artificial Intelligence (ECAI-2010)*, pages 471–476. IOS Press, 2010.

[26] Pieter Collins. Model checking dynamical systems. *Nieuw archief voor Wiskunde*, 5/17(3):214–220, September 2016.

[27] Alain Colmerauer. Equations and inequations on finite and infinite trees. In *Proc. of the Intern. Conf. on Fifth Generation Computer Systems, FGCS 1984*, pages 85–99, 1984.

[28] Alain Colmerauer. An introduction to prolog III. *Commun. ACM*, 33(7):69–90, 1990.

[29] Augusto B. Corrêa, Guillem Francès, Florian Pommerening, and Malte Helmert. Delete-relaxation heuristics for lifted classical planning. In *the 31st International Conference on Automated Planning and Scheduling, ICAPS 2021*, pages 94–102, 2021.

[30] Joaquim Ortiz de Haro, Erez Karpas, Michael Katz, and Marc Toussaint. A conflict-driven interface between symbolic planning and nonlinear constraint solving. *IEEE Robotics*

Autom. Lett., 7(4):10518–10525, 2022.

[31] Elad Denenberg and Amanda Jane Coles. Mixed discrete continuous non-linear planning through piecewise linear approximation. In *the 29th International Conference on Automated Planning and Scheduling, ICAPS 2019*, pages 137–145, 2019.

[32] Laurent Doyen, Goran Frehse, George J. Pappas, and André Platzer. Verification of hybrid systems. In *Handbook of Model Checking*, pages 1047–1110. Springer, 2018.

[33] Enrique Fernández-González, Brian C. Williams, and Erez Karpas. Scottyactivity: Mixed discrete-continuous planning with convex optimization. *J. Artif. Intell. Res.*, 62:579–664, 2018.

[34] Alberto Finzi, Fiora Pirri, and Ray Reiter. Open world planning in the situation calculus. In *the 7th Conference on Artificial Intelligence (AAAI-00)*, pages 754–760, 2000.

[35] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Scientific Press series. Thomson/Brooks/Cole, 2003.

[36] Maria Fox and Derek Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.*, 20:61–124, 2003.

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

[38] R. Gamkrelidze. *Principles of Optimal Control Theory*. Mathematical Concepts and Methods in Science and Engineering. Springer, 2013.

[39] R. V. Gamkrelidze. On sliding optimal states. *Doklady Akademii Nauk SSSR*, 143:1243–1245, 1962.

[40] Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. δ-complete decision procedures for satisfiability over the reals. In *Automated Reasoning - 6th Intern. Joint Conference, IJCAR*, volume 7364 of *Lecture Notes in Computer Science*, pages 286–300. Springer, 2012.

[41] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *CoRR*, abs/2010.01083, 2020.

[42] Hector Geffner and Blai Bonet. A Concise Introduction to Models and Methods for Automated Planning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 7(2):1–141, 2013.

[43] Carla P. Gomes. Artificial intelligence and operations research: challenges and opportunities in planning and scheduling. *Knowl. Eng. Rev.*, 15(1):1–10, 2000.

[44] Claude Cordell Green. *"The Application of Theorem Proving to Question-Answering Systems"*. PhD thesis, Stanford University, https://www.kestrel.edu/home/people/green/publications/green-thesis.pdf https://en.wikipedia.org/wiki/Cordell_Green, 1969.

[45] Gopal Gupta and Enrico Pontelli. A constraint-based approach for specification and verification of real-time systems. In *the 18th IEEE Real-Time Systems Symposium (RTSS)-1997*, pages 230–239, 1997.

[46] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2019.

[47] Patrick J. Hayes. Computation and deduction. In *Mathematical Foundations of Computer Science: Proceedings of Symposium and Summer School, Strbské Pleso, High Tatras, Czechoslovakia, September 3-8, 1973*, pages 105–117. Mathematical Institute of the

Slovak Academy of Sciences, 1973.

[48] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *J. Comput. Syst. Sci.*, 57(1):94–124, 1998.

[49] Timothy J. Hickey and David K. Wittenberg. Rigorous modeling of hybrid systems using interval arithmetic constraints. In Rajeev Alur and George J. Pappas, editors, *Hybrid Systems: Computation and Control, 7th International Workshop HSCC-2004*, volume 2993 of *Lecture Notes in Computer Science*, pages 402–416. Springer, 2004.

[50] J. Hooker. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 2000.

[51] Joxan Jaffar, Michael J. Maher, Kim Marriott, and Peter J. Stuckey. The semantics of constraint logic programs. *J. Log. Program.*, 37(1-3):1–46, 1998.

[52] King's College London Planning Group. GitHub Repository of PDDL+ Benchmarks. https://github.com/KCL-Planning/DiNo/tree/master/ex, 2016.

[53] Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. dReach: δ-Reachability Analysis for Hybrid Systems. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 200–205. Springer, 2015.

[54] Robert Kowalski. Logic programming. In *Handbook of the History of Logic: Computational Logic*, volume 9, pages 523–569, 2014.

[55] Robert A. Kowalski. The early years of logic programming. *Commun. ACM*, 31(1):38–43, 1988.

[56] Pascal Lauer, Álvaro Torralba, Daniel Fiser, Daniel Höller, Julia Wichlacz, and Jörg Hoffmann. Polynomial-time in PDDL input size: Making the delete relaxation feasible for lifted planning. In Zhi-Hua Zhou, editor, *Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI 2021*, pages 4119–4126. ijcai.org, 2021.

[57] Hector J. Levesque. *Thinking as Computation: A First Course*. MIT Press, 2012.

[58] Fangzhen Lin and Raymond Reiter. State constraints revisited, available at http://www.cs.toronto.edu/cogrobo/Papers/constraint.pdf. *J. Log. Comput.*, 4(5):655–678, 1994.

[59] Hai Lin and Panos J. Antsaklis. *Hybrid Dynamical Systems: Fundamentals and Methods*. Springer, 2022.

[60] Jianfeng Liu, Carl D. Laird, Joseph K. Scott, Jean-Paul Watson, and Anya Castillo. Global solution strategies for the network-constrained unit commitment problem with ac transmission constraints. *IEEE Transactions on Power Systems*, 34(2):1139–1150, 2019.

[61] J.W. Lloyd. *Foundations of Logic Programming*. Symbolic Computation. Springer, 2nd edition, 2012.

[62] Andrea Lodi. Mixed integer programming computation. In *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pages 619–645. Springer, 2010.

[63] Matthew R. Maly, Morteza Lahijanian, Lydia E. Kavraki, Hadas Kress-Gazit, and Moshe Y. Vardi. Iterative temporal motion planning for hybrid systems in partially unknown environments. In C. Belta and F. Ivancic, editors, *Proceedings of the 16th Intern. Conf. on Hybrid Systems: Computation and Control, HSCC 2013*, pages 353–362. ACM, 2013.

[64] Kim Marriott, Peter J. Stuckey, and Mark Wallace. Constraint logic programming. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2, pages 409–452. Elsevier, 2006.

[65] Arman Masoumi, Megan Antoniazzi, and Mikhail Soutchanski. Modeling Organic Chemistry and Planning Organic Synthesis. In *Global Conference on AI, GCAI-2015, Georgia*, volume 36 of *EPiC Series in Computing*, pages 176–195. EasyChair, 2015.

[66] Shaun Mathew. Heuristic Planning for Continuous Systems In Hybrid Temporal Situation Calculus, Master Thesis. Technical report, TMU, Toronto Metropolitan (formerly Ryerson) University, Department of Copmputer Science, Sep 2021.

[67] Rami Matloob and Mikhail Soutchanski. Exploring organic synthesis with state-of-the-art planning techniques. In *Scheduling and Planning Applications woRKshop (SPARK) at the 26th ICAPS, London, UK, June 12 - 17*, pages 52–61, 2016.

[68] John McCarthy. Situations, actions and causal laws. Technical Report Memo 2, Stanford University Artificial Intelligence Laboratory, Stanford, CA, 1963. Reprinted in Marvin Minsky, editor, Semantic Information Processing, MIT Press, 1968.

[69] John McCarthy and Patrick Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh Univ. Press, 1969.

[70] Sheila A. McIlraith. Integrating Actions and State Constraints: A Closed-form Solution to the Ramification Problem (sometimes). *Artif. Intell.*, 116(1-2):87–121, 2000.

[71] Michela Milano, editor. *Constraint and Integer Programming: Toward a Unified Methodology*. Operations Research/Computer Science Interfaces Series. Springer, 2003.

[72] Michela Milano and Mark Wallace. Integrating operations research in constraint programming. *Ann. Oper. Res.*, 175(1):37–76, 2010.

[73] MINLPLib. A Library of Mixed-Integer and Continuous Nonlinear Programming Instances, maintained by Stefan Vigerske, svigerske at gams.com . http://minlplib.org/.

[74] Chris Paxton, Nathan D. Ratliff, Clemens Eppner, and Dieter Fox. Representing robot task plans as robust logical-dynamical systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019*, pages 5588–5595, 2019.

[75] Giuseppe Della Penna, Daniele Magazzeni, Fabio Mercorio, and Benedetto Intrigila. UPMurphi: A tool for universal planning on PDDL+ problems. In Alfonso Gerevini, Adele E. Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 2009.

[76] Chiara Piacentini, Daniele Magazzeni, Derek Long, Maria Fox, and Chris Dent. Solving realistic unit commitment problems using temporal planning. In *26th Intern. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 421–430, 2016.

[77] Wiktor Mateusz Piotrowski, Maria Fox, Derek Long, Daniele Magazzeni, and Fabio Mercorio. Heuristic planning for PDDL+ domains. In *25th International Joint Conference on Artificial Intelligence, IJCAI-2016*, pages 3213–3219, 2016.

[78] Fiora Pirri and Ray Reiter. Some contributions to the metatheory of the situation calculus. *Journal of the ACM (JACM)*, 46(3):325–361, 1999.

[79] Erion Plaku, Lydia E. Kavraki, and Moshe Y. Vardi. Motion planning with dynamics by a synergistic combination of layers of planning. *IEEE Trans. Robotics*, 26(3):469–482, 2010.

[80] Andrei D. Polyanin and Valentin F. Zaitsev. *Handbook of Ordinary Differential Equations Exact Solutions, Methods, and Problems*. Chapman & Hall, 3rd edition, 2018.

[81] Stefan Ratschan. Safety verification of non-linear hybrid systems is quasi-decidable.

Formal Methods Syst. Des., 44(1):71–90, 2014.

[82] Raymond Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380, San Diego, 1991. Academic Press.

[83] Raymond Reiter. Natural actions, concurrency and continuous time in the situation calculus. In *5th Intern. Conf. on Principles of Knowledge Representation and Reasoning (KR'96)*, pages 2–13, 1996.

[84] Raymond Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT, see http://cognet.mit.edu/book/knowledge-action, 2001.

[85] Enrico Scala, Patrik Haslum, Daniele Magazzeni, and Sylvie Thiébaux. Landmarks for numeric planning problems. In *26th International Joint Conference on Artificial Intelligence, IJCAI 2017*, pages 4384–4390, 2017.

[86] Enrico Scala, Patrik Haslum, Sylvie Thiebaux, and Miquel Ramirez. Interval-based relaxation for general numeric planning. In *Proceedings of the Twenty-Second European Conference on Artificial Intelligence*, ECAI'16, page 655–663, 2016.

[87] Natarajan Shankar. Combining model checking and deduction. In *Handbook of Model Checking*, pages 651–684. Springer, 2018.

[88] Kish Shen and Joachim Schimpf. Eplex: Harnessing mathematical programming solvers for constraint logic programming. In Peter van Beek, editor, *11th International Conference on Principles and Practice of Constraint Programming - CP2005*, volume 3709 of *Lecture Notes in Computer Science*, pages 622–636. Springer, 2005.

[89] Y. Shoukry, P. Nuzzo, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada. SMC: satisfiability modulo convex programming. *Proc. IEEE*, 106(9):1655–1679, 2018.

[90] Tran Cao Son, Enrico Pontelli, Marcello Balduccini, and Torsten Schaub. Answer set planning: A survey. *CoRR*, abs/2202.05793, 2022.

[91] Mikhail Soutchanski. Planning As Reasoning in the Situation Calculus Based on Regression and Heuristics. Unpublished PROLOG source code, TMU (formerly Ryerson University), Department of Computer Science, Toronto, Canada, June 2016.

[92] Mikhail Soutchanski and Ryan Young. Planning as Theorem Proving with Heuristics. *arXiv*, 2303.13638, 2023.

[93] G. Teschl. *Ordinary Differential Equations and Dynamical Systems*. Graduate studies in mathematics. AMS, https://www.mat.univie.ac.at/~gerald/ftp/book-ode/ode.pdf, 2012.

[94] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *24th Intern. Joint Conference on Artificial Intelligence IJCAI*, pages 1930–1936, 2015.

[95] Luis Urbina. Analysis of hybrid systems in CLP(R). In *2nd International Conference on Principles and Practice of Constraint Programming*, volume 1118 of *Lecture Notes in Computer Science*, pages 451–467. Springer, 1996.

[96] Mauro Vallati, Daniele Magazzeni, Bart De Schutter, Lukas Chrpa, and Thomas McCluskey. Efficient macroscopic urban traffic models for reducing congestion: A PDDL+ planning approach. *30th AAAI Conference on Artificial Intelligence*, Mar. 2016.

[97] Maarten H. van Emden and Robert A. Kowalski. The semantics of predicate logic as a

programming language. *J. ACM*, 23(4):733–742, 1976.
[98] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.*, 106(1):25–57, 2006.
[99] H. Paul Williams. *Logic and Integer Programming*. Springer, 2009.
[100] Laurence Wolsey. *Integer Programming, 2nd Edition*. Wiley, 2021.