Available online at www.sciencedirect.com
SCIENCE DIRECT®
Artificial Intelligence 166 (2005) 194–253
Artificial Intelligence
www.elsevier.com/locate/artint

Processes and continuous change in a SAT-based planner

Ji-Ae Shin *, Ernest Davis

Courant Institute, New York University, New York, NY 10012, USA

Received 27 September 2004; accepted 6 April 2005

Available online 10 May 2005

Abstract

The TM-LPSAT planner can construct plans in domains containing atomic actions and durative actions; events and processes; discrete, real-valued, and interval-valued fluents; reusable resources, both numeric and interval-valued; and continuous linear change to quantities. It works in three stages. In the first stage, a representation of the domain and problem in an extended version of PDDL+ is compiled into a system of Boolean combinations of propositional atoms and linear constraints over numeric variables. In the second stage, a SAT-based arithmetic constraint solver, such as LPSAT or MathSAT, is used to find a solution to the system of constraints. In the third stage, a correct plan is extracted from this solution. We discuss the structure of the planner and show how planning with time and metric quantities is compiled into a system of constraints. The proofs of soundness and completeness over a substantial subset of our extended version of PDDL+ are presented.

Keywords: SAT-based planning; LPSAT; Continuous time; Metric quantities; Processes

1. Introduction

Numeric and geometric entities that change continuously in time are central features of many domains, especially physical domains: the position of an object in space, the amount of gasoline in a tank, the temperature of water in a pot, and so on. Early generations of

* Corresponding author.
E-mail addresses: jiae@cs.nyu.edu (J. Shin), davise@cs.nyu.edu (E. Davis).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2005.04.001

domain-independent planners did not deal with numeric quantities at all, and even now few planners deal with continuous change. The TM-LPSAT system described in this paper is the first planner that uses the SAT-based planning methodology to deal with continuous change, as well as many other aspects of numeric quantities.

Over the past decade, dozens of new powerful engines for propositional satisfiability have become available [55] and are now being used in a broad range of applications. One very successful application has been the development of *SAT-based propositional planning*, in which a planning problem is compiled into a set of propositional constraints in such a way that a solution to the constraints demarcates a valid plan [32,34,35]. Recently, a new class of inference engines[1] based on propositional satisfiability solvers has been developed for systems of Boolean combinations of propositional atoms and linear constraints over real-valued quantities [3,5,53].

In this paper, we show how the SAT-based planning framework can be extended, using SAT-based arithmetic constraint solvers, to deal with domains that involve continuous time, resources, and real-valued quantities.

The TM-LPSAT planner constructs plans in domains with the following features:

- The effects and preconditions of actions can involve discrete, real-valued, and interval-valued fluents.
- An action can change the value of a real-valued fluent either continuously, as a linear function of time, or discretely.
- An action may be either atomic or durative (taking place over an extended time interval).
- An action may take real- or interval-valued parameters.
- Actions may be concurrent.
- Exogenous events may occur.
- Autonomous processes can be defined in the language.
- Processes that make a continuous change on the same fluent may be concurrent.
- Reusable resources, both numeric and interval-valued, can be defined in the language.

Fig. 1 shows the architecture of TM-LPSAT. The input to TM-LPSAT consists of a domain description and a problem specification represented in PDDL+ [24,25] (more precisely, in a version of PDDL+ with certain restrictions and extensions as described in Section 3). The *compiler* compiles the planning problem into a set of constraints, each of which is a disjunction of propositional atoms and linear (in)equalities over numeric variables. The set of constraints is passed to the *SAT-based arithmetic constraint solver* which finds a solution if one exists. From the solution, the *decoder* extracts a valid plan. The overall system is thus a powerful and elegant planner for a wide range of problems.

Our main contribution in TM-LPSAT has been the development of the compiler. From our point of view, the constraint solver can be viewed a black box, that takes as input a set of

[1] We will call these "SAT-based Arithmetic Constraint Solver" in this paper. They are also called "SAT-based Decision Procedure" or "Theorem Prover" in the literature.

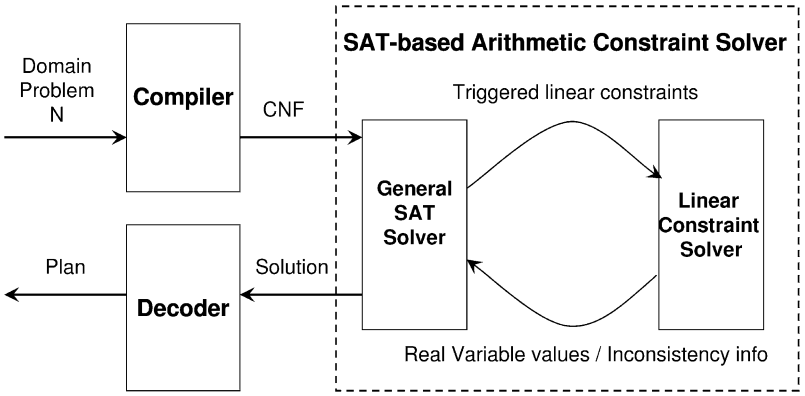

Fig. 1. Architecture of TM-LPSAT.

constraints of the form described above and outputs a solution if one exists and a flag if no solution exists. A number of different architectures for such a constraint solver are possible, at least in principle; it could be complete or heuristic, deterministic or probabilistic. In developing and testing TM-LPSAT, we have used two pre-existing SAT-based constraint solvers, LPSAT [53] and MathSAT [3,9]. In Section 2.2 we discuss their architecture. In Section 8 we will sketch a branch-and-bound architecture that would enable the solver to solve optimization problems.

Two sample problems will illustrate the power of the TM-LPSAT planner, and will help introduce the sample domains that we will discuss in Section 5:

Problem 1.1. An agent must deliver 5 gallons of water to a location LD. Currently the agent is at a location LS 100 feet away, with two four-gallon buckets. At LS there is also a tap that pours water at the rate of 0.1 gallons per second. The agent can move at 5 feet per second.

The following plan will enable the agent to achieve his goal in a total of 70 seconds: He turns on the tap and let it pours into bucket 1 for 10 seconds. Bucket 1 now holds 1 gallon. The agent turns off the tap, puts bucket 2 under the tap, and turns on the tap. Then, he carries bucket 1 to LD, empties bucket 1 at LD, and returns to LS. The round trip takes him 40 seconds, so bucket 2 now holds 4 gallons. He picks up bucket 2, carries it to LD, and empties it.

If the agent can carry two buckets at once, then a simpler solution is possible: He pours 3 gallons into bucket 1, 2 gallons into bucket 2, carries them both to LD, and empties them, again completing the task in 70 seconds.

Problem 1.2. A computer architecture uses variable-length partitions as its memory model; that is, each job occupies a consecutive segment of RAM, which is fixed throughout the lifetime of the job. The machine has 128M of RAM. The operating system needs to schedule jobs with the following characteristics:

Job	Time	Space
A	100	80M
B	50	15M
C	120	20M
D	40	65M
E	100	20M
F	40	40M

Assume that the jobs are I/O bound, so that the time requirement is independent of how many jobs are currently active.

The following plan completes all the jobs by time 160:

Job	Start	End	Segment
A	0	100	0-80
C	0	120	80-100
B	0	50	100-115
E	50	150	108-128
F	100	140	0-40
D	120	160	40-105

This paper is organized as follows. Section 2 reviews previous and related work, including the work on SAT-based planning, SAT-based arithmetic constraint solvers, and PDDL+, which we draw on in the construction of TM-LPSAT. Section 3 discusses the extensions and restrictions we have made to PDDL+. Section 4 discusses the temporal ontology. Section 5 presents a few sample domains and planning problems that TM-LPSAT can handle. Section 6, the core of our research, enumerates the rules for translating a problem in PDDL+ into a system of constraints. (Table 1 on page 210 contains a summary of the constraints.) Section 7 discusses the soundness and completeness of our system. Section 8 presents our conclusions and discusses future work. Appendix A gives a complete listing of the PDDL+ definition of the "Bucket" domain and the problem described in Problem 1.1. Appendix B gives the proof that TM-LPSAT is sound and complete over a substantial subset of our extended version of PDDL+ Level 5.

2. Previous and related work

The TM-LPSAT planner builds on three foundations:

- SAT-based planning: In this planning paradigm, a planning problem represented in a high-level planning language is compiled into a corresponding set of propositional formulas. Solving the planning problem thus corresponds to solving the propositional satisfiability problem (SAT) over these formulas.
- SAT-based arithmetic constraint solvers, constraint satisfaction engines that find solutions to Boolean combinations of propositional atoms and linear (in)equalities.
- The PDDL+ specification language for planning domains and problems.

Also related, though not directly used in TM-LPSAT, are

- Other planning paradigms for dealing with metric time and numeric quantities.
- Other automated reasoning applications that deal with continuous change.

We will discuss in turn each of these categories of previous work and their relation to TM-LPSAT.

2.1. Planning as propositional satisfiability

The architecture of a SAT-based propositional planner is shown in Fig. 2. The idea of SAT-based propositional planning [32,34,35] is to convert a planning problem in a domain with discrete actions and fluents[2] with discrete values into a set of propositional constraints. This is done as follows:

- An upper bound N is guessed[3] for the number of steps needed in the plan. Time points are labeled 0 . . . N.
- The following propositional atoms are defined at each time point I:
  A. For each fluent F, for each possible value V of F, the statement that the value of F at time I is V.
  B. For each action A, the statement that A is executed at time I.

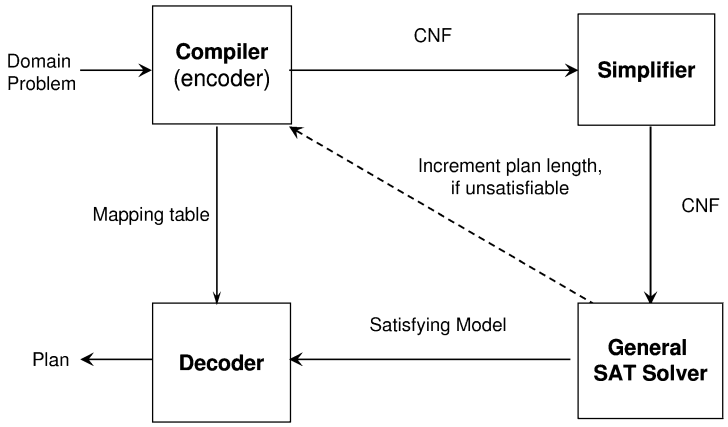

Fig. 2. Architecture of a SAT-based propositional planner.

[2] Throughout this paper, we will use the word "fluent" in the temporal logic sense of "entity that takes on different values at different times" rather than meaning the particular PDDL+ construct of that name. Temporal logic "fluents" include PDDL+ "predicates".

[3] Rather than "guess" here, one can systematically search for the smallest possible value of N in either of two ways: (1) Begin with a random guess on the length of plan. If a plan is found, do binary search over the length of the plan. If no plan is found, then guess a plan length higher than the current one, and iterate. (2) Use a Graphplan-like search [7] to find a lower bound on the length of the shortest possible plan; initialize N to that value; and then iteratively increase N until a solution is found.

- The laws governing the domain are imposed by asserting every instance of every law at every moment of time. In classical planning domains the major categories of laws are: causal laws, domain constraints, and frame axioms.
  The paradigm, indeed, will support essentially *any* computable constraint; e.g., that the number of times action A is executed must be a prime number; a fluent will change five time units after some particular action, etc. The main limiting factor on incorporating such constraints is finding systematic ways to express them in a general domain definition language such as PDDL+.
- The problem instance is asserted by stating that the starting conditions hold at time 0 and that the goal conditions hold at time N.
- The constraints are then fed to a propositional satisfiability solver. If a solution of the constraints can be found, then the set of actions that are marked as occurring in the solution constitutes a valid plan.

SAT-based propositional planners can be implemented easily and, with the current generation of satisfiability solvers [55], quite effectively. The planners also have no additional difficulty in dealing with ADL features such as conditional effects or quantifications. The major drawback of SAT-based planning is that large domains can lead to enormously large systems of constraints. Particularly dangerous are functions with many arguments; a fluent function or action function with k arguments generates a collection of atoms of size exponential in k.

Since the introduction of SATPLAN [32,34], a number of other SAT-based planners have been developed, including BLACKBOX [35] and MEDIC [21]. Building a SAT-based planner involves two main types of choices. The first is the representational issue of choosing an encoding: What propositional atoms should be used, and how domain constraints should be encoded as axioms. The effectiveness of different encoding schemes has been studied extensively [21,33]. The second choice is the technique used to solve the satisfiability problem; both probabilistic methods like GSAT [32] and deterministic methods like extensions [55] of DPLL algorithm [14] have been studied.

Temporal planning over integer time, involving constraints such as, "Action A requires 3 units of time to complete", can easily be handed in this framework, as long as the integers involved are small. One defines a time point at each integer, and then encodes such constraints in the formulas "If A starts at T_0 then it ends at T_3", "If A starts at T_1 then it ends at T_4", and so on [40].

The LPSAT planner [53,54] developed the LPSAT engine to extend the approach further to solve problems in metric planning; that is, planning with real-valued quantities, such as the quantity of gasoline in a tank. However, the LPSAT planner could not handle problems involving durative actions or continuous change.

Indeed, the claims were made that the SAT-based planning paradigm could not be extended to deal with continuous time, because there would be an infinite number of ground actions, corresponding to the infinite set of choices as to when to execute an action and how long to continue it [37,49]. The construction of TM-LPSAT has disproved the claims. The way this issue is resolved in TM-LPSAT is to encode a history in terms of a finite set of *interesting* time points at which something changes, rather than trying to encode all time

points on the time line. The clock time of an interesting time point is a numeric variable that is assigned a value by the constraint solver.

Many of the rules in TM-LPSAT for generating constraints come directly out of these previous systems. The rules that deal with effects and preconditions connecting atomic actions with discrete fluents are the same as in the SAT-based propositional planners. The rules that deal with discrete (discontinuous) effects of actions on a numerical fluent, and with numerical preconditions of actions, are the same as in the LPSAT planner.

2.2. SAT-based arithmetic constraint solver

As shown in Fig. 1, a SAT-based arithmetic constraint solver consists primarily of two coupled modules [1]: A DPLL-based systematic SAT solver [55], such as RelSAT [6] and MiniSAT [20], and an incremental linear programming (LP) solver, such as Cassowary [8].

A DPLL-based SAT solver does a depth-first search with backtracking through the space of partial truth assignments. At its deduction phase, unit resolution and propagation are applied. Modern, high-powered SAT solvers enhance the basic backtracking search using such techniques as conflict-driven learning, random restarts, non-chronological backtracking, and branching heuristics.

The two modules are combined as follows: The input to the constraint solver is a set of generalized clauses. Each clause is a disjunction; each disjunct is either a propositional literal or a linear equality or inequality over numeric variables. The SAT solver first looks for a propositional (partial) solution, treating each linear equation as a propositional atom (called a *trigger*), then the LP solver tries to solve the set of inequalities that have been marked as TRUE in the (partial) solution. If that set is inconsistent, then the SAT solver utilizes information on inconsistency detected through back-jumping or learning (adding a clause stating that these linear inequalities are not all TRUE), and it looks for a new propositional solution. It continues going back and forth between propositional and numeric mode until either finding a solution, establishing that no solution exists, or reaching the given time limit.

Since the introduction of the LPSAT architecture by Wolfman and Weld [53], MathSAT [3,9] and more general theorem provers such as CVC Lite [5] have been developed in verification community. These solvers vary in the SAT solving techniques that they incorporate; in their search heuristics; and in the special cases of "easy" LP categories that they identify.

2.3. PDDL+

PDDL (Planning Domain Definition Language) is a declarative language for the definition of causal domains and planning problems. The basis of our work is PDDL+, which was the most recent extension[4] to PDDL when we began work on TM-LPSAT. PDDL+ comprises five levels. Level 1 contains discrete actions and fluents. Level 2 adds features

[4] Since then, PDDL2.2 [19], extended for IPC4, was released. The features in PDDL+ remain intact; additional features included in PDDL2.2 are *derived predicates* and *timed initial literals* (sort of deterministic events). These features could be easily incorporated in TM-LPSAT.

for numeric quantities. Level 3 allows durative actions that cause discrete changes occurring at the beginning and at the end of the action. Level 4 allows durative actions that cause continuous changes throughout the occurrence of the action. (Levels 1 through 4 collectively comprise PDDL2.1 [25].) Level 5 [24], proposed but not approved by the IPC (International Planning Committee), is a deterministic real-time temporal model of exogenous events and autonomous processes. (McDermott [42] proposes an alternative for incorporating processes and events.)

Thus in PDDL+ continuous time can be modeled in three ways: durative actions with discrete effects, durative actions with discrete/continuous effects along with atomic actions, or processes and events along with atomic actions. These, however, cannot be mixed together.

2.4. Planners dealing with time and metric quantities

The state-of-the-art domain independent planners that competed at IPC3 [39] and IPC4,[5] which are mostly heuristic-based, display impressive performance in handling numeric and/or temporal domains. However, they are quite limited in the range of temporal and metric constraints they can deal with; typically, they require that actions have a fixed, constant duration and use a fixed quantity of resources. By contrast, dealing with more expressive temporal metric constraints, such as unknown durations and uncertain usages of resource, imposes no additional difficulties on the compilation phase of TM-LPSAT, though presumably the solving phase takes longer to solve the numeric constraints. None of those planners can deal with a real-time temporal model involving autonomous processes, although LPG [27] and SAPA [17] claim to handle durative actions with continuous change.

The Sekitei program [36] is a metric planner that uses a modified Graphplan search with numeric resources to solve the problem of placing software components on a network. In principle, the planning technique accommodates non-linear constraints; the current implementation, however, deals only with linear constraints. It does not deal with continuous change.

The plan graph generated in Graphplan [7] is a representation of essentially the same plan space as that used in SAT-based planners. It is therefore possible to use plan graphs as the basis for the compilation phase of a SAT-based planner; this is done in BLACKBOX [35]. Graphplan-based temporal planners include TGP [50], MILP [16] and LPGP [38]. LPGP models and handles temporal constraints over durative actions in a way similar to TM-LPSAT: rather than projecting time on the plan graph as done in TGP, temporal constraints imposed among actions in a (partial) plan are checked for consistency by a LP solver while extracting a plan. The difference is that in LPGP a plan is searched backward, while in TM-LPSAT search for a satisfying solution is non-directional. Consequently TM-LPSAT does not suffer from the difficulty caused by backward search, such as dealing with a durative action whose ending action is not included in the plan, but whose starting action needs to be included in the plan. MILP builds a plan graph for logical constraints

[5] http://ls5-www.cs.uni-dortmund.de/~edelkamp/ipc4.

in the same way as in LPGP, and converts the graph, together with temporal constraints among actions, into an integer linear programming problem.

A number of partial-order planners have dealt, to greater or lesser extent, with problems involving continuous change, including Processes [29], DEVISER [51], SPIE [52], GORDIUS [48], FORBIN [15], Excalibur [18], and ZENO [44,45]. Most of these addressed continuous change only in a substantially more restricted setting than TM-LPSAT. ZENO, by contrast, permitted a very general plan specification language, though its models of concurrency and of processes were less general than TM-LPSAT—ZENO could not handle concurrent continuous change of quantities. Like TM-LPSAT, ZENO was restricted to the piecewise linear function and called a LP solver within plan refinement loop. It was extremely slow; Wolfman and Weld [54] report that ZENO was unable to solve even the simplest of the logistic problems that were used to test LPSAT.

McDermott [42,43] has extended his estimated-regression planner to deal with processes and continuous change. Unlike TM-LPSAT, his planner is not complete (arguably an advantage, of course). It finds zero crossings using binary search, so presumably it could easily be extended to non-linear functions; however, the current implementation has the same restriction as TM-LPSAT to linear functions with constant coefficients.

2.5. Formalisms for modeling continuous change

The best known study of processes in the AI literature is QP theory [22], which initiated a large body of research on physical reasoning with processes. This is at the extreme opposite end in terms of the language of quantities used; effects of processes are characterized purely in qualitative terms. A number of important ideas developed in this line of research have yet to be incorporated into the planning literature, such as indirect influences. Davis [13] gives a logical analysis of QP theory.

Another formalism closely related to our work is a theory of hybrid system [30]. A hybrid automaton combines a finite state machine undergoing a series of discrete change with real-valued variables undergoing continuous change. A hybrid system is a collection of interacting hybrid automata. Fox and Long [24] have defined a semantics for PDDL+ in terms of hybrid systems. The planning problem corresponds to the "reachability" problem in a theory of hybrid system. A bounded reachability problem of a linear hybrid system was formulated as a satisfiability problem in [4]. Their encoding is based on state transitions with absolute time and clocks; on the other hand, our encoding was based on the constraints imposed by the operators happening at the time points.

3. Extensions and restrictions to PDDL+

The input specification language for TM-LPSAT extends PDDL+ in four ways:

The first extension is that actions in TM-LPSAT may have numeric parameters. For instance, there can be actions "pour(N, BS, BD)" of pouring N gallons from bucket BS to bucket BD; "set-oven(T)" of setting the thermostat in an oven to temperature T; or "play_key(K, V)" of playing piano key K at volume V. PDDL2.1 [25] excludes this feature that existed in the original version [41], but their arguments do not strike us as cogent.

Numeric parameters obviously greatly increase the expressive power of the language, and, in the TM-LPSAT approach, impose no additional computational burden.

One restriction, however, does have to be imposed on actions with numeric and interval parameters: There cannot be two or more concurrent actions[6] with the identical non-numeric parameters. For instance, the actions "pour(5, b1, b2)" and "pour(2, b1, b2)" cannot be executed concurrently, though "pour(5, b1, b2)" and "pour(2, b3, b4)" can be concurrent. The restriction is *necessary* because the entire SAT-based methodology rests on the assumption that, once you have guessed the number of significant time points, the number of possible entities, propositions, and numeric parameters can be bounded; if an unbounded collection of simultaneous actions of the form "pour(N, c1, b1)" can be generated, that would be a problem. The restriction is *reasonable* because such numeric parameters are typically used in one of two ways. If the value of the parameter is assigned to a fluent—e.g., "set-dial(N, D)" results in dial D being set to value N—then two actions with different numeric parameters would be mutually exclusive. If the value of the parameter is used to increment a fluent—e.g., "pour(N, BS, BD)" increases the quantity of liquid in BD by N and increases the quantity in BS by N—concurrent actions pour(5, b1, b2) and pour(2, b1, b2) can be combined into a single action pour(7, b1, b2). There are a few exceptions; for instance, the action "sound(F, V)", sounding a tone with frequency F and volume V, executed by a robot with electronic speakers. It is possible for such a robot to execute "sound(F1, V1)", "sound(F2, V2)" ... concurrently with different frequencies. Our representation cannot handle this case.

By virtue of this restriction, an action type is identified by the name of the functor and the non-numeric parameters. For example, we may speak of the action type "pour($\cdot$, b1, b2)" (pouring some amount from b1 into b2) and be sure that at most one of these occurs at one time.

The second extension of PDDL+ is that our specification language supports *reusable* metric resources, including numeric and interval-valued; that is, resources that are held by an action while the action lasts and released when the action is complete. We denote it by a "*use*" statement of the form "(use ?resource ?amount)".

The motivation behind this extension is as follows: PDDL+ has no explicit provision for resources. It treats numeric resources like any other numeric quantities. Thus concurrent (shared) uses of reusable resource among atomic actions cannot be modeled in PDDL+. For example, suppose that an agent has K identical effectors, and that there is a collection of atomic actions, such as flipping a switch, each of which requires the use of one effector. Then, clearly, it should be possible for the agent to execute K such actions concurrently. However, this can only be represented in PDDL+ by representing each effector separately. The result would be that each different assignment of actions to individual effectors would be considered separately, thus multiplying the branching factor by K factorial. The use of interval resources among actions, atomic or durative, cannot be expressed in PDDL+, because there are infinitely many choices for the lower and upper bounds of the interval to be allocated to the action.

[6] The axioms in [46] allow two such durative actions to continue concurrently, though not to start simultaneously. This requires a more complex representation, which identifies durative actions by their starting time.

RAM memory in Problem 1.2 can be represented as a reusable interval resource, and its concurrent uses are disjoint subintervals of the resource. Other kinds of domains where interval resources are useful include the placing of books on a shelf; the assignment of frequency ranges to broadcasters; and so on.

The third extension of PDDL+ is that our language supports *interval-valued* fluents and quantities. We have incorporated in our language and includes Allen's 13 binary interval relations [2] and several other basic useful functions on intervals.

The fourth extension is that we distinguish between numeric functions whose values are constant over time and given in the problem statement and numeric functions whose values vary over time. The former are marked as being of type *float*; the latter are of type *fluent*. For example, in the Bucket domain, "(capacity ?b - bucket)" is of sort *float* whereas "(level ?b – bucket)" is of sort *fluent*. This distinction was made in the original PDDL [41], but was removed in PDDL2.1. This feature is particularly important in TM-LPSAT for two reasons. First, when an entity changes its value over time, it is necessary to create a separate variable for the value of the entity at each time point. Thus, if there are N time points, then each fluent F generates N numeric variables, whereas a float F generates no numeric variables. Second, if X and Y are variables then the equation $X = AY$ is a *linear* equation if the value of A is known at compilation time, but it is a *non-linear* equation if the value of A is not known. Since TM-LPSAT can only deal with linear equations, quantities like the flow-rate of a tap must be floats, so that equations like "change-in-quantity = flow-rate $*$ duration-of-flow" are linear equation in the variables "change-in-quantity" and "duration-of-flow".

A few features of PDDL+ cannot be handled in the current version of TM-LPSAT. First, TM-LPSAT cannot optimize a specified plan metric, a limitation inherited from the architecture of the arithmetic constraint solvers we use. Second, the language must be restricted so that, in any multiplication, all but one of the terms can be statically evaluated; and, in any division, the denominator can be statically evaluated. Otherwise, the result will be a non-linear equation, which existing SAT-based arithmetic constraint solvers cannot deal with, and which will certainly be much more difficult for any possible constraint solver. All other features of PDDL+ are included.

4. Temporal ontology

We use a linear, real-valued time line. The representation used in the constraint language output by TM-LPSAT characterizes the time line in terms of the states of the world at a collection of *significant time points*. A significant time point is one where "something changes"; roughly speaking, some action, event, or process occurs, starts, or ends. In the intervals between significant time points, fluents are either constant, or, if they are numeric, they may undergo continuous change as a linear function of time. Every discontinuous change, or change in the derivative of a numeric fluent, occurs at a significant time point. Thus, there are two states associated with each time point T. The "state before T" consists of the values of fluents and activity levels immediately before the changes that take place at T; the "state after T" consists of their values *after* the changes that take place at T.

Each time point has a *clock time*, which is a non-negative real value. These clock times become numeric variables in the system of constraints set up by the compiler.

One tricky point arises in any theory that includes both real-valued time and atomic events and actions: How should one deal with atomic events/actions that, intuitively, should occur one immediately after another? Suppose for instance, that action A has precondition P and effect Q and that event E has triggering condition Q and effects (not P) and (not Q). It should be possible for A to be executed, and then E will be triggered. The problem is, when do these occur? If there is a gap between A and E, then why doesn't E occur sooner? If A and E occur at the same time, then how can you be sure that A is occurring before E (which is possible) and not the other way around (which is impossible)?

The semantics defined by Fox and Long [24] for PDDL+ Level 5 involves an unusual model of the time line.[7] An event that is triggered by an action or another event occurs "immediately after", with no time gap between. To deal with this, there need to be two distinct time points with equal clock times. Thus, we represent the situation by saying that A occurs at time point T_5, say, and that E occurs at time point T_6 and that these are different *time points* even though $\text{clock}(T_5) = \text{clock}(T_6) = 17.28$ sec. We impose an order on these two time points but there is no time gap between them.

Fox and Long's semantics both for Level 4 and for Level 5 requires that a time point when an *action* occurs be separated from the previous time point by a fixed positive constant ε, corresponding to the reaction time of the agent or the precision of the agent's clock. This dependence of the theory on the arbitrary quantity ε is ugly, and in our implementation of TM-LPSAT we have eliminated it in Level 5. If we can idealize events as occurring in immediate succession, why not actions as well? We have maintained the ε gap in our implementation of Level 4, where it applies to all time points (there are no atomic events in Level 4).[8]

Note that an event E *must* disable its own triggering condition; else there would have to be additional occurrences of E at T_{i+2}, at T_{i+3}, etc.; the result would be that the system of constraints would have no solution with finitely many time points.

An *atomic action* occurs instantaneously. An action is characterized by preconditions that must hold before the action and effects that hold after the action. For example, the action "turn on the faucet" has the precondition that the faucet is off and has the effect that the faucet is on.

In PDDL+, a *durative action* is conceptualized as consisting of three epochs: initialization, continuation, and termination. The initialization and termination resemble atomic actions; they are instantaneous and are characterized by preconditions and discrete effects.

[7] Actually, this paper by Fox and Long is not at all clear on this point. Our interpretation here is our best guess as to what is intended. If this isn't right, then it is easily changed; one of the advantages of the SAT-based planner is that making that kind of change generally affects a only few specific axioms.

[8] One problematic situation is if the invariant conditions of a durative action become FALSE at a time that is less than ε after the previous time point. TM-LPSAT considers such a case to be impossible; the epsilon gap axiom (6.1) in Section 6 requires that any two significant time points be separated by at least ε, whereas the zero-crossing axiom (10.11) requires that there be a significant time point exactly when the invariant condition ceases to hold. Hence, any plan that gives rise to such a situation is considered invalid.

The continuation may take any length of time greater than ε. Its invariants must be satisfied as long as it continues. Its effects may either take effect continuously for its entire duration, like the effects of a process, or discretely at its end.

A durative action is feasible only if it can be carried through to termination; it cannot be begun and then abandoned.

For example, one can define "filling bucket B from tap T" as a durative action with the following properties. The initialization has the preconditions that tap T is off; that the bucket and the agent are at the same location as the tap; and that the bucket is not full; and it has the effect that tap T is on. The continuation has the precondition that the tap is turned on, and that the tap and the bucket are at the same location. It has the continuous effect that the level in the bucket rises at flow-rate(T). The termination has the precondition that the tap is on it has the effect that the tap is off.

An *event* is like an atomic action, except that, whereas an atomic action *may* occur if its preconditions hold (if the actor so chooses), an atomic event *must* occur if its precondition hold. For example, suppose that some of the buckets are fragile, with a weight limit that is less than their volumetric capacity. If the quantity of liquid inside exceeds the weight limit, the bottom falls out. This can be characterized in terms of an atomic event "break-Bucket(B)". The preconditions are that B is unbroken and that level(B) $\geqslant$ weightLimit(B). The effects are that B is broken and that level(B) $= 0$.

A *process* is active over an extended interval. It is characterized by preconditions and effects. The preconditions must hold through the interval; if the preconditions cease to hold, the process stops. The effects of a process are, in the language of Forbus [22], *direct influences* on numeric fluents. Specifically, each process has a fixed influence on some collection of real-valued fluents; the derivative of the fluent at a given time is the sum of its influences over all active processes and actions that influence it.

For example, the process "fillingBucket(B – bucket T – tap L – location)" has the precondition that tap T is currently pouring into B and that the bucket is not yet full. (Of course, the tap will continue to pour even when the bucket is full, but it will cease to fill the bucket.) The process has the effect of increasing level(B) at the rate flow(T). (We allow only taps that are fully on or off.) There can be several co-located taps pouring simultaneously into the same bucket; if so, the rate of increase of the level in the bucket is the sum of the flow-rates of the individual taps.

PDDL+ permits concurrent actions under fairly restrictive conditions, designed to ensure (a) that the result of concurrent actions is meaningful; (b) that the actions do not interact, either destructively or synergistically. However, two actions whose effect is to increase or decrease a given numeric fluent can be executed concurrently, since the net effect is well-defined as the sum of the separate effects. For example, one can pour into bucket b1 both from bucket b2 and from bucket b3 simultaneously. Essentially, these conditions amount to requiring that the actions be commutative; that is, that they can be executed in any order and that the result of executing them is the same in all orderings. The actual condition imposed is sufficient, though not necessary, to ensure commutativity; this is in order that the conditions for concurrency can be computed easily and statically.

5. Sample domains

Let us illustrate some of the PDDL+ constructs that TM-LPSAT can deal with:

Example 5.1. The atomic action of pouring quantity Q of water from one bucket to another can be encoded as follows:

```
(:action pour
 :parameters   (?a - agent ?bs - bucket ?bd - bucket ?q - real ?l - location)
 :precondition (and (at ?a ?l) (carrying ?a ?bs) (at ?bd ?l)
                    (> ?q 0)
                    (<= ?q (level ?bs))
                    (<= ?q (- (capacity ?bd) (level ?bd))))
 :effect       (and (increase (level ?bd) ?q)
                    (decrease (level ?bs) ?q))
)
```

Note the real-valued parameter ?q; the planner can choose to pour any positive amount ?q as long as ?q is not more than the amount of water in the source, and not more than the amount of room in the destination.

Example 5.2 (*PDDL+ Level* 3). The action of filling a bucket can be characterized as a durative action with a discrete effect as follows:

```
(:durative-action fillBucket1
 :parameters (?a - agent ?b - bucket ?t - tap ?l - location)
 :duration   (at end (<= ?duration (/ (- (capacity ?b) (level ?b)) (flow-rate ?t))))
 :condition  (and  (at start (not (on ?t)))
                   (at start (at ?a ?l)) (at start (at ?b ?l)) (at start (at ?t ?l))
                   (over all (on ?t)) (over all (at ?b ?l))
                   (at end (on ?t)))
 :effect     (and  (at start (on ?t))
                   (at end (not (on ?t)))
                   (at end (increase (level ?b) (* ?duration (flow-rate ?t)))))
)
```

The value of the duration will be set by the planner; this determines the amount of water to fill the bucket with. It is critical, here and in Examples 5.3 and 5.4, that the quantity "(flow-rate ?t)" can be evaluated *statically*. If this quantity is a variable, then the equation becomes non-linear, and the existing SAT-based arithmetic constraint solvers cannot deal with it.

The PDDL+ semantics allow a fluent whose value changes as an effect of a durative action to be referable and updatable by other actions during the course of the action. Thus it is possible for one bucket to be filled by two different taps concurrently. For instance, "fillBucket1(a1,b1,t1,sl1)" and "fillBucket1(a2,b1,t3,sl1)" can be concurrent. However, due to the mutex rule called *no moving target* on "(level ?b)", the two actions cannot finish at the same time.

In this model, when a durative action makes a change to a numeric fluent, the change occurs instantaneously at the end points of the action. However, in most cases, the actual change to a fluent occurs gradually during the course of the action. Therefore, in the middle of the occurrence of the durative action, the value given by this model is not correct. The model of durative actions given in Example 5.3 overcomes this limitations.

Example 5.3 (*PDDL+ Level* 4). The action of filling a bucket can be characterized as a durative action causing continuous change as follows:

```
(:durative-action fillBucket2
 :parameters (?a - agent ?b - bucket ?t - tap ?l - location)
 :duration   ()
 :condition  (and (at start (not (on ?t)))
                  (at start (at ?a ?l)) (at start (at ?b ?l)) (at start (at ?t ?l))
                  (over all (at ?b ?l)) (over all (on ?t))
                  (over all (<= (level ?b) (capacity ?b)))
                  (at end (on ?t)))
 :effect     (and (at start (on ?t))
                  (at end (not (on ?t)))
                  (increase (level ?b) (* #t (flow-rate ?t)))))
```

In the last line above, "#t" is a special variable which, at each instant during the execution of the durative action, denotes the length of time that has elapsed since the action started.

Unlike the model in Example 5.2, this representation allows other actions to access the correct value of a continuously changing fluent at any time point over the period of the action.

Example 5.4 (*PDDL+ Level* 5). The action of filling a bucket can be characterized yet again as an atomic action of turning on the tap, followed by a process of flow from the tap into the bucket, followed by an atomic action of turning off the tap.

```
(:action turnOnTap
 :parameters      (?a - agent ?t - tap ?b - bucket ?l - location)
 :precondition    (and (at ?a ?l) (at ?b ?l) (at ?t ?l) (not (on ?t)))
 :effect          (and (on ?t) (filling ?t ?b))
)
(:process fillingBucket
 :parameters      (?b - bucket ?t - tap ?l - location)
 :precondition    (and (filling ?t ?b)
                       (<= (level ?b) (capacity ?b))
                       (at ?b ?l))
 :effect          (increase (level ?b) (* #t (flow-rate ?t)))
)
(:action turnOffTap
 :parameters      (?a - agent ?t - tap ?b - bucket ?l - location)
 :precondition    (and (at ?a ?l) (at ?t ?l) (on ?t) (filling ?t ?b))
```

```
 :effect          (and (not (on ?t)) (not (filling ?t ?b)))
)
```

Example 5.5 (*Reusable metric resources*). We can model the domain of filling a bucket in a different way: Assume that the taps are classified as of *small* capacity or of *big* capacity. A number of taps, either of the same capacity or not, may be at each location. Let "(flow-rate ?tot)" be the flow-rate of a tap of type ?tot; let "(no-of-taps ?tot ?l)" be the number of taps of type ?tot in location ?l. Then "fillBucket2" shown in Example 5.3 can be represented as follows:

```
(:durative-action Modified-fillBucket2
 :parameters (?a - agent ?b - bucket ?tot - TypeOfTap ?l - location)
 :duration   ()
 :condition  (and (at start (at ?a ?l)) (at start (at ?b ?l))
                  (over all (at ?b ?l))
                  (over all (<= (level ?b) (capacity ?b))))
 :effect     (and (increase (level ?b) (* #t (flow-rate ?tot)))
                  (use (no-of-taps ?tot ?l) 1))
)
```

If there are a large number of taps of each type at a given location, then using this representation very much reduces symmetry in the search space over the previous representation in which taps are represented individually: In an individualistic representation, the search space may include every possible set of taps; here, by representing the collection of a type of taps as a *multiple-capacity resource*, each such set is summarized by two numeric fluents.

Also representing a resource as a numeric fluent suggests a way to deal with dynamically creating and destroying objects.

Example 5.6 (*Partitioned interval resource*). As described in Problem 1.2, in an operating system that uses variable-length partitions as a memory model, each job occupies a consecutive segment of RAM which is fixed until it finishes. "(RAM-space)" can be defined as a resource of type *interval* in our extended PDDL+. The consecutive segments allocated to jobs running concurrently are disjoint subintervals of the RAM space.

```
(:durative-action executeJob
 :parameters (?j - job)
 :duration   (= ?duration (time-for ?j))
 :condition  (and (at start (not (active ?j)))
                  (over all (active ?j))
                  (at end (active ?j)))
 :effect     (and (at start (active ?j))
                  (at end (not (active ?j)))
                  (use (RAM-space) (memory-for ?j)))
)
```

6. Compilation to constraints

In this section, we describe how domain definition and problem specification given in PDDL+ is translated into a collection of constraints, where each constraint is the Boolean combination of propositional atoms and linear (in)equalities over numeric variables.

The constraints[9] presented in this section are summarized in Table 1. The examples to be seen in this section are from the "Bucket" domain defined in Appendix A, unless otherwise specified.

We define the following propositional atoms and numeric variables.

Definition (*Propositional atoms*).

- For each time T_i, for each Boolean fluent F, the assertion that F holds at T_i. We notate this "$F[T_i]$".

Table 1
Summary of constraints

Category	Constraints	Section	Page
Atomic action	Effects	6.1.1	212
	Preconditions	6.1.2	214
	Mutual exclusion	6.1.3	214
Event	Effects	6.2.1	214
	Preconditions	6.2.2	214
	Immediate triggering by discrete change	6.2.3	215
	Mutual exclusion	6.2.4	215
Process	Effects	6.3.1	215
	Preconditions	6.3.2	216
Zero crossings of events and processes	Triggering/terminating by continuous change	6.10.1	220
Durative action	Precondition and effects	6.4.1	216
	Constraint on duration	6.4.2	217
	Coherence	6.4.3	217
	Invariant conditions	6.4.4	218
	Continuous effects	6.4.5	218
Frame axiom	Propositional or interval fluents	6.5.1	218
	Numeric fluents	6.5.2	218
Time points	Gap between time points	6.6	218
Reusable metric resources	Allocation and deallocation	6.7.1	219
	Propagation	6.7.2	219
	Constraint on capacity	6.7.3	219
Reusable interval resources	Segment allocation	6.9.1	219
	Frame axiom	6.9.2	220
	Non-overlap	6.9.3	220
Intervals	Interval fluents	6.8	219

[9] The corresponding axioms are numbered based on the subsection number, as a prefix, where these constraints are dealt with.

- For each time T_i, for each non-Boolean discrete fluent F, for each value V, the assertion that F has value V at T_i. We notate this "$F[T_i] = V$".
- For each time T_i, for each atomic action/event E, the assertion that E occurs at T_i. We denote it "active(E)[T_i]".
- For each time T_i, for each process P, the assertion "active(P)[T_i]" assert that P is active over the open interval (T_i, T_{i+1}).
- For each time T_i, for each durative action A, the assertions that A starts at T_i; that A is continuing at T_i; and that A ends at T_i. We denote these "starts(A)[T_i]", "continues(A)[T_i]" and "ends(A)[T_i]", respectively.

Definition (*Numeric variables*).

- The clock time of every significant time point T_i, denoted "c(T_i)".
- For every time T_i, for every numeric fluent Q, the value of Q before and after T_i. We notate these "$Q[T_i^-]$" and "$Q[T_i^+]$", respectively. These are not equal if some atomic action or event discretely changes the value of Q at time T_i. (Note that, in domains where all change is discrete, $Q[T_i^-]$ is always equal to $Q[T_{i-1}^+]$ whereas in theories where all change is continuous, $Q[T_i^+]$ is always equal to $Q[T_i^-]$. The need for two values at a time point therefore only arises in theories that combine discrete and continuous change, as in PDDL+ Levels 4 and 5.)
- For every time T_i, for every interval fluent Z, the lower and upper bound of Z at T_i, denoted "left(Z, T_i)" and "right(Z, T_i)". Note that we do not have continuously changing intervals.
- For each numeric fluent Q, for each action or event A that changes Q incrementally (i.e., executes a discrete "increase" or "decrease"), the amount of increase or decrease that an occurrence of A makes to Q at time T_i. This is denoted "$\Delta(A, Q, T_i)$". This enables us to add these up over concurrent actions/events.
- For each numeric fluent Q, for each durative action or process A that changes Q continuously, for each time T_i, the net change in Q due to A between T_i and T_{i+1}. This is denoted "$\Gamma(A, Q, T_i, T_{i+1})$".
- For any durative action A, "Duration(A, T_i)" is a numeric variable for the duration of the instance of A that starts in T_i.
- Let $A(P_1 \ldots P_k, Q_1 \ldots Q_m, Z_1 \ldots Z_p)$ be an action where $P_1 \ldots P_k$ are discrete parameters; $Q_1 \ldots Q_m$ are numeric parameters; and $Z_1 \ldots Z_p$ are interval parameters. Then, by the restriction mentioned in Section 3.2.3, at any particular time T_i, for any particular values $V_1 \ldots V_k$ of the discrete parameters, there is at most one valuation on the Q_i and the Z_i for which an action of the form $A(P_1 \ldots P_k, Q_1 \ldots Q_m, Z_1 \ldots Z_p)$ begins at time T_i. The value of each such Q_j and the values of the lower and upper bound of Z_j are numeric variables; it may appear in a term on the right hand side of an assignment statement or in a condition.
  For example: "pour(?a,?bs,?bd,?q,?l)" is an action with the real-valued parameter ?q. There is therefore a numeric variable "pour$_{?q}$(a1,b3,b4,l3)[T_5]" meaning the amount that a1 should pour from b3 to b4 at l3 at time T_5.
- For each resource R, durative action A, and time T_i, the amount of R that A uses at time T_i. This is denoted "U(R, A, T_i)".

- For any durative action A that uses interval resource R, for any time T_i, numeric variables representing the lower and upper bounds of the segment of R allocated to A at T_i. We denote these "lower(A, R, T_i)" and "upper(A, R, T_i)".

Notational convention. We will use the following convention for labeling time-dependent terms:

- If a complex term α over fluents is evaluated using the values before a discrete change is made at time T_i, we will denote this evaluation as $\alpha[T_i^-]$. That is, it is evaluated with the values of propositional fluents at T_{i-1} and the values of numeric fluents *before* T_i, $Q[T_i^-]$.
- If a complex term α over fluents is evaluated after a discrete change is made at time T_i, we will denote this evaluation as $\alpha[T_i^+]$. That is, it is evaluated with the values of propositional fluents at T_i, and the values of numeric fluents *after* T_i, $Q[T_i^+]$.

We begin by guessing at an upper bound N on the number of significant time points that will be needed to solve the problem. The significant time points are then $T_0 \ldots T_N$.

As discussed in Section 4, we assume throughout that there cannot be two actions, events or processes executing concurrently whose name is the same except for numerical parameters.[10] For example, the actions "pour(a1,b2,b3,2,l1)" and "pour(a1,b2,b3,5,l1)" cannot be executed concurrently; there cannot be two concurrent processes of the form "fillingBucket(b1,t2,l3)" and so on.

6.1. Atomic actions

6.1.1. Effects

A: If an effect of action A is to assign term α to discrete or interval fluent F, then add the constraint:

$$(1.1) \quad \mathit{active}(A)[T_i] \Rightarrow [F[T_i] = \alpha[T_i^-]].$$

For example, one constraint generated by the action "turnOnTap" is

$$\texttt{active(turnOnTap(a1,t1,b2,l3))}[T_5] \Rightarrow \texttt{on(t1)}[T_5].$$

(Here the term α is just the implicit Boolean value TRUE.)

B: If an effect of action A is to assign term α to numeric fluent F, then add the constraint:

$$(1.2) \quad \mathit{active}(A)[T_i] \Rightarrow [F[T_i^+] = \alpha[T_i^-]].$$

[10] This is slightly at variance with the PDDL+ semantics, which does allow this for durative actions. For example, it is possible that in the bucket domain given in Example 5.5, "Modified-fillBucket2(a1,b1,ST,sl1)" starting at T_2 and "Modified-fillBucket2(a1,b1,ST,sl1)" starting at T_4 can continue concurrently until T_6, as long as the bucket b1 is not full until T_6.

For example, walking between two locations can be represented as a "walking" process triggered by "go" action and "arrive" event. The "go(?a,?sl,?dl)" action sets the distance for the agent to walk as follows:

```
(assign (distance-to-walk ?a ?dl) (distance ?dl ?sl)).
```

The constraint associated with this would be:

$$\texttt{active(go(a1,sl1,dl1))}[T_5] \Rightarrow$$
$$\texttt{distance-to-walk(a1,dl1)}[T_5^+] = \texttt{distance(dl1,sl1)}.$$

C: If an effect of action A is to increase numeric fluent Q by the term α, then add the constraints:

(1.3) $\mathit{active}(A)[T_i] \Rightarrow [\Delta[A, Q, T_i] = \alpha[T_i^-]]$.

(1.4) $\neg\mathit{active}(A)[T_i] \Rightarrow [\Delta[A, Q, T_i] = 0]$.

For example, two constraints associated with the "pour" action are:

$$\texttt{active(pour(a2,b2,b3,}\cdot\texttt{,l1)},T_5) \Rightarrow$$
$$\Delta\texttt{(pour(a2,b2,b3,}\cdot\texttt{,l1),level(b3)},T_5) = \texttt{pour}_{?q}\texttt{(a2,b2,b3,l1)}[T_5].$$
$$\neg\texttt{active(pour(a2,b2,b3,}\cdot\texttt{,l1)},T_5) \Rightarrow$$
$$\Delta\texttt{(pour(a2,b2,b3,}\cdot\texttt{,l1), level(b3)},T_5) = 0.$$

The first constraint above is read, "If agent a2 pours water from bucket b2 to bucket b3 at location l1 at time T_5, then the increase in the level of water in b3 due to this action is equal to the amount that has been poured".

D: Let $A_1 \ldots A_k$ be all the action/events that can change numeric fluent Q incrementally. Let $E_1 \ldots E_p$ be all the action/events that can assign to Q. Add the constraint:

$$\neg\mathit{active}(E_1)[T_i] \wedge \cdots \wedge \neg\mathit{active}(E_p)[T_i] \Rightarrow \left[Q[T_i^+] = Q[T_i^-] + \sum_j \Delta(A_j, Q, T_i) \right]. \quad (1.5)$$

For example, suppose that there are three buckets, b1, b2, b3, one agent a1 and two locations l1 and l2. Then the level in b1 can be changed either by pouring out of b1 to b2 or b3 or by pouring into b1 from b2 or b3. We have therefore the following constraint:

$$\begin{aligned}
&\texttt{level(b1)}[T_5^+] - \texttt{level(b1)}[T_5^-] = \\
&\quad \Delta\texttt{(pour(a1,b1,b2,}\cdot\texttt{,l1),level(b1)},T_5) + \\
&\quad \Delta\texttt{(pour(a1,b1,b2,}\cdot\texttt{,l2),level(b1)},T_5) + \\
&\quad \Delta\texttt{(pour(a1,b1,b3,}\cdot\texttt{,l1),level(b1)},T_5) + \\
&\quad \Delta\texttt{(pour(a1,b1,b3,}\cdot\texttt{,l2),level(b1)},T_5) + \\
&\quad \Delta\texttt{(pour(a1,b2,b1,}\cdot\texttt{,l1),level(b1)},T_5) + \\
&\quad \Delta\texttt{(pour(a1,b2,b1,}\cdot\texttt{,l2),level(b1)},T_5) + \\
&\quad \Delta\texttt{(pour(a1,b3,b1,}\cdot\texttt{,l1),level(b1)},T_5) + \\
&\quad \Delta\texttt{(pour(a1,b3,b1,}\cdot\texttt{,l2),level(b1)},T_5).
\end{aligned}$$

Of course, in any specific scenario all but at most two of these are 0, because at most two of these events can occur concurrently. In most instances of these constraints, all the terms end up being 0. For this reason, the actual process of solving these constraints is not nearly as difficult as one might guess from just looking at the number and size of the constraints.

E: Conditional effects: If an effect of one of the above types is conditional on expression β then add $\beta[T_i^-]$ as a conjunct on the left side of the above implication.

6.1.2. Preconditions

If action A has precondition β, then add the constraint

$$(1.6) \quad \mathit{active}(A)[T_i] \Rightarrow \beta[T_i^-].$$

For example, one constraint generated by the action "turnOnTap" is

$$\texttt{active(turnOnTap(a1,t2,b1,l3))}[T_5] \Rightarrow \neg\texttt{on(t2)}[T_4] \wedge \texttt{at(a1,l3)}[T_4] \wedge \texttt{at(t2,l3)}[T_4] \wedge \texttt{at(b1,l3)}[T_4].$$

6.1.3. Mutual exclusion

If action A is mutually exclusive (mutex) with action or event E then add the constraint:

$$(1.7) \quad \mathit{active}(A)[T_i] \Rightarrow \neg\mathit{active}(E)[T_i].$$

As mentioned in Section 4, the PDDL+ rules [25] for mutual exclusion are complex, but statically determined.

6.2. Events

6.2.1. Effects

The axioms for the effects of an event have exactly the same form as those for the effects of an action. (Section 6.1.1 above.)

6.2.2. Preconditions

We assume that any numeric precondition of an event is a non-strict (in)equality (that is, of the form $\tau \geqslant 0$ where τ is a term). Otherwise, if there were a precondition $\tau > 0$ where τ was a term involving continuously changing fluents, there would be no first instant at which the precondition became TRUE, and therefore there might be no way in which the event could be triggered at the exact moment of change. The same applies to preconditions of processes.

Let β be the precondition of event E. Add the constraint:

$$(2.1) \quad \mathit{active}(E)[T_i] \Leftrightarrow \beta[T_i^-].$$

For example, suppose that we define the event "breakBucket" in the "Bucket" domain as follows:

```
(:event breakBucket
 :parameters   (?b - bucket)
 :precondition (and (not (broken ?b))
```

```
                    (>= (level ?b) (weight-limit ?b)))
 :effect       (and (broken ?b) (assign (level ?b) 0))
)
```

This gives the constraint:

$$\texttt{active(breakBucket(b2))}[T_5] \Leftrightarrow \texttt{¬broken(b2)}[T_4] \wedge [\texttt{level(b2)}[T_5^-] \texttt{ >= weight-limit(b2)}].$$

6.2.3. Immediate triggering of events by discrete change

Let β be the preconditions of event E. Add the constraint:

$$(2.2) \quad \beta[T_i^+] \Rightarrow [c(T_{i+1}) = c(T_i)].$$

This constraint ensures that when the event E is triggered at T_{i+1} by a discrete change made by actions or events at T_i, it happens immediately, without any finite time duration between the change and the event.

For example, suppose that "(weight-limit b1)" is 55 gallon, that "(level b1)" is 50 gallon at T_{i-1}, and that the atomic action "(pour a1 b2 b1 10 l1)" occurs at T_i. Then the event "breakBucket" must occur at T_{i+1}, and T_{i+1} and T_i must have equal clock times.

Note that the zero crossing axiom (10.7) in Section 6.10.1 assumes that the event is triggered when a numeric precondition attains its threshold value, and therefore does not correctly handle a discrete change that discontinuously pushes a precondition past its threshold value, as in the above example.

6.2.4. Mutual exclusion

Any interference between an action and an event is resolved in a way that gives priority to the event over the action. This is enforced by axiom (2.1) and axiom (1.6): axiom (2.1) asserts that the event *must* occur if the preconditions hold; axiom (1.6) asserts only that the action can be carried out *only if* the preconditions hold. Therefore, if the preconditions of both event E and action A are satisfied, but it is logically inconsistent that both the event and the action should occur, the logical consequence is that the event does occur and that the action therefore does not.

It is the domain designer's responsibility to make sure that events happening at the same time point do not interfere each other; otherwise, the theory is inconsistent.

6.3. Processes

6.3.1. Effects

A: For each process P, for each quantity Q influenced by P, let Φ be the influence of P on the derivative of Q. For each time T_i add the constraints:

$$(3.1) \quad \mathit{active}(P)[T_i] \Rightarrow \Gamma(P, Q, T_i, T_{i+1}) = \Phi \cdot (\mathrm{c}(T_{i+1}) - \mathrm{c}(T_i)).$$

$$(3.2) \quad \neg \mathit{active}(P)[T_i] \Rightarrow \Gamma(P, Q, T_i, T_{i+1}) = 0.$$

Note that Φ must be constant and statically evaluable; otherwise, the system becomes non-linear.

For example, the process "fillingBucket(b2,t3,l2)" generates the constraints:

```
active(fillingBucket(b2,t3,l2))[T5] ⇒
    [ Γ(fillingBucket(b2,t3,l2),level(b2),T5,T6) =
      flow-rate(t3) * (c(T6) - c(T5)) ].
¬active(fillingBucket(b2,t3,l2))[T5] ⇒
    [ Γ(fillingBucket(b2,t3,l2),level(b2),T5,T6) = 0 ].
```

B: For each quantity Q, let $P_1 \ldots P_m$ be the processes that potentially affect Q. Add the constraint:

$$Q[T_{i+1}^-] = Q[T_i^+] + \sum_j \Gamma(P_j, Q, T_i, T_{i+1}). \quad (3.3)$$

For example, suppose there are two taps t1 and t2 and two locations l1 and l2. Then the four processes that might affect "level(b1)" are "fillingBucket(b1,t1,l1)", "fillingBucket(b1,t1,l2)", "fillingBucket(b1,t2,l1)", and "fillingBucket(b1,t2,l2)". Thus we get the constraint:

```
level(b1)[T6⁻] − level(b1)[T5⁺] =
    Γ(fillingBucket(b1,t1,l1),level(b1),T5,T6) +
    Γ(fillingBucket(b1,t1,l2),level(b1),T5,T6) +
    Γ(fillingBucket(b1,t2,l1),level(b1),T5,T6) +
    Γ(fillingBucket(b1,t2,l2),level(b1),T5,T6).
```

6.3.2. Preconditions

Let β be the precondition for process P. Add the constraint:[11]

$$active(P)[T_i] \Leftrightarrow \beta[T_i^+] \wedge \beta[T_{i+1}^-]. \quad (3.4)$$

The atom "active(P)[T_i]" means that P is active over an interval starting with T_i. The condition β must continue to hold over this entire interval. The time point when P terminates must be a significant time point. Hence, β holds both after T_i and before T_{i+1}.

If the process is triggered or terminated by a discrete change, then that change must occur at a significant time point, and hence this axiom will suffice to make P triggered or terminated. If the process is triggered by a continuous change, then the zero crossing axioms given in Section 6.10 below suffice to ensure that the exact moment of change will be constructed as a significant time point.

For example, the process "fillingBucket(b2,t3,l2)" generates the constraint:

```
active(fillingBucket(b2,t3,l2))[T5] ⇔
  filling(t3,b2)[T5] ∧ at(b2,l2)[T5] ∧ at(t3,l2)[T5] ∧
  [level(b2)[T5⁺] ⩽ capacity(b2)] ∧
  [level(b2)[T6⁻] ⩽ capacity(b2)].
```

6.4. Durative actions

6.4.1. Conditions and effects at start and at end

The axioms for these are exactly analogous to those for atomic actions.

[11] The formulation of these axioms in [47] was not quite correct.

6.4.2. Constraints on duration

In PDDL+ it is possible to specify, either that the duration of a durative action is equal to a specified term, or that it is bounded by two specified terms. One can specify that these terms be evaluated either at the beginning of the action (time-annotated as *at start*) or at the end of the action (time-annotated as *at end*). Each such constraint is translated directly into the corresponding constraint on "Duration(A, T_i)".

If a duration constraint is given in the form "(at start β(?duration))" the corresponding axioms have the form:

$$(4.1)\quad \mathit{starts}(A)[T_i] \Rightarrow \beta(\mathit{Duration}(A, T_i))[T_i^-].$$

That is, the instance of action A that starts in T_i has a duration that constrained by β where β is evaluated with values at T_i^-.

Similarly, if a duration constraint is given in the form "(at end β(?duration))" the corresponding axioms have the form:

$$\begin{aligned}(4.2)\quad &[\mathit{starts}(A)[T_i] \wedge \mathit{continues}(A)[T_{i+1}] \wedge \cdots \wedge \mathit{continues}(A)[T_{j-1}] \wedge \\ &\quad \mathit{end}(A)[T_j]] \\ &\Rightarrow \beta(\mathit{Duration}(A, T_i))[T_j^-].\end{aligned}$$

(Here and in axiom (4.3) below, if $j = i + 1$, then there are no "continues" literals in the left-hand side of the implication.)

For example, in the durative action "fillBucket1" given in Example 5.2, the constraint on duration is encoded as the following constraint:

```
starts(fillBucket1(a1,b1,t1,l2))[T2] ∧
       continues(fillBucket1(a1,b1,t1,l2))[T3] ∧
       ends(fillBucket1(a1,b1,t1,l2))[T4]
⇒ [ Duration(fillBucket1(a1,b1,t1,l2),T2) <=
              (capacity(b1) − level(b1)[T4−]) / flow-rate(t1)].
```

6.4.3. Coherence

For a durative action A, for each time T_i, $1 \leqslant \mathrm{i} < N$, add the following constraints:

A: Elapsed time between the starting action and the ending action. Add the constraint for all j, $i < j \leqslant N$:

$$\begin{aligned}(4.3)\quad &[\mathit{starts}(A)[T_i] \wedge \mathit{continues}(A)[T_{i+1}] \wedge \cdots \wedge \mathit{continues}(A)[T_{j-1}] \wedge \\ &\quad \mathit{ends}(A)[T_j]] \\ &\Rightarrow [c(T_j) - c(T_i) = \mathit{Duration}(A, T_i)].\end{aligned}$$

B: A durative action does not continue before the beginning or after the end of the plan. Add the constraint:

$$(4.4)\quad \neg\mathit{continues}(A)[T_1] \wedge \neg\mathit{continues}(A)[T_N].$$

C: For continuity, add the constraint:

(4.5) $starts(A)[T_i] \Rightarrow continues(A)[T_{i+1}] \vee ends(A)[T_{i+1}]$.

(4.6) $ends(A)[T_i] \Rightarrow continues(A)[T_{i-1}] \vee starts(A)[T_{i+1}]$.

(4.7) $continues(A)[T_i] \Rightarrow ends(A)[T_{i+1}] \vee continues(A)[T_{i+1}]$.

(4.8) $continues(A)[T_i] \Rightarrow starts(A)[T_{i-1}] \vee continues(A)[T_{i-1}]$.

6.4.4. Invariant conditions

Let β be the invariant conditions for a durative action A. Add the constraint:

(4.9) $continues(A)[T_i] \Rightarrow \beta[T_i^+]$.

(4.10) $starts(A)[T_i] \Rightarrow \beta[T_i^+]$.

Termination (i.e., from TRUE to FALSE) of the invariant conditions at a "continues" point by continuously changing quantities is handled by axiom of zero crossing from TRUE to FALSE, axiom (10.11) in Section 6.10.1.

6.4.5. Continuous effects over the period of a durative action

The axiom for continuous effects of a durative action are exactly analogous to the axioms given in Section 6.3.1 for the continuous effects of a process.

"starts$(A)[T_i]$" and "continues$(A)[T_i]$" initiate a continuous change over T_i and T_{i+1}.

6.5. Frame axioms

6.5.1. Propositional or interval fluents

For any fluent F let $A_1 \ldots A_k$ be the actions and events that potentially change F. For each time T_i, for each value V of F, add the constraint:

(5.1) $\neg active(A_1)[T_i] \wedge \cdots \wedge \neg active(A_k)[T_i] \Rightarrow F[T_i] = F[T_{i-1}]$.

6.5.2. Numeric fluents

No additional frame axioms are needed. If no atomic actions or events that change quantity F are active at time T_i, then all the terms in the sum in equation of axiom (1.5) will be 0, so the equation will state that F does not change. If no processes or durative actions that change F are continuing between T_i and T_{i+1}, then all the terms in the sum in Eq. (3.3) will be 0, so the equation will state that F does not change.

6.6. Gap between significant time points

In Level 4, we have the constraint that, for each T_i,

(6.1) $c(T_{i+1}) - c(T_i) \geqslant \varepsilon$.

In Level 5, we have the constraint[12] that, for each T_i,

(6.2) $c(T_{i+1}) \geqslant c(T_i)$.

[12] See Section 4.

6.7. Reusable metric resources

The encoding we give here is for a finite resource shared among durative actions. The encoding for sharing resources among atomic actions or in a mixed collection of atomic actions and durative actions is given in [46]; the latter uses two variables for the resource level at each time point. An example would be where a robot with multiple identical manipulators must use some of them for durative actions, such as carrying a tray, and concurrently use others for atomic actions, such as flipping a light switch.

Recall that "$U(R, A, T_i)$" denotes the amount of R that A uses at time T_i.

6.7.1. Resource allocation and deallocation

For any numeric resource R, and durative action A, let β be the expression describing the amount of R that A would use during its period. Add the constraints:

$$(7.1)\quad \mathit{starts}(A)[T_i] \Rightarrow U(R, A, T_i) = \beta[T_i^-].$$

$$(7.2)\quad \neg\mathit{starts}(A)[T_i] \Rightarrow U(R, A, T_i) = 0.$$

$$(7.3)\quad \mathit{ends}(A)[T_j] \wedge \mathit{starts}(A)[T_i] \Rightarrow U(R, A, T_j) = -\beta[T_i^-].$$

$$(7.4)\quad \neg\mathit{ends}(A)[T_j] \Rightarrow U(R, A, T_j) = 0.$$

6.7.2. Propagation

For any resource R let $A_1 \ldots A_k$ be the actions that could use R; let "$L(R, T_i)$" be the level of resource R at T_i. Add the constraint:

$$(7.5)\quad L(R, T_i) = L(R, T_{i-1}) - \sum_j U(R, A_j, T_i).$$

6.7.3. Capacity constraint

For each time T_i, add the constraint:

$$(7.6)\quad 0 \leqslant L(R, T_i) \leqslant \mathit{capacity}(R).$$

6.8. Intervals

Predicates and functions over intervals can be translated in the standard way into (in)equalities and functions over their endpoints [12,46].

6.9. Reusable interval resources

Recall that "lower(A, R, T_i)" and "upper(A, R, T_i)" be the lower and upper bounds of the segment of R allocated to A at T_i. Let "left(R)" and "right(R)" be the lower and upper bounds of interval resource R.

6.9.1. Segment allocation

$$(9.1)\quad \mathit{starts}(A)[T_i] \Rightarrow [\mathit{upper}(A, R, T_i) - \mathit{lower}(A, R, T_i) = \beta[T_i^-]].$$

(9.2) $lower(A, R, T_i) \geqslant left(R)$.

(9.3) $upper(A, R, T_i) \leqslant right(R)$.

6.9.2. Frame axiom: Segments don't move

(9.4) $continues(A)[T_i] \Rightarrow [lower(A, R, T_{i+1}) = lower(A, R, T_i)]$.

(9.5) $continues(A)[T_i] \Rightarrow [upper(A, R, T_{i+1}) = upper(A, R, T_i)]$.

(9.6) $starts(A)[T_i] \Rightarrow [lower(A, R, T_{i+1}) = lower(A, R, T_i)]$.

6.9.3. Non-overlap

Let A_1 and A_2 be two distinct durative actions that use R.

(9.7) $continues(A_1)[T_i] \wedge continues(A_2)[T_i]$
$\Rightarrow [[lower(A_2, R, T_i) \geqslant upper(A_1, R, T_i)] \vee$
$[lower(A_1, R, T_i) \geqslant upper(A_2, R, T_i)]]$.

6.10. Zero crossings

6.10.1. Triggering/terminating by continuous change

One final type of constraint is rather trickier. This has to do with an event or process being triggered or terminated by a continuously changing numerical fluent attaining a particular value.[13] Suppose that process P1 is active between times T_a and T_b and is steadily increasing the value of fluent Q; that process P2 will be triggered when Q reaches value V; and that this transition will occur at a time T_x *between* T_a and T_b. Suppose, further, that in the absence of P2, no significant change would occur between T_a and T_b, so they would be consecutive significant time points. The problem is, how do we force the system of constraints to recognize the time point T_x? That is, how can we prevent the system from accepting a solution in which T_a and T_b are consecutive time points and process P2 starts at time T_b? (Worse yet, consider a case where P2 is only triggered if Q is between V1 and V2; Q is less than V1 at time T_a and Q is greater than V2 at time T_b. Then the system of constraints will discover that P2 is not triggered at time T_a and not triggered at time T_b and will conclude that it never occurs at all.)

The same thing can happen, in the reverse direction, with the numeric conditions of processes and the "over all" conditions of durative actions: We must check that they continue to hold throughout the interval, not just that they hold at the endpoints.

The solution rests on the fact that all numeric conditions are Boolean combinations of linear constraints, and that, within our domains, any numeric fluent that changes continuously is a linear function of time. A simple solution, therefore, is as follows: Assume that every numerical constraint that appears as any kind of precondition for events or processes has the form $Q(t) \geqslant 0$, where $Q(t)$ is a linear function of the numerical variables and of

[13] It would appear, though the point is not entirely clear, that in the definition of PDDL+ Level 5, one process cannot directly trigger another, nor can one process terminate another or itself; such an interaction must be mediated by an event. We do not see what purpose this restriction serves, so we have not required it.

time t. We can "track" each such constraint $Q(t)$ and make sure that we "notice" whenever any such constraint becomes TRUE or becomes FALSE by asserting that it does not change from positive to negative or vice versa without an intermediate "significant time point" where it is zero. This gives the following two constraints:

$$(10.1)\quad \neg[(Q[T_i^+] > 0) \wedge (Q[T_{i+1}^-] < 0)].$$

$$(10.2)\quad \neg[(Q[T_i^+] < 0) \wedge (Q[T_{i+1}^-] > 0)].$$

These are respectively equivalent to

$$(10.3)\quad Q[T_i^+] > 0 \Rightarrow Q[T_{i+1}^-] \geqslant 0.$$

$$(10.4)\quad Q[T_i^+] < 0 \Rightarrow Q[T_{i+1}^-] \leqslant 0.$$

This is just a continuity constraint over Q of a form familiar from qualitative process theory [22].

The problem with these constraints is that they will generate lots of spurious time points, where a constraint of this form becomes TRUE or FALSE, but no actual event or process is triggered, because the constraint is only one of a set of preconditions and the other preconditions are not TRUE. Generating spurious time points is extremely undesirable, of course, because the number of propositional atoms and the size of the constraint set is proportional to the number of time points.

We need, therefore, to rephrase the above constraints in such a way that they will generate a significant time point only when a numerical constraint changes its truth value and thereby causes an entire set of preconditions to changes its truth value. We will first deal with the case where a truth value changes from FALSE to TRUE and then with the case where it changes from TRUE to FALSE. First, we put every precondition of an event or process into disjunctive normal form; that is, we express it as the disjunction of a collection of conjuncts. (This in itself can be a fairly complex manipulation of the PDDL+, especially in the case of conditional expression.) Now, consider any such conjunct:

$$F_1 \wedge \cdots \wedge F_k \wedge Q_1 \geqslant 0 \wedge \cdots \wedge Q_m \geqslant 0,$$

where the F_i are literals and the Q_i are linear functions.

What we wish to assert is that, if this condition is not satisfied at T_i, then it remains unsatisfied until T_{i+1}; equivalently, if it is satisfied at any time T between T_i and T_{i+1} then it is satisfied at T_i^+. Symbolically,

$$(10.5)\quad \left[\exists_{T\in(T_i,T_{i+1})} \bigwedge_p F_p[T] \wedge \cdots \wedge_p Q_p[T] \geqslant 0\right] \Rightarrow \bigwedge_p F_p[T_i^+] \wedge \cdots \wedge_p Q_p[T_i^+] \geqslant 0.$$

We now have to convert the quantified formula on the left hand side of this implication to an evaluable expression. This is done as follows:

- The values of the F_p do not change between two consecutive significant time points. That is, $F_p[T] = F_p[T_i]$.

- Since between any two significant time points the Q_p are all linear and hence monotonic functions of time, we know that, if $Q_p[T] \geqslant 0$ then either $Q_p[T_i^+] > 0$ or $Q_p[T_{i+1}^-] > 0$ or $Q_p[T_i^+] = Q_p[T_{i+1}^-] = 0$.

Hence the following axiom is sufficient to achieve the above condition:

$$\begin{aligned}(10.6)\quad &\Bigg[\bigwedge_p F_p[T_i] \wedge \\ &\qquad \bigwedge_p [Q_p[T_i^+] > 0 \vee Q_p[T_{i+1}^-] > 0 \vee Q_p[T_{i+1}^-] = Q_p[T_i^+] = 0]\Bigg] \\ &\quad \Rightarrow \bigwedge_p Q_p[T_i^+] \geqslant 0.\end{aligned}$$

The following set of axioms[14] is slightly stronger, but substantially simpler: for each Q_j, assert

$$\begin{aligned}(10.7)\quad &\Bigg[\bigwedge_p F_p[T_i] \wedge Q_j[T_i^+] < 0 \wedge \bigwedge_{p \neq j} [Q_p[T_i^+] \geqslant 0 \vee Q_p[T_{i+1}^-] \geqslant 0]\Bigg] \\ &\quad \Rightarrow Q_j[T_{i+1}^-] \leqslant 0.\end{aligned}$$

The logical relations between the above axioms is that [the conjunction over i of axioms (10.3)] implies [the conjunction over j of axioms (10.7)] which further implies axiom (10.6). Since axiom (10.6) implies (10.5), the conjunction of (10.7) implies (10.5). That means that if we enforce (10.7), that will enforce (10.5) and ensure that no significant zero crossing are missed. On the other hand, since (10.3) implies (10.7), that means that (10.7) can be satisfied if there are enough time points to satisfy (10.3)—i.e., there is a time point for every zero crossing of the constraints. That, however, is a worst-case upper bound; in practice, (10.7) generates few if any time points that are not significant.

(The proof of the above implications is as follows. That axiom (10.3) implies axiom (10.7) is trivial, as axiom (10.7) differs from axiom (10.3) only in having additional conditions on the left side of the implication. That axiom (10.6) implies axiom (10.5) was discussed above. That axiom (10.7) implies axiom (10.6) can be justified as follows. Axiom (10.7) has the form

$$(10.8)\quad \beta \wedge Q_j[T_i^+] < 0 \Rightarrow Q_j[T_{i+1}^-] \leqslant 0.$$

Taking the contrapositive of the conditions on Q_j we have

$$(10.9)\quad \beta \wedge Q_j[T_{i+1}^-] > 0 \Rightarrow Q_j[T_i^+] \geqslant 0.$$

Now, since trivially

$$Q_j[T_i^+] = Q_j[T_{i+1}^-] = 0 \Rightarrow Q_j[T_i^+] \geqslant 0 \quad \text{and} \quad Q_j[T_i^+] > 0 \Rightarrow Q_j[T_i^+] \geqslant 0,$$

[14] The formulation of these axioms in [47] was not quite right.

axiom (10.9) is equivalent to

$$(10.10)\quad \beta \wedge [Q_j[T_{i+1}^-] > 0 \vee Q_j[T_i^+] > 0 \vee Q_j[T_i^+] = Q_j[T_{i+1}^-] = 0] \Rightarrow Q_j[T_i^+] \geqslant 0.$$

Now, β is the conjunction

$$\bigwedge_p F_p[T_i] \bigwedge_{p \neq j} [Q_p[T_i^+] \geqslant 0 \vee Q_p[T_{i+1}^-] \geqslant 0].$$

We can weaken axiom (10.10) by strengthening the condition in β on the left-hand side of the implication. Specifically, we replace

$$[Q_p[T_i^+] \geqslant 0 \vee Q_p[T_{i+1}^-] \geqslant 0] \quad \text{by}$$
$$[Q_p[T_{i+1}^-] > 0 \vee Q_p[T_i^+] > 0 \vee Q_p[T_i^+] = Q_p[T_{i+1}^-] = 0].$$

Substituting these forms in the left hand side of axiom (10.10), and combining the constraint on Q_j with the same constraints on Q_p where $p \neq j$ gives us axiom (10.6). End of proof.)

The case of change from TRUE to FALSE applies in somewhat different cases. On the one hand, the preconditions of events do not have to be checked. As soon as the precondition of an event becomes TRUE, it is executed and necessarily "turns off" its own precondition; hence, these never become FALSE by virtue of the change to a continuous fluent. On the other hand, the invariant conditions of durative actions do have to be checked. We do not have to detect zero crossings for durative actions from FALSE to TRUE, because a durative action is optional, and if the planner decided to execute it, then a time variable for its starting time will be generated. On the other hand, the invariant conditions for a durative action could change from TRUE to FALSE and then back to TRUE between T_i and T_{i+1}, and that should be detected and marked as impossible.[15]

To construct the axiom for checking for changes from TRUE to FALSE, we simply "run time backward"; if a precondition changes from TRUE to FALSE when time is run in the positive direction, then it changes from FALSE to TRUE when time is run backward. It suffices, therefore, just to exchange T_{i+1}^- and T_i^+ in the numerical terms in axiom (10.7):

$$(10.11)\quad \left[\bigwedge_p F_p[T_i] \wedge Q_j[T_{i+1}^-] < 0 \wedge \bigwedge_{p \neq j} [Q_p[T_i^+] \geqslant 0 \vee Q_p[T_{i+1}^-] \geqslant 0]\right] \Rightarrow Q_j[T_i^+] \leqslant 0.$$

The effect of these constraints is, essentially, to generate the necessary intermediate time points by a sort of proof by contradiction, but a logic-based system such as TM-LPSAT has no trouble with proof by contradiction.

For example: The process "fillingBucket(b1,t2,l2)" has the propositional conditions "filling(t2,b1)", "at(t2,l2)" and "at(b1,l2)", and the numeric condition "capacity(b1)–

[15] This can happen if there is a disjunctive precondition that depends on a continuously increasing fluent, such as $((\geqslant 2Q) \vee (\leqslant Q\ 4))$.

level(b1) $\geqslant 0$". These conditions therefore generates the two constraints:

$$[\ \texttt{filling(t2,b1)}[T_5] \wedge \texttt{at(t2,l2)}[T_5] \wedge \texttt{at(b1,l2)}[T_5] \wedge [\ \texttt{capacity(b1)}-\texttt{level(b1)}[T_5^+] < 0\ ]\ ] \Rightarrow [\ \texttt{capacity(b1)}-\texttt{level(b1)}[T_6^-] \leqslant 0\ ].$$

$$[\ \texttt{filling(t2,b1)}[T_5] \wedge \texttt{at(t2,l2)}[T_5] \wedge \texttt{at(b1,l2)}[T_5] \wedge [\ \texttt{capacity(b1)}-\texttt{level(b1)}[T_6^-] < 0\ ]\ ] \Rightarrow [\ \texttt{capacity(b1)}-\texttt{level(b1)}[T_5^+] \leqslant 0\ ].$$

Putting together the constraints from all these categories, it seems like a lot of constraints, and in many cases it is. But things are not quite as bad as they look. For any given plan, many of the numerical variables are of the form $\Delta[A, Q, T_i]$ where A is inactive at T_i or $\Gamma[P, Q, T_i, T_{i+1}]$ where P is inactive between T_i and T_{i+1}, and are therefore equal to 0. Many of the constraints turn out to be equations between variables or between a variable and a constant; these can be eliminated by variable renaming and constant folding. Others are difference constraints of the form $V_i - V_j \geqslant C$ where V_i and V_j are variables and C is a constant; these are also easy to deal with [3].

6.10.2. Extended example of zero crossing

Let us give an artificial example to illustrate how the above zero crossing constraints work. Suppose that we have the following world: There is a numeric fluent N and two Boolean fluents P and Q. Process R is always active and causes N to grow at the rate of 1 unit per second. Event E is triggered if $1 \leqslant N \leqslant 2$ and P is TRUE, and it causes P to be FALSE. Event F is triggered if $N \geqslant 3$ and P is TRUE, and it causes Q to be TRUE and P to be FALSE. Action A has no precondition and causes P to be TRUE. Initially $\mathrm{N} = 0$, P is TRUE and Q is FALSE. The goal is that Q should be TRUE.

Note that only event F can bring about Q, and that F can only occurs if P is TRUE and the time is at least 3. Any time between 1 and 2, if P becomes TRUE, it will immediately cause P to be FALSE. Therefore the correct plan is to wait until any time after 2 and then execute A to make P TRUE. F will then occur at time 3 (or immediately after A, if A was executed later than time 3).

The PDDL+ representation of this world is shown in Table 2. The corresponding set of axioms is shown in Table 3.

Note that, if we omit the zero crossing axioms 17 and 18 in Table 3, there would be a solution with two time points[16], T_1 at clock time 0 and T_2 at clock time 3. The remaining axioms do not "notice" that E would be triggered in between. At clock time 3, since P is still TRUE, F will be triggered, and will cause Q to be TRUE. Table 4 shows this "solution" symbolically. However, the zero crossing axiom 17 excludes this solution; the left hand side of the implication is TRUE, and the right hand is FALSE.

Indeed, it is easily shown that there is no solution to the axioms with only two time points. Since $\neg\mathrm{Q}[T_1]$ and $\mathrm{Q}[T_2]$ by 14 we have active$(F)[T_2]$. By 6 we have $\mathrm{N}[T_2^-] \geqslant 3$.

[16] T_0 denotes the initial state. A plan starts at T_1.

Table 2
PDDL+ representation of extended example

```
(define (domain Extended-Example)
(:requirements :time)
(:predicates P Q)
(:functions (N) - fluent)
(:action A
 :parameters ()
 :precondition ()
 :effect P
)
(:event E
 :parameters ()
 :precondition (and (<= 1 N) (<= N 2) P)
 :effect (not P)
)
(:event F
 :parameters ()
 :precondition (and (<= 3 N) P)
 :effect (and Q (not  P))
)
(:process R
 :parameters ()
 :precondition ()
 :effect (increase N (* #t 1))
)
)
(define (problem EE-problem)
(:domain Extended-Example)
(:requirements :time)
(:inits  (= (N) 0)
         P )
(:goal  Q )
)
```

By 4 we have $\neg$active(E)[T_1]. Using 15, 1, and 20 we have P[T_2]. But now we have a contradiction with 17.

Similarly, there is no solution with three time points. There is a solution with four time points, shown symbolically in Table 5 corresponding to the plan described above.

7. Soundness and completeness

We have proven a soundness and completeness proof for TM-LPSAT over a restricted class of problems in our extended version of PDDL+. Stating the proof involves the following three steps.

First, we must give a suitable definition of the semantics of PDDL+ Level 5. To allow the possibility of multiple time points with the same clock time, as described in Section 4, we use the following non-standard temporal model: A *time point* is a pair $\langle X, N\rangle$ where X is a positive real number (the clock time) and N is a positive integer (the N'th time points at

Table 3
Axioms for extended example

1. active(A)[T_i] $\Rightarrow$ P[T_i]. (Effect of A. Axiom (1.1))
2. active(A)[T_i] $\Rightarrow$ TRUE. (Precondition of A – vacuous. Axiom (1.2))
3. active(E)[T_i] $\Rightarrow \neg$P[T_i]. (Effect of E. Section 6.2.1)
4. active(E)[T_i] $\Leftrightarrow$ P[T_{i-1}] $\wedge\ 1 \leqslant N[T_i^-] \wedge N[T_i^-] \leqslant 2$.
   (Precondition of E. Axiom (2.1))
5. active(F)[T_i] $\Rightarrow$ Q[T_i] $\wedge \neg$P[T_i]. (Effect of F. Section Section 6.2.1)
6. active(F)[T_i] $\Leftrightarrow$ P[T_{i-1}] $\wedge\ 3 \leqslant N[T_i^-]$. (Precondition of F. Axiom (2.1))
7. active(A)[T_i] $\Rightarrow \neg$active(E)[T_i].
8. active(A)[T_i] $\Rightarrow \neg$active(F)[T_i].
   (7 and 8 are mutex conditions. In this case, they are redundant. Section 6.2.4)
9. N[T_i^+] − N[T_i^-] = 0.
   (Frame axioms for N at significant time points. Axiom (1.5).
   Since there are no actions or events that affect N, the sum on the right
   is taken over the null set.)
10. active(R)[T_i] $\Rightarrow \Gamma$(R,N,T_i, T_{i+1}) = 1 · (c(T_{i+1}) - c(T_i)).
    (Direct influence of process R on N. Axiom (3.1))
11. $\neg$active(R)[T_i)] $\Rightarrow \Gamma$(R,N,T_i, T_{i+1}) = 0.
    (Influence of process R on fluent N. Axiom (3.2))
12. N[T_{i+1}^-] − N[T_i^+] = Γ(R,N,T_i, T_{i+1}).
    (Net effect of processes on N. Axiom (3.3))
13. active(R)[T_i)] $\Leftrightarrow$ TRUE. (Precondition of R. Axiom (3.4))
14. $\neg$active(F)[T_i] $\Rightarrow$ Q[T_i] $\Leftrightarrow$ Q[T_{i-1}]. (Frame axiom for Q. Axiom (5.1))
15. [$\neg$active(A)[T_i] $\wedge \neg$active(E)[T_i] $\wedge \neg$active(F)[T_i]] $\Rightarrow$ [P[T_i] $\Leftrightarrow$ P[T_{i-1}]].
    (Frame axiom for P. Axiom (5.1))
16. c(T_{i+1}) $\geqslant$ c(T_i). (Sequence of time points. Axiom (6.2))
17. P[T_i] $\wedge$ N[T_i^+]−1 < 0 $\wedge$ [2−N[T_i^+] $\geqslant 0 \vee$ 2−N[T_{i+1}^-] $\geqslant 0$] $\Rightarrow$ N[T_{i+1}^-]−1 $\leqslant 0$.
    (First zero crossing rules for precondition of E. Axiom (10.7))
18. P[T_i] $\wedge$ 2−N[T_i^+] < 0 $\wedge$ [N[T_i^+]−1 $\geqslant 0 \vee$ N[T_{i+1}^-]−1 $\geqslant 0$] $\Rightarrow$ 2−N[T_{i+1}^-] $\leqslant 0$.
    (Second zero crossing rules for precondition of E. Axiom (10.7))
19. P[T_i] $\wedge$ N[T_i^+]−3 < 0 $\Rightarrow$ N[T_{i+1}^-]−3 $\leqslant 0$.
    (Zero crossing rule for precondition of F. Axiom (10.7))
20. P[T_0] $\wedge \neg$Q[T_0] $\wedge$ N[T_0^+] = 0. (Initial state.)
21. Q[T_K]. (Goal.)

Table 4
Solution to constraints with no zero crossing axiom

Time	c(T_i)	active(A)	active(E)	active(F)	active(R)	N[T_i^-]	N[T_i^+]	P	Q
0	0	F	F	F	F	0	0	T	F
1	0	F	F	F	T	0	0	T	F
2	3	F	F	T	T	3	3	F	T

Table 5
Correct solution

Time	c(T_i)	active(A)	active(E)	active(F)	active(R)	N[T_i^-]	N[T_i^+]	P	Q
0	0	F	F	F	F	0	0	T	F
1	0	F	F	F	T	0	0	T	F
2	1	F	T	F	T	1	1	F	F
3	3	T	F	F	T	3	3	T	F
4	3	F	F	T	T	3	3	F	T

that clock time). Time points are ordered lexicographically; that is, $\langle X1, N1\rangle < \langle X2, N2\rangle$ iff $X1 < X2$ or $[X1 = X2$ and $N1 < N2]$. A *history* is a mapping that maps any time point within a bounded interval to the values of the fluents at that point, and the sets of actions, events, and processes that are active at that point. A history is *consistent* with a PDDL+ domain description if it obeys the rules set forth in the description. A *plan* is a mapping that maps any time point to the set of actions that are executed at that point.

History H is a *projection* of plan P starting in situation S0 relative to domain description D if

- H and P specify the same actions at the same times.
- H is consistent with D.
- S0 is the starting state of H.

A *goal* is a property of histories. A *planning problem* is a specification of a starting state, a goal, and a domain description. A plan is a *correct solution* of a planning problem if every projection of the plan from the starting state relative to the domain description satisfies the goal.

Note that constraint-based planning techniques give correct results only if the only source of uncertainty is the actions to be carried out; once the actions are specified, there is only one possible projection. If there is more than one possible projection, or if there is anything unspecified in the starting state, then a constraint-based planner will make the most optimistic assumptions about these; that is, it will set these uncontrolled parameters in the same way that it sets the actions to be carried out.

Second, we must properly delimit the class of problems. A problem is a *candidate* for TM-LPSAT if the following two conditions are satisfied:

- Let H be a history. We say that time point T is *significant* in H if either at least one action is executed at H(T); at least one event occurs at H(T); or at least one process begins or ends at H(T). A planning problem is *finitely solvable* if there exists a history that satisfies the problem with finitely many significant time points.
- Every arithmetic function that appears in the PDDL+ domain description is a linear function of the numeric fluents and non-constant numeric parameters involved with constant coefficients.

Note that the first condition is a *semantic* constraint over the class of histories considered, and that the second condition is a *syntactic* constraint over the form of the PDDL+ description. Moreover, it is in general only semi-decidable whether the semantic constraint holds. That is inelegant, but there does not seem to be any way around it.

Third, we must use the right notion of "completeness". (There are several different possible notions of what it means for a planner to be complete.) TM-LPSAT is complete in the following sense: Let G be any planning problem that is a candidate for TM-LPSAT. Then if TM-LPSAT is executed with a sufficient number of time points, it will return a plan that is a valid solution to the problem.

In Appendix B, we give an extensive, though not fully formal, account of the semantic definition, of the precise statement of the theorem, and of the proof.

8. Conclusions and future work

There exist very few domain-independent planners that can handle problems involving continuous change to numeric quantities. The TM-LPSAT planner demonstrates that the SAT-based planning framework can be extended to deal with such problems. Other features incorporated in TM-LPSAT include real-valued and interval-valued fluents, exogenous events and processes, atomic and durative actions with numeric parameters, and reusable metric and interval resources. To permit the representation of some of these features, we have introduced a number of extensions into the PDDL+ description language. We have tested our encoding generated by the TM-LPSAT compiler, using different SAT-based arithmetic constraint solvers, on a number of problems of varying complexity and characteristics. We have proven that TM-LPSAT is sound and complete for a significant subset of this extended description language.

The contributions of our work are:

- We have shown that the SAT-based planning framework can be used for reasoning about continuous change. This disproves previous claims, cited in Section 2.1, that this would be impossible.
- The capability of TM-LPSAT for dealing with issues typical of scheduling problems, such as metric quantities and reusable resources, suggests that SAT-and-LP-based planning techniques may offer a bridge spanning the divide between domain-independent planning and scheduling.
- Our approach to dealing with continuous change and continuous time—specifically, the characterization of overall behavior in terms of the values of fluents at "significant" time points—may also be applicable to other planning methodologies, such as Graphplan [7].

The current version of TM-LPSAT has the following limitations:

- The existing SAT-based arithmetic constraint solvers can only deal with Boolean combinations of linear (in)equalities and propositional atoms. This makes it necessary to require that all numeric terms in preconditions and in effects are linear functions of fluents, and that any continuous effect of processes and durative actions is a constant influence on the derivative of the affected fluent.
- It is not possible in TM-LPSAT to specify a given plan metric to be optimized. This limitation is inherited from the architecture of the constraint solvers we used, such as LPSAT [53] or MathSAT [3,9]. (The search strategy in TM-LPSAT will return a plan with the minimum number of significant time points; but this is not even the same as the plan with the minimum number of actions, let alone any other metric.)
- Neither the compilation process nor the encoding is optimized in the current version of TM-LPSAT.

- Scalability is certainly a concern, as in all SAT-based planners. The question is, how far can you go, using optimized encodings and heuristics for constraint solvers based on domain characteristics, before running into intractable combinatorial explosion.

Some thoughts on overcoming these limitations:

As regards non-linear constraints: It would certainly be possible using current technology to develop an more powerful arithmetic constraint solver that could solve Boolean combinations of non-linear constraints. How effective such a solver could be made, we cannot guess. If such an engine were constructed, then that would allow two easy extensions to TM-LPSAT. First, obviously, it would make it possible to use non-linear arithmetic terms in preconditions and on the right-hand side of assignment and increment effects. Second, a little more subtly, it would allow processes whose continuous effect on a output fluent is constant, but depends on a numeric fluent set by a atomic action or event. For example, the "Bucket" domain could be modified to allow the agent to turn on a tap to any desired level of flow-rate. In the Zeno domain of [53], it would be possible to set a desired constant speed for the airplane, which would affect both the rate of motion and the rate of fuel consumption. Since, in such domains, all numeric fluents would be piecewise linear functions of time, with breaks only at significant time points, the TM-LPSAT compilation rules, including the zero crossing rules, would still be valid; indeed, the proof of their soundness and completeness would be essentially unchanged.

Extensions beyond that would involve substantially greater difficulties. If the effect of a process depends on a fluent whose value changes continuously, then there is a differential equation to be solved, particularly if the dependencies have a cycle. If numeric fluents can be non-linear functions of time, then it becomes hard to guarantee that they are monotonic functions of time; and if they are not monotonic, then the zero crossing rules of Section 6.10.1 are insufficient. The constraint solver would need to incorporate a zero crossing detector.

We can think of a couple of approaches to adapt TM-LPSAT to plan optimization.[17] If an upper bound is placed on N, the number of time points, then an objective function M can be optimized by adding a constraint of the form $M \leqslant B$ where B is a constant, and doing binary search to find the smallest possible value of B. However, this strategy cannot be used to find the overall optimal plan, where N is not bounded. In principle, of course, one could dovetail the search for B and N; but such dovetailing is surely more suited to proofs in computation theory than to practical programming.

A more promising approach would be to modify the interaction between the SAT solver and the LP solver in the arithmetic constraint solver to use branch and bound [31]. The problem space consists of a collection of states. Each state is represented by two sets of

[17] One difficulty about plan optimization is that, in domains as rich as these, there may not exist any optimal plan; it is easy to construct problems in which plans can be made better and better as the number of actions increases, as the number of significant time points increases, as the duration of the plan goes either to infinity or to a finite limit, or as the value of a numeric action parameter goes either to infinity or to a finite limit. Moreover, even if an optimal solution is known to exist—e.g., the metric is always a positive integer, and one is searching for a minimum—proving that a particular solution is optimal may be undecidable.

constraints, one of logical constraints and the other of mixed logical linear constraints, such that the two sets share action variables appearing in the mixed constraints. The relaxed version of mixed constraints is solved, only if the set of logical constraints is satisfiable; the optimal value of the relaxed constraints gives the lower bound (for a minimization problem) of the mixed constraints so that it can be used either to bound the optimal value of the problem (if the values of action variables in the solution satisfy the set of logical constraints), to prune the search space, or to branch from the current state. This approach, we believe, may prune the search space quickly due to the action variables shared between the two sets of constraints. The major challenge is to derive heuristics to decide a branching variable or to pick up a state to solve next. Currently we are working on this approach using an incremental SAT solver and an LP solver.

Optimization techniques known for the propositional domains may be extended for temporal metric domains: adding domain axioms, such as state constraints, that are inferred through domain analysis [26] as preprocessing step; reducing encoding size by removing unnecessary action instantiations through type analysis [23] at compilation stage; simplifying binary clauses in the encoding produced by the compiler, which are generally numerous in the SAT encoding of planning [10].

In the SAT-based arithmetic constraint solver, it is known that the running time is dominated by the time consumed by the LP solver [1,54]. In order to reduce calls to the LP solver, one optimization technique at the encoding (compiling) phase would be to lift arithmetic constraints mutually exclusive up to the Boolean level. In the example of Section 6.10.2, the inequalities $N[T_i] \leqslant 2$ in axiom 4 of Table 3 and $3 \leqslant N[T_i]$ in axiom 6 are mutually exclusive. If we were to add this mutual exclusion as a clause, then the SAT solver could not assign them (or, more precisely, their Boolean triggers) TRUE in a partial propositional solution. Thus, it would never be necessary to pass the two of them together to the LP solver to find the inconsistency. The detection of arithmetic constraints mutually exclusive in the encoding can generally be done as a preprocessing step in a constraint solver, called *static learning*. It, however, is certainly more expensive than at the encoding phase. The layered structure adopted in MathSAT [3,9] is also an effective approach to reduce unnecessary calls to the LP solver (or, generally computationally more expensive routines).

It was known that a branching heuristic utilizing characteristics specific to encoding of planning domains (i.e. nondeterminism on choices of actions) can drastically reduce running times [28]. Our preliminary experiments also show that different branching heuristics could make a big difference in running time of a planning problem in temporal metric domains. We, however, do not know any SAT solver or SAT-based arithmetic constraint solver specially tailored for planning domains. Our ongoing work on branch-and-bound architecture built on LP and SAT solvers is to combine completeness and these heuristics coming from planning domains.

We have done some preliminary experiments[18] with IPC3 problem domains [39] and "Bucket" domain, and a number of SAT-based arithmetic solvers based on different algorithms. The intention of the experiments was to observe the feasibility of our encoding, the

[18] The report is available at http://cs.nyu.edu/~jiae/papers/experiments.pdf.

scalability of this approach in temporal metric domains, and how features of different constraint solvers react with characteristics of constraints specific to different kinds of planning (metric planning in discrete time, temporal planning with durative actions, temporal metric planning of real-time model).

The difficulty of solving a constraint set generally grows rapidly with its size, and shows striking difference in performance among different constraint solvers. This is particularly apparent in the problems of the "Bucket" domain, which are more constrained, and which intertwine metric constraints with temporal constraints. For temporal planning problems in which the arithmetic constraint set consists mostly of equality equations, we have found that the solver with a specialized routine for equality equation, a variant of Bellman–Ford algorithm, in the layered and delayed architecture [3,9] performs considerably better than others. Currently the constraint solver is used as a black box and advanced analysis is left for future work.

Finally, we plan to explore some further applications of this type of planning. It should be possible to implement some kinds of spatial reasoning by allowing region-valued fluents and motion as a process. If regions are restricted to polygons or polyhedra, either fully specified, or of a specified maximum complexity, and all motions are constant-velocity translations, then it should be possible to compile these domains into systems of linear constraints.

Acknowledgements

Our special thanks go to Steven Wolfman and Daniel Weld for making their LPSAT program available for our research. The MathSAT group kindly allowed us access to the versions of MathSAT. We also would like to thank the anonymous reviewers for valuable suggestions and comments. The research reported in this paper was supported by NSF grant IIS-0097537.

Appendix A. The bucket domain and problem in an extended PDDL+

A.1. The bucket domain

```
;; ================================================================
;; "Bucket" Domain:                                               ;;
;;       Deliver a specified amount of water to a specified       ;;
;;       location(s) by a specified deadline.                     ;;
;;                                                                ;;
;;  Assumptions:                                                  ;;
;;       - Zero or more than one tap are in each location.        ;;
;;       - Each tap fills only one bucket at a time.              ;;
;;       - Each bucket can be filled by more than one tap         ;;
;;         in a location at a time, allowing concurrent continuous ;;
;;         change on the level of a bucket.                       ;;
;; ================================================================
```

```
(define  (domain  Buckets)
(:requirements  :time  :continuous-effects)
(:types         agent  bucket  tap  location)
(:predicates    (at ?o - (either agent bucket tap)  ?l - location)
                (on ?t - tap)
                (filling ?t - tap ?b - bucket)
                (carrying ?a - agent ?b - bucket)
                (is_walking ?a - agent ?d - location)
                (connected ?s - location ?d - location)
)

(:functions     (capacity ?b - bucket) - float
                (flow_rate ?t - tap) - float
                (walking_speed ?a - agent) - float
                (distance ?s - location ?d - location) - float
                (amount_of_water ?l - location) - fluent
                (distance_to_walk ?a - agent ?d - location) - fluent
                (level ?b - bucket) - fluent
)

;; ================================================================
;; Filling buckets with taps                                     ;;
;; ================================================================

(:action        turnOnTap
 :parameters    (?a - agent ?t - tap ?b - bucket ?l - location)
 :precondition  (and    (at ?a ?l)
                        (at ?b ?l)
                        (at ?t ?l)
                        (not (on ?t)))
 :effect        (and    (on ?t)
                        (filling ?t ?b))
)

(:action        turnOffTap
 :parameters    (?a - agent ?t - tap ?b - bucket ?l - location)
 :precondition  (and    (at ?a ?l)
                        (at ?t ?l)
                        (on ?t)
                        (filling ?t ?b))
 :effect        (and    (not (on ?t))
                        (not (filling ?t ?b)))
)

(:process       fillingBucket
 :parameters    (?b - bucket ?t - tap ?l - location)
 :precondition  (and   (at ?b ?l)
                       (at ?t ?l)
                       (filling ?t ?l)
                       (<= (level ?b) (capacity ?b)))
 :effect        (increase (level ?b) (* #t (flow_rate ?t)))
)
```

```
;; ================================================================
;; Moving buckets between locations                               ;;
;; ================================================================

(:action        pickUp
 :parameters    (?a - agent ?b - bucket ?l - location)
 :precondition  (and    (at ?a ?l)
                        (at ?b ?l))
 :effect        (and    (not (at ?b ?l))
                        (carrying ?a ?b))
)

(:action        putDown
 :parameters    (?a - agent ?b - bucket ?l - location)
 :precondition  (and    (at ?a ?l)
                        (carrying ?a ?b))
 :effect        (and    (at ?b ?l)
                        (not (carrying ?a ?b)))
)



(:action        go
 :parameters    (?a - agent ?s - location ?d - location)
 :precondition  (and    (at ?a ?s)
                        (or (connected ?s ?d) (connected ?d ?s))
                        (not (is_walking ?a ?d)))
 :effect        (and    (not (at ?a ?s))
                        (is_walking ?a ?d)
                        (assign (distance_to_walk ?a ?d)
                                (distance ?d ?s)))
)

(:process       walking
 :parameters    (?a - agent ?d - location)
 :precondition  (and    (is_walking ?a ?d)
                        (>= (distance_to_walk ?a ?d) 0)
 :effect        (decrease (distance_to_walk ?a ?d)
                          (* #t (walking_speed ?a)))
)

(:event         arrive
 :parameters    (?a - agent ?d - location)
 :precondition  (and    (is_walking ?a ?d)
                        (<= (distance-to-walk ?a ?d) 0))
 :effect        (and    (not (is_walking ?a ?d))
                        (at ?a ?d))
)
;; ================================================================
;; Filling among buckets in a location                            ;;
;; ================================================================

(:action        pour
```

```
 :parameters
        (?a - agent ?s - bucket ?d - bucket ?q - real ?l - location)
 :precondition  (and    (at ?a ?l)
                        (carrying ?a ?s)
                        (at ?d ?l)
                        (> ?q 0)
                        (<= ?q (level ?s))
                        (<= ?q (- (capacity ?d) (level ?d))))
 :effect        (and    (decrease (level ?s) ?q)
                        (increase (level ?d) ?q))
)

(:action        deliver
 :parameters    (?a - agent ?b - bucket ?l - location ?q - real)
 :precondition  (and    (at ?a ?l)
                        (carrying ?a ?b)
                        (> ?q 0)
                        (<= ?q (level ?b)))
 :effect        (and    (increase (amount_of_water ?l) ?q)
                        (decrease (level ?b) ?q))
)
```

A.2. A bucket problem

```
;; ===================================================================
;;                                                                   ;;
;; The Problem 1.1 in Section~1                                      ;;
;;                                                                   ;;
;; A possible solution:                                              ;;
;;      1. turnOnTap(ERNIE,TAP1,BUCKET1,SL)                          ;;
;;         ==> fillingBucket(BUCKET1,TAP1,SL) on                     ;;
;;      2. turnOffTAP(ERNIE,TAP1,BUCKET1,SL)                         ;;
;;      3. turnOnTap(ERNIE,TAP1,BUCKET2,SL)                          ;;
;;         ==> fillingBucket(BUCKET2,TAP1,SL) on                     ;;
;;      4. pickUp(ERNIE,BUCKET1,SL)                                  ;;
;;      5. go(ERNIE,SL,DL) ==> walking(ERNIE,DL) on                  ;;
;;      6. arrive(ERNIE,DL)                                          ;;
;;      7. deliver(ERNIE,BUCKET1,DL,1)                               ;;
;;      8. go(ERNIE,DL,SL) ==> walking(ERNIE,SL) on                  ;;
;;      9. arrive(ERNIE,SL)                                          ;;
;;      10. turnOffTAP(ERNIE,TAP1,BUCKET2,SL)                        ;;
;;      11. pickUp(ERNIE,BUCKET2,SL)                                 ;;
;;      12. go(ERNIE,SL,DL) ==> walking(ERNIE,DL) on                 ;;
;;      13. arrive(ERNIE,DL)                                         ;;
;;      14. deliver(ERNIE,BUCKET2,DL,4)                              ;;
;; ===================================================================

(define         (problem problem1.1)
(:domain        Buckets)
(:requirements  :time :continuous-effects)
(:objects       SL - location   DL - location
                TAP1 - tap
                BUCKET1 - bucket   BUCKET2 - bucket
```

```
                ERNIE - agent
)
(:init          (at ERNIE SL)
                (at BUCKET1 SL)
                (at BUCKET2 SL)
                (at TAP1 SL)
                (= (flow_rate TAP1) 0.1)
                (= (walking_speed ERNIE) 5)
                (= (capacity BUCKET1) 4)
                (= (capacity BUCKET2) 4)
                (= (distance SL DL) 100)
                (= (distance DL SL) 100)
                (= (amount_of_water SL) 0)
                (= (amount_of_water DL) 0)
                (= (distance_to_walk ERNIE SL) 0)
                (= (distance_to_walk ERNIE DL) 0)
                (= (level BUCKET1) 0)
                (= (level BUCKET2) 0)
                (connected SL DL)
                (connected DL SL)
)
(:goal          (and    (>= (amount_of_water DL) 5))
                        (<= ?total-time  70))
)
)
```

Appendix B. Proof of soundness and completeness

In this appendix we present the proof that TM-LPSAT is sound and complete over a substantial subset of our extended version of PDDL+ Level 5. We do not carry this analysis to the point of a full formal semantics, in the sense of a fully specified mapping from the symbolic form of the PDDL+ description to the ontological model; rather, we rely on an informal reading of the PDDL+ description. We are confident that the aspects of the formal semantics not dealt with here involve only issues that are well established in the theory of formal semantics of representations and that are essentially orthogonal to the issues that we will deal with here. Our focus here is on defining how a valuation over discrete and numeric variables characterizes behavior over real-valued time and on establishing that the formal constraints generated by TM-LPSAT correspond to the meaning of the PDDL+ representation.

The subset of PDDL+ Level 5 that we deal with here includes atomic actions, events, discrete and numeric fluents, and processes. It does not include interval-valued fluents or resources, but we are confident that extending the proof to cover these is both straightforward and uninteresting. For the remainder of this appendix, we will use "PDDL+" to mean that subset of PDDL+ that we are dealing with here.

To simplify the exposition, in most of this section we will ignore the issue of actions with numerical parameters; these make the definitions more complex but do not present any substantive difficulty. At the end, we will sketch how these can be incorporated.

Formulating and proving these theorems involves the following steps:

1. Defining the ontology of the microworld in which PDDL+ plans are executed.
2. Defining the semantics of a PDDL+ problem statement in terms of this ontology.
3. Defining the relation between a valuation over the variables used in TM-LPSAT and microworld entities.
4. Identifying the rare circumstances in which the physical projection of a system of processes is underdetermined.
5. Formalizing and proving the sense in which TM-LPSAT is "sound" and "complete". (In particular, there are several different senses in which a planner can be "complete"; only one of these applies to TM-LPSAT.)

B.1. Ontology

We assume that there are disjoint finite sets of *actions*, *events*, *processes*, and *fluents*. Each fluent F has associated with it a set of *possible values*, denoted vals(F). If F is *discrete* then vals(F) is a finite set of non-numeric values. If F is *numeric*, then vals(F) is the set of real numbers.

Definition B.1. A *situation* is a four-tuple $\langle A, E, P, M\rangle$ where A is a set of actions; E is a set of events; P is a set of processes, and M is a mapping over the set of fluents, such that, for any fluent F, $M(F) \in \text{vals}(F)$.

We will use a Pascal-style dot notation to denote the fields of a tuple; for example, if S is a situation, then S.A is the set of actions in S, S.P is the set of processes, and so on.

To allow the possibility of multiple time points with the same clock time, as discussed in Section 4, we use the following non-standard temporal model:

Definition B.2. A *time point* is a pair $\langle X, N\rangle$ where X is a real number and N is a positive integer.

Intuitively, $\langle X, N\rangle$ is the Nth time point (counting from 0) at clock time X. Time points are ordered lexicographically; that is, $\langle X1, N1\rangle < \langle X2, N2\rangle$ iff $X1 < X2$ or $[X1 = X2$ and $N1 < N2]$.

Definition B.3. A *time interval* I is a non-empty set of time points such that, if $T1 \in I$, $T2 \in I$ and $T1 < T < T2$ then $T \in I$. If $T1$ and $T2$ are time points with $T1 < T2$ then the *closed interval* $[T1, T2]$ is, as usual, the set of all time points T such $T1 \leqslant T \leqslant T2$.

Definition B.4. For any time point $T = \langle X, N\rangle$, if $N > 0$ then the time point *preceding* T is the point $\langle X, N - 1\rangle$. If $N = 0$, then there is no time point preceding T.

Definition B.5. Let $I = [\langle X1, 0\rangle, \langle X2, N\rangle]$ be a closed time interval. A *history* H over I is a mapping from I to situations.

The following abbreviations will be convenient. Let H be a history over the time interval $I = [\langle X1, 0\rangle, \langle X2, N\rangle]$. We will write $I = \text{dom}(H)$ (read "the domain of H"), and the

real interval $[X1, X2] = \mathrm{dom}_X(H)$ (read "the X-domain of H"). For any time point $T \in \mathrm{dom}(H)$ and fluent F we will write $H(T, F)$ as an abbreviation for $[H(T).\mathrm{M}](F)$, and we will write $\Phi_{H,F}(X)$ for the function over $\mathrm{dom}_X(H)$ defined by $\Phi_{H,F}(X) = H(\langle X, 0\rangle, F)$. Note that $H(T).\mathrm{A}$, $H(T).\mathrm{E}$, and $H(T).\mathrm{P}$ are respectively the set of actions, events, and processes active in H at time point T.

Definition B.6. A history H over $[T0, T1]$ is *compact* if the following holds: For any TA, TB, if $TA \neq T0$, $TA.\mathrm{X} = TB.\mathrm{X}$, $TA.\mathrm{N} < TB.\mathrm{N}$ and $H(TA).\mathrm{A} = H(TA).\mathrm{E} = \emptyset$ then $H(TB).\mathrm{A} = H(TB).\mathrm{E} = \emptyset$.

That is, looking at a sequence of time points $\langle X, 0\rangle$, $\langle X, 1\rangle$, ... all of the time points when an action or event happens are "compacted" together at the beginning of the sequence, with the exception of the starting time point of this history.

Definition B.7. Let H be a history and let $X \in \mathrm{dom}_X(H)$. We say that *the processes in H are constant around X* if there exists a neighborhood (XA, XB) of X over which the active processes do not change. That is, for every TC, TD if $XA < TC.\mathrm{X} < XB$ and $XA < TD.\mathrm{X} < XB$, then $H(TC).\mathrm{P} = H(TD).\mathrm{P}$. If the processes in H are not constant around X, then X is a *time of process change* in H.

Definition B.8. Let H be a history and let T be a time point in $\mathrm{dom}(H)$. T is a *significant* time point in H if either

- T is the starting time point of H.
- T is the ending time point of H.
- $H(T).\mathrm{A} \neq \emptyset$.
- $H(T).\mathrm{E} \neq \emptyset$.
- $T.\mathrm{X}$ is a time of process change and $T.\mathrm{N} = 0$.

Definition B.9. A history H has *finite complexity* if it has finitely many significant time points. H is *monotonous* over real interval (XA, XB) if there is no significant time point T such that $T.\mathrm{X} \in (XA, XB)$.

In all that follows, we will write "history" to mean "compact history of finite complexity".

Definition B.10. Let H be a history, T a time point in the domain of H, and F a fluent. Assume that T is not starting time point of H. Value V is the *value of F before T* if the following conditions are satisfied.

- If $T.\mathrm{N} > 0$, then $V = H(T1, F)$ where $T1$ is the point preceding T.
- If $T.\mathrm{N} = 0$ and F is discrete, then there exists a $T0 < T$ such that for all $T1$, if $T0 < T1 < T$ then $V = H(T1, F)$.
- If $T.\mathrm{N} = 0$ and F is numeric, then V is the limit of $\Phi_{H,F}(X)$ as X approaches $T.\mathrm{X}$ from below.

If α is a term computed over fluents, then the value of α before T is α computed over the values of the fluents before T. If α is a Boolean expression and the value of α before T is TRUE, in the above sense, then we say that α holds before T.

Definition B.11. A *plan* P is a mapping over a bounded time interval I, such that,

- For any $T \in I$, $P(T)$ is a finite set of actions;
- $P(T) = \emptyset$ for all but finitely many T.

We write $I = \text{dom}(P)$, the domain of P.

Intuitively, $P(T)$ is the set of actions that the plan says should be executed at time T.

B.2. Semantics of PDDL+

Definition B.12. A *PDDL+ planning problem* is a triple $\langle D, S, G\rangle$ where D is a PDDL+ domain representation, S is a PDDL+ representation of a starting situation, and G is a PDDL+ representation of a goal.

In Section 6 we did not describe how PDDL+ representations of a starting situation and of a goal are compiled into axioms (a) because it is obvious; (b) because it is the same as in any SAT-based planner. We will similarly not discuss the issue here. We do, however, assume that a PDDL+ representation of a starting situation *uniquely* determines the situation. If S is a PDDL+ representation of a starting situation, we will write "Sit(S)" to denote the actual situation. As for goals, for our purposes here a "goal" can be essentially any property of a history; we assume that the translation of a PDDL+ goal into the corresponding property is done correctly.

The following long definition contains the details of the meaning of the constructs of PDDL+ in terms of the properties of a history. As is common in this kind of semantic definition, the left-hand side of the definition is an almost tautological rewording of the right-hand side. Likewise notable is the strong resemblance of the definition here to the description of the constraint compiler in Section 6. This resemblance (a) is to be expected; (b) means that large parts of the proof of correctness are trivial; (c) limits substantially the degree to which the exhibition of a soundness and correctness proof of this kind actually increases the reader's confidence in the compiler or augments her understanding of it.

Definition B.13. Let H be a history and let D be a PDDL+ domain representation. *H conforms to D* if the following conditions hold:

1. If action $Z \in H(T).A$ or event $Z \in H(T).E$ and D specifies that Z assigns term τ to fluent F, then $H(T, F)$ is equal to the value of τ before T.
2. If F is a numeric fluent and if there is no action $Z \in H(T).A$ nor event $Z \in H(T).E$ such that D specifies that Z assigns a value to fluent F, then $H(T, F)$ is equal to the

value of F before T plus the sum over (all actions/events $Z \in [H(T).\mathrm{A} \cup H(T).\mathrm{E}]$) of the increase/decrease that Z causes in F.

3. (Frame property) Let F be a discrete fluent and let $T1, T2$ be time points such that $T1 < T2$. Then $H(T2, F) = H(T1, F)$, unless there exists a time T such that $T1 < T \leqslant T2$ and either an action $Z \in H(T).A$ or an event $Z \in H(T).E$ such that D specifies that Z assigns a value to fluent F.
4. If action $Z \in H(T).\mathrm{A}$, then the precondition for Z holds before T in H (in the sense of Definition B.10).
5. Event E is in $H(T).\mathrm{E}$ if and only if the precondition of E holds before T in H.
6. Process P is in $H(T).\mathrm{P}$ if and only if the precondition of P holds before T in H.
7. Suppose that H is monotonous over real interval $(X1, X2)$. Let F be a numeric fluent. Then:
   7.1 The function $\Phi_{H,F}$ is continuous and differentiable throughout $(X1, X2)$.
   7.2 For any $x \in (X1, X2)$ the derivative $\frac{\mathrm{d}}{\mathrm{d}X}(\Phi_{H,F}(X))$ at time x is equal to the sum over all processes P in $H(\langle x, 0\rangle).\mathrm{P}$ of the influence of P on F at time $\langle x, 0\rangle$.
   7.3 For all sufficiently large N, $H(\langle X1, N\rangle, F)$ is equal to the limit of $\Phi_{H,F}(x)$ as x approaches X1 from above.

Definition B.14. Let P be a plan; let H be a history; let S be a PDDL+ representation of a starting situation; and let D be a PDDL+ domain description. H is a *projection* of plan P starting in S and following D if the following conditions hold:

- $\mathrm{dom}(P) = \mathrm{dom}(H)$;
- For all $T \in \mathrm{dom}(H)$, $H(T).\mathrm{A} = P(T).\mathrm{A}$;
- $\mathrm{Sit}(S)$ is the starting situation of H; and
- H conforms to D.

As we shall see in Lemma B.1, for any P, S, D there exists at most one such projection, with rare exceptions to be discussed below.

Definition B.15. Let $R = \langle D, S, G\rangle$ be a PDDL+ problem. History H is a *historical solution* of R if H starts in $\mathrm{Sit}(S)$, conforms to D, and achieves G. A finite plan P is a *planning solution* of R if every projection of P starting in S and following D is a historical solution of R.

Note that constraint-based planning techniques give correct results only if the only source of uncertainty is the actions to be carried out; once the actions are specified, there is only one possible projection. If there is more than one possible projection, or if there is anything unspecified in the starting situation, then a constraint-based planner will make the most optimistic assumptions about these; that is, it will set these uncontrolled parameters in the same way that it sets the actions to be carried out. Note also that the definition above of the correctness of a plan only works for *complete* plan representations; partial plan representations, such as those returned by TWEAK [11], require a more complex definition of correctness.

B.3. Valuations and their interpretations

Definition B.16. Let R be a PDDL+ problem representation. Let $T_0 \ldots T_k$ be a sequence of $k+1$ distinct symbols, called "time point variables". We define the set "ATOMS$[R, k]$" to be the set of all the following atoms: For each time point variable T_i,

- The atom "c(T_i)".
- For each action, event, or process Z in R, the atom "active(Z)[T_i]".
- For each discrete fluent F in R, the atom "$F[T_i]$".
- For each numeric fluent F in R, the atoms "$F[T_i^-]$" and "$F[T_i^+]$".
- For each numeric fluent F and each action or event Z that potentially changes F incrementally, the atom "$\Delta(F, Z)[T_i]$".
- For each numeric fluent F and each process P that potentially influences F, the atom "$\Gamma(F, P, T_i, T_{i+1})$".

Definition B.17. A *T-valuation* V is an assignment of each atom in ATOMS$[R, k]$ to a value of the appropriate sort, such that, for each $i < j$, V("c(T_i)") $\leqslant$ V("c(T_j)").

Definition B.18. Let V be a T-valuation over ATOMS$[R, k]$. The *time point mapping for V* is the function τ from T_i to time points defined as follows: $\tau(T_i) = \langle X, J \rangle$ where

- $X =$ V("c(T_i)");
- T_i is the Jth time point variable such that $X =$ V("c(T_i)"). That is, V("c(T_i)") = V("c(T_{i-1})") $= \cdots =$ V("c(T_{i-J})") $\neq$ V("c(T_{i-J-1})").

Definition B.19. Let V be a T-valuation over ATOMS$[R, k]$. Let τ be the time point mapping for V. Plan P is *indicated by V* if:

- For each T_i, P($\tau(T_i)$) = the set of all actions A such that V("active(A)[T_i]") = TRUE.
- For each T, if $T \neq \tau(T_i)$ for all i, then $P(T) = \emptyset$.

Definition B.20. Let D be a domain description, let V be a T-valuation, and let H be a history. Let τ be the time point mapping of V. *H corresponds to V* if:

1. dom(H) = $[\tau(T_0), \tau(T_k)]$.
2. If $i < j$ then $\tau(T_i) < \tau(T_j)$.
3. For each T_i, $H(\tau(T_i))$.A = $\{A \mid V(\text{"active}(A)[T_i]\text{"}) = \text{TRUE}\}$.
4. For each T_i, $H(\tau(T_i))$.E = $\{E \mid V(\text{"active}(E)[T_i]\text{"}) = \text{TRUE}\}$.
5. For any discrete fluent F, $H(\tau(T_i), F) = V(\text{"}F[T_i]\text{"})$.
6. For any numeric fluent F, $H(\tau(T_i), F) = V(\text{"}F[T_i^+]\text{"})$.
7. Let T be any time point in dom(H). Let T_i be the greatest time variable such that $\tau(T_i) \leqslant T$.
   7.1. If $T \neq \tau(T_i)$ then $H(T)$.A = $H(T)$.E = $\emptyset$.
   7.2. For any discrete fluent F, $H(T, F) = V(\text{"}F[T_i]\text{"})$.
   7.3. If T.X = $\tau(T_i)$.X, then

7.3.1. For any numeric fluent F, $H(T, F) = V(\text{“}F[T_i^+]\text{”})$.
7.3.2. For any process P, $P \in H(T).\text{P}$ if the preconditions of P, as defined in D, are satisfied before T in H.
7.4. If $T.\text{X} > \tau(T_i).\text{X}$ then
7.4.1. For any numeric fluent F, let

$$q = \big(T.\text{X} - \tau(T_i^+).\text{X}\big)/\big(\tau(T_{i+1}).\text{X} - \tau(T_i).\text{X}\big).$$

Then $H(T, F) = (1 - q)V(\text{“}F[T_i^+]\text{”}) + qV(\text{“}F[T_{i+1}^-]\text{”})$.
(Linear interpolation between significant points.)
7.4.2. $H(T).\text{P} = \{P \mid V(\text{“active}(P)[T_i]\text{”}) = \text{TRUE}\}$.

Note that at a significant time T_i and at all subsequent non-significant times with the same clock time, the activity of a process in H is rather awkwardly defined in terms of the preconditions specified in the domain description D (which is the only reason for including D as a parameter in this definition at all). The reason for this is as follows: Recall that in our discussion of axiom (3.4) we defined “active(P)[T_i]” to mean that P was active over an interval starting in T_i. Therefore, if P comes to an end at T_i, then in V, “active(P)[T_i]” is FALSE. However in H, P will be active at $T = \tau(T_i)$ if its preconditions are satisfied at T_i; thus, P will still be active at T in H if the preconditions are becoming FALSE due to a zero crossing, which will be come negative only after T, but it will be inactive at T in H if the preconditions have just become FALSE in T due to a discrete change. The valuation V by itself does not distinguish these two cases; one needs to know the domain definition. Of course, since the effect of a process is differential, whether or not a process is active at an instant actually makes no difference.

B.4. Indeterminate projections

As discussed above, the constraint-based approach to planning relies on the assumption that any plan has a unique projection; that is, once you fix the actions you are to do, that determines everything else that can happen. Unfortunately, in a theory that include processes of the kind in PDDL+, there are rare cases where that assumption is false.

The problem arises for the following reason: A system of processes in effect imposes a set of ordinary differential equations (ODE's) over the numerical fluents involved. In standard applications of ODE's one can rely on a standard existence and uniqueness result for initial value problems to guarantee that, having set up the starting condition and the differential equation, history can develop in only one way. However, this result only holds when the “driving function” for the ODE is *continuous*. PDDL+ processes define a *discontinuous* driving function, so neither existence nor uniqueness is guaranteed and indeed there are cases where history can develop in more than one way.

Consider the following example: There is one numeric fluent F and two processes $P1$ and $P2$. $P1$ has precondition TRUE and increases F at the rate of 1 unit per second. $P2$ has the precondition $F \leqslant 0$ and decreases F at the rate of 1 unit per second. Suppose that $F = 0$ at time $T = 0$. Then for any $T1 \geqslant 0$, the following is a consistent behavior:

For any time T,

If $0 \leqslant T \leqslant T1$ then $P1$ and $P2$ are active at T and $F = 0$.
If $T1 < T$ then only $P1$ is active at T and $F = T - T1$.

This corresponds to the fact that the differential equation

$$\dot{F} = \begin{cases} 0 & \text{if } F \leqslant 0, \\ 1 & \text{if } F > 0 \end{cases}$$

has infinitely many solutions (if one allows a "solution" to have finitely many points where it is continuous but not differentiable).

However, a history H only encounters this problem at time T if the following condition is met: (this is a necessary condition for the problem, not a sufficient condition).

Indeterminacy condition. There is a dependency cycle between processes, numeric fluents, and preconditions:

$$P_1 \to F_1 \to \Phi_1 \to P_2 \to \cdots \to \Phi_k \to P_1,$$

such that process P_i is active at T and influences fluent F_i; F_i is involved in precondition Φ_i; and Φ_i is a precondition of process P_{i+1} which is satisfied but just on the borderline at time T.

A history H satisfies the Unique Projection Condition (UPC) if it never satisfies the indeterminacy condition.

In principle, it would be possible to add the unique projection condition as a TM-LPSAT constraint. The theoretical advantage would be that doing so would give a slightly improved pair of soundness and completeness results, as discussed below. We have not implemented it, however. In most actual domains where the PDDL+ representation is at all a reasonable approximation, it is possible to determine at compilation time that no circularity can arise among fluents and processes; hence, no UPC axioms will be formulated. If a domain does have this kind of circular dependence among fluents and processes, then it is very unlikely that it can be adequately modeled using constant-rate influences. The theoretical improvement to the algorithm in the rare cases where the UPC axioms do have an effect did not seem worth the programming effort.

B.5. Constraints and theorems

Finally, we introduce a notation for the set of constraints generated by TM-LPSAT, state our soundness and completeness theorems, and prove the theorems.

Definition B.21. Let R be a planning problem and let k be a positive integer. We define TM-LPSAT(k, R) to be the set of constraints constructed from R over ATOMS[R, k] as defined in Sections 6.1, 6.2, 6.3, 6.5, 6.6, and 6.10, using versions (10.7) and (10.11) of the zero crossing axioms, but excluding the mutex axioms in Sections 6.1.3 and 6.2.4. The mutex axioms are useful for enforcing certain regularity conditions that are important in other contexts, and they are part of the standard semantics of PDDL+, but in the context of this proof they just get in the way.

Let Λ be an algorithm (the constraint solver) that achieves the following: Λ takes as input a set of constraints C of the form produced by TM-LPSAT. If the constraints C have a solution, then Λ returns a T-valuation that is a solution of C. If the constraints C do not have a solution, then Λ returns a flag indicating there is no solution.

Theorem B.1 (Soundness). *Let R be a planning problem. If Λ*(TM-LPSAT(k, R)) *returns a* T*-valuation V, and V satisfies the* UPC *then the plan indicated by V is a planning solution to R.*

Theorem B.2 (Completeness). *Let R be a planning problem. If there exists a planning solution to R, then for some value of k, Λ*(TM-LPSAT(k, R)) *returns a* T*-valuation V. By Theorem* B.1*, if V satisfies the* UPC *then V indicates a planning solution to R.*

The above pair of theorems does not close quite tight for the following reason: One can construct a problem R where there is a correct plan $P1$ that satisfies the UPC, but where there is also a plan $P2$ that does not satisfy the UPC, such that some but not all of the projections of $P2$ satisfy the goal. Such a plan $P2$ is not a correct solution of the problem, since, if you execute it, you cannot be sure of accomplishing the goal. In this case, there are two things that Λ(TM-LPSAT(R)) may do:

- It may return a valuation that indicates $P1$. In that case, you can check that $P1$ satisfies the UPC, and you are certain that it is a correct plan.
- It may return a valuation that indicates $P2$. In that case, you can detect that $P2$ violates the UPC. If you accept $P2$, you are accepting an incorrect plan. If you reject $P2$ then you have failed to find a plan, even though there exists a plan that satisfies the UPC.

We can tighten this, in principle, by adding the UPC to the constraints generated by TM-LPSAT. Let TM-LPSAT^U(k, R) be TM-LPSAT(k, R) together with the UPC. Thus, if TM-LPSAT^U is applied to the above problem it will return $P1$ and not $P2$.

Theorem B.3. *Let R be a planning problem.*

- *For any k, if Λ*(TM-LPSAT^U(R, k)) *returns a* T*-valuation V, then V indicates a planning solution to R satisfying the* UPC*.*
- *If there exists a planning solution to R satisfying the* UPC*, then there exists k such that Λ*(TM-LPSAT^U(R, k)) *returns a* T*-valuation V that indicates a planning solution to R satisfying the* UPC*.*

B.6. Proofs

The proofs of the above three theorems follow, in a series of lemmas.

Lemma B.1. *Let D be a* PDDL+ *domain description and let S be a* PDDL+ *representation of a starting situation. Let P be a finite plan. Then there exists at most one history H*

of finite complexity that is a projection of P, starts in S, conforms to D, and satisfies the UPC.

Proof. (By contradiction.) Suppose that there exist two such histories $H1 \neq H2$. Note that, if TS is the starting time point of $\mathrm{dom}(P)$ then $\mathrm{Sit}(S) = H1(TS) = H2(TS)$. Note also that $\mathrm{dom}(P) = \mathrm{dom}(H1) = \mathrm{dom}(H2)$. Let interval I be the maximal initial segment of $\mathrm{dom}(P)$ such that, for all $T \in I$ $H1(T) = H2(T)$. (I can be constructed as the union of all initial segments $I1$ of $\mathrm{dom}(P)$ such that, for all $T \in I1$, $H1(T) = H2(T)$.) There are three cases to be considered:

Case 1: I has the form $[TS, TE]$ for some ending point TE. Let $TE1$ be the time point following TE. Then P determines the actions in $TE1$. Definition B.13 parts 5 and 6 determine the events and processes in $TE1$. Definition B.13 parts 1, 2, and 3 determine the value of all the fluents in $TE1$. Hence $H1(TE1) = H2(TE1)$.

Case 2: I has the form $\{T \mid TS \leqslant T \wedge T.\mathrm{X} < XE\}$ for some upper bound XE. Let $TE1 = \langle XE, 0\rangle$. Then the situation in $TE1$ is determined, by the same argument as in part (1).

Case 3: I has the form $\{T \mid TS \leqslant T \wedge T.\mathrm{X} \leqslant XE\}$ for some upper bound XE. In this case, there is no "next" situation after I, so the arguments in cases 1 and 2 do not apply. Rather, we proceed as follows: Since $H1$ and $H2$ have finite complexity, there exists $XF > XE$ and NE such that no events or actions after $\langle XE, NE\rangle$ and before $\langle XF, 0\rangle$ in either $H1$ or $H2$. That is, no actions or events occur in this interval; and the values of discrete fluents does not change during this interval; and the class of active processes does not change in the interval $\{T \mid \langle XE, NE\rangle \leqslant T < \langle XF, 0\rangle\}$. Let $TE = \langle XE, NE\rangle$. By virtue of the UPC, there is a topological sorting of numerical fluents, processes that are active at TE, and numerical preconditions of processes that are active at TE that are just on the borderline such that every process comes after its preconditions, every fluent comes after the processes that influence it, and every precondition comes after the fluents that it references. Since numerical fluents are continuous in the absence of actions and events, there exists $XG > XE$ such that at all time points T where $XE < T.\mathrm{X} < XG$, all numerical preconditions that were not on the borderline at TE remain with the same truth value as at TE. Going through this topological sorting in order, therefore, we can predict that there is an interval $[TE, TH]$ where $TH.\mathrm{X} > XE$ where the value of a numeric fluent is determined only the value at TE together with processes preceding it in the list; where the truth of a borderline precondition is determined only by numerical fluents preceding it on the list; and where the activities of processes is determined only by preconditions preceding it on the list. Thus, the history is determined for some time after I, contrary to the assumption. □

Lemma B.2. *For any* T*-valuation V and domain description D there exists a unique history H that corresponds to V relative to D.*

Proof. Definition B.20 gives an explicit, fully determined, construction of H from V and D. □

In all the following definitions, let R be a planning problem; let D be the domain description of R; let V be a T-valuation; let H be a history; and let τ be the time point mapping of V.

Definition B.22. A valuation V *covers* time point T if there exists a time point variable T_i such that $T = \tau(T_i)$.

Lemma B.3. *If H corresponds to V relative to D and either some action or event A occurs in T at H, then V covers T.*

Proof. Contrapositive of part 7.1 in Definition B.20. □

Definition B.23. Let T be a time point in dom(H).

- T is *significant*-1 if there is a precondition Φ of an event that is TRUE in T.
- T is *significant*-2 if there is a precondition Φ of a process such that Φ is FALSE in H before T and Φ is TRUE in $H(T)$.
- T is *significant*-3 if there is a precondition Φ of a process for which one of the following holds:
  - Φ is TRUE in H before T and FALSE in $H(T)$.
  - Φ is TRUE in $H(T)$, $T.\mathrm{N} = 0$, and there exists $X2 > T.\mathrm{X}$ such that, for all $X1$, $N1$, if $T.\mathrm{X} < X1 < X2$ then Φ is FALSE in $H(\langle X1, N\rangle)$.

Lemma B.4. *Let I be an initial segment of* dom(H). *Let T_i be a time variable such that $\tau(T_i) \in I$ and such that $\tau(T_i).\mathrm{X} < \tau(T_{i+1}).\mathrm{X}$. Let time interval $I1 = \{T \in I \mid \tau(T_i) < T < \tau(T_{i+1})\}$. Let $T1$ be any time point in $I1$. Let $T2$ be a time point in $I1$ such that $T2.\mathrm{X} > \tau(T_i).\mathrm{X}$. If V satisfies* TM-LPSAT(k, R) *and H corresponds to V then*

A. $\tau(T_{i+1}).\mathrm{N} = 0$.
B. *If* $\tau(T_i).\mathrm{X} < T.\mathrm{X} < \tau(T_{i+1}).\mathrm{X}$ *then P is active in T if and only if* $V(\text{“active}(P)[T_i]\text{”}) = \mathrm{TRUE}$.
C. *H is monotonous over the real interval* $(\tau(T_i).\mathrm{X}, T2.\mathrm{X})$.
D. *If F is a discrete fluent then* $H(T1, F) = H(\tau(T_i), F)$.
E. *Let* $PP = H(T2).\mathrm{P}$. *Let F be a numeric fluent. Let Γ be the sum over $P \in PP$ of the influence of P on F. Then* $H(T1, F) = H(\tau(T_i), F) + \Gamma \cdot (T1.\mathrm{X} - \tau(T_i).\mathrm{X})$.
F. *For any numerical fluent F, the value of F before T_{i+1} is equal to* $V(\text{“}F[T_{i+1}^{-}]\text{”})$.

Proof.

A. Immediate by Definition B.18 of a time point mapping.
B. From Definition B.20 part 7.4.2, for all such T, the active processes are those such that $V(\text{“active}(P)[T_i]\text{”}) = \mathrm{TRUE}$.
C. By Lemma B.3, no actions or events can occur in H at uncovered points. By part B, the set of processes remains constant; hence there are no times of process change. Hence ($T2.\mathrm{X}$,$T3.\mathrm{X}$) is monotonous.

D. Immediate from Definition B.20 part 7.2.

E. There are two cases to consider:

E.1. $T1.\mathrm{X} = \tau(T_i).\mathrm{X}$. In this case, the result is immediate from Definition B.20 part 7.3.1.

E.2. $T1.\mathrm{X} > \tau(T_i).\mathrm{X}$. By axioms (3.1), (3.2), and (3.3) the difference $V(\text{“}F[T_{i+1}^-]\text{”}) - V(\text{“}F[T_i^+]\text{”})$ is equal to [the sum over [all processes P such that $V(\text{“active}(P)[T_i]\text{”}) = \mathrm{TRUE}$] of the influence of P on F] times $[V(\text{“c}(T_{i+1})\text{”}) - V(\text{“c}(T_i)\text{”})]$. By part B this set is PP and this sum is Γ. By Definition B.20 part 2, $\mathrm{V}(\text{“c}(T_i)\text{”}) = \tau(T_i).\mathrm{X}$, so we have

$$V\big(\text{“}F[T_{i+1}^-]\text{”}\big) - V\big(\text{“}F[T_i^+]\text{”}\big) = \Gamma \cdot \big(\tau(T_{i+1}).\mathrm{X} - \tau(T_i).\mathrm{X}\big).$$

By Definition B.20 part 7.4.1,

$$H(T, F) = (1-q)V\big(\text{“}F[T_i^+]\text{”}\big) + qV\big(\text{“}F[T_{i+1}^-]\text{”}\big), \quad \text{where}$$
$$q = \big(T.\mathrm{X} - \tau(T_i).\mathrm{X}\big)/\big(\tau(T_{i+1}).\mathrm{X} - \tau(T_i).\mathrm{X}\big).$$

By Definition B.20 part 7.3.1, $H(\tau(T_i), F) = V(\text{“}F[T_i^+]\text{”})$. Combining the above with some algebraic manipulation gives the desired result.

F. Again there are two cases:

F.1. $\tau(T_{i+1}).\mathrm{N} > 0$. In that case the value of F before $\tau(T_{i+1})$ is the value $H(T1, F)$ where $T1$ is the situation that precedes $\tau(T_{i+1})$. Since history H is compact, some action or event must occur in $T1$. Hence, by Lemma B.3, $T1 = \tau(T_i)$. By Definition B.20 part 6, $H(T1, F) = V(\text{“}F[T_i^+]\text{”})$. By axioms (3.1), (3.2), and (3.3), $V(\text{“}F[T_i^+]\text{”}) = V(\text{“}F[T_{i+1}^-]\text{”})$.

F.2. $\tau(T_{i+1}).\mathrm{N} = 0$. In that case the value of F before $\tau(T_i)$ is the limit of $\Phi_{H,F}(X)$ as X approaches $\tau(T_i).\mathrm{X}$ from below.
By part (E), $\Phi_{H,F} = H(\tau(T_i), F) + \Gamma \cdot (\tau(T_{i+1}).\mathrm{X} - \tau(T_i).\mathrm{X})$.
Using (E) and axioms (3.1), (3.2), and (3.3), this is equal to $V(\text{“}F[T_{i+1}^-]\text{”})$. □

Lemma B.5. *If H corresponds to V, then there can be at most finitely many points that are significant-1, significant-2, or significant-3.*

Proof. By Definition B.20, between any two points $\tau(T_i)$ and $\tau(T_{i+1})$ every discrete fluent is constant and every numeric fluent is a linear function of time. Any numeric preconditions are a linear inequality over the numeric fluents, and thus a linear inequality in time over this time interval. A time point that is significant-1, -2, or -3 must either involve a change to a discrete fluent, which do not occur at uncovered points, or a zero crossing of a numeric constraint, which can occur at most once for each such constraint between $\tau(T_i)$ and $\tau(T_{i+1})$. □

Lemma B.6. *If H corresponds to V and V satisfies axiom* (10.6)*, then V covers any time point that is significant-1 or significant-2.*

Proof. (By contradiction.) (This proof is essentially the same as the discussion in Section 6.10, but set in a specific formal context.) Let T be a time point that is significant-1

or significant-2 but not covered. Let T_i be the greatest time point variable for which $\tau(T_i) < T$; thus $T < \tau(T_{i+1})$. Let $T1 = \tau(T_i)$ and $T2 = \tau(T_{i+1})$. Let the precondition of the event or process involved be put into DNF: $\Theta_1 \vee \cdots \vee \Theta_z$. Then all of the Θ_i are FALSE before T and at least one is TRUE at T. Let $\Theta(T)$ be a constraint that becomes TRUE at T. Let TF be a time such that $T1 \leqslant TF < T$ and $\Theta(TF)$ is FALSE. $\Theta(T)$ has the form $\bigwedge_p F_p(T) \wedge \bigwedge_p Q_p(T) \geqslant 0$. where $F(p)$ are discrete constraints. By Definition B.20, $F_p(T) = F_p(T1)$; since $\Theta(T)$ is TRUE, $\bigwedge_p F_p(T1)$ must be TRUE. Thus we must have $\bigwedge_p Q_p(T) \geqslant 0$ but not $\bigwedge_p Q_p(TF) \geqslant 0$. Since each Q_p is a linear function of time, if $Q_p(T) \geqslant 0$ but $Q_p(TF) < 0$ then $Q_p(T1) < 0$. Also, since each Q_p is a linear function of time if $Q_p(T) \geqslant 0$ then either $Q_p(T1) > 0$ or $Q_p(T2^-) > 0$ or $Q_p(T1) = Q_p(T2^-) = 0$. where $Q_p(T2^-)$ is the limit of $Q_p(T)$ as T approaches $T2$ from below. But $Q_p(T1) = V(\text{“}Q_p(T_i^+)\text{”})$ and $Q_p(T2^-) = V(\text{“}Q_p(T_{i+1}^-)\text{”})$; so this possibility is excluded by axiom (10.6). □

Let us define an additional axiom for zero crossings from TRUE to FALSE analogous to axiom (10.6):

$$(10.12)\quad \left[\bigwedge_p F_p[T_i] \wedge \bigwedge_p [Q_p[T_{i+1}^-] > 0 \vee Q_p[T_i^+] > 0 \vee Q_p[T_i^+] = Q_p[T_{i+1}^-] = 0]\right] \Rightarrow \left[\bigwedge_p Q_p[T_{i+1}^-] \geqslant 0\right].$$

As with axiom (10.11), we constructed this axiom by starting with axiom (10.6) and interchanging F[T_i^+] and F[T_{i+1}^-].

Lemma B.7. *If H corresponds to V and V satisfies axiom* (10.12)*, then V covers any time point that is significant*-3.

Proof. Exactly analogous to the proof of Lemma B.6, with the following changes: Since Θ changes from TRUE to FALSE, choose TF such that $T < TF \leqslant T_{i+1}$ and $\Theta(TF)$ is FALSE. Since Q_p is a linear function of time, $Q_p(T) \geqslant 0$ and $Q_p(TF) < 0$, it follows that $Q_p(T2) < 0$. □

Lemma B.8. *Suppose that V satisfies* TM-LPSAT(k, R) *using axiom* (10.6) *and H corresponds to V. If the preconditions of event E are satisfied before T in H, then E occurs in H at time T.*

Proof. (By contradiction.) Suppose that there is a time T when precondition Θ of event E holds before T but event E is not active. Since E is not active, T is not a significant time point. We have the following case analysis:

- $T.\mathrm{N} > 0$. In this case, Θ holds in $T1$ where $T1$ is the situation preceding T. This would violate axioms (2.1) and (2.2).

- $T.\mathrm{N} = 0$, and $T = \tau(T_i)$ is a covered time point. By Lemma B.4, the value of β before T is equal to $V(\text{“}\beta(T_i^-)\text{”})$. Thus, this violates axiom (2.1).
- $T.\mathrm{N} = 0$ and T is not a covered time point. Let $T1$ be the greatest covered time point such that $T1 < T$. Let TS be the smallest value such that $TS \geqslant T1$ and such that $\Theta(TX)$ holds for all $TX \in [TS, T]$. (Since H has finite complexity and since numerical preconditions are non-strict inequalities, it is easily shown that such a smallest value exists.) If $TS = T1$ then TS is covered. If $\Theta(T)$ comes to be TRUE in TS as a result of a discrete change or of a discontinuous change to a numeric variable, then TS must be covered. If $\Theta(T)$ comes to be TRUE in TS as a result of a continuous change in a numeric variable, then TS is significant-1 in H and hence covered. Therefore, since TS is covered, by axioms (2.1) and (2.2), E occurs in the successor to TS; hence the successor to TS must be covered. But this contradicts the definition of $T1$. □

Lemma B.9. *Suppose that V satisfies* TM-LPSAT(k, R) *using axioms* (10.6) *and* (10.12) *and H corresponds to V. P is active in H at time T if and only if the preconditions of process P are satisfied before T in H.*

Proof. Let T_i be the maximum time point such that $\tau(T_i) \leqslant T$. If $\tau(T_i).\mathrm{X} = T.\mathrm{X}$, then the result is immediate from Definition B.20 part 7.3.2. Otherwise, note that between $\tau(T_i)$ and $\tau(T_{i+1})$ there are no actions or events or time points that are significant-2 or significant-3. Since numerical precondition are all non-strict inequalities, we have the following possible cases:

- Some precondition β of P is satisfied for all $T1$ such that $\tau(T_i) \leqslant T1 < \tau(T_{i+1})$. Then $\beta[T_i^+]$ is TRUE and, by Lemma B.4 part F, $\beta[T_{i+1}^-]$ is TRUE. Hence by axiom (3.4), $V(\text{“active}(P)[T_i]\text{”}) = \text{TRUE}$. Hence by Definition B.20 P is active in $H(T)$.
- Some precondition β of P is satisfied in $\tau(T_i)$ but depends on a numerical constraint that is on the borderline and just about to become FALSE; and no precondition of P is satisfied for any $T1$ such that $\tau(T_i).\mathrm{X} < T1.\mathrm{X} < \tau(T_{i+1}).\mathrm{X}$. In this case, $\beta[T_{i+1}^-]$ is FALSE, so by axiom (3.4), $V(\text{“active}(P)[T_i]\text{”}) = \text{FALSE}$. Hence by Definition B.20 P is not active in $H(T)$.
- No precondition β of P is satisfied in $\tau(T_i)$. In this case, no precondition β can be satisfied for any $T1$ such that $\tau(T_i) \leqslant T1 < \tau(T_{i+1})$. Therefore $\beta[T_i^+]$ is FALSE, so by axiom (3.4) $V(\text{“active}(P)[T_i]\text{”}) = \text{FALSE}$. Hence by Definition B.20 P is not active in $H(T)$. □

Lemma B.10. *If V satisfies* TM-LPSAT(k, R) *using axioms* (10.6) *and* (10.12) *and H corresponds to V, then H conforms to the domain description D of R.*

Proof. If an action or event Z occurs in H at time T, then by Definition B.20 there is a T_i in V such that $\tau(T_i) = T$ and such that V(“active(Z)[T_i]”). By axioms (1.1)–(1.5) for actions and the corresponding axioms for events (Section 6.2.1), the effects of these actions will be reflected in the value of the fluents in T_i in a way that exactly matches

constraints 1, 2, and 3 in Definition B.13. Thus Definition B.13 constraints 1, 2, and 3 are always satisfied.

By axioms (1.6) and (2.1), if V("active(Z)[T_i]") then $\beta[T^-]$ must hold. By Lemma B.4 part F, this is equivalent to the condition that β must hold before T_i. So constraint 4 and the left to right implication of constraint 5 of Definition B.13 must hold.

The right to left implication of constraint 5 is Lemma B.6. Constraint 6 is Lemma B.7. Constraint 7 follows immediately from Lemma B.4. □

Corollary B.11. *Lemma* B.10 *continues to hold if axioms* (10.6) *and* (10.12) *are replaced, either by axioms* (10.3) *and* (10.4) *or by axioms* (10.7) *and* (10.11)*.*

Proof. As discussed in Section 6.10, these new axioms are stronger than (10.6) and (10.12), so a T-valuation that satisfies the conditions of Corollary B.11 *a fortiori* also satisfies the conditions of Lemma B.10. □

Lemma B.12. *Let R be a planning problem; let V be a* T*-valuation that satisfies* TM-LPSAT(k, R) *and let H be a history that corresponds to V. Then H is a historical solution of R.*

Proof. Immediate from Corollary B.11 and Definition B.15. □

Let Λ be a constraint solver, with the properties defined on p. 243.

Theorem B.1 (Soundness)*. Let R be a planning problem. If* Λ(TM-LPSAT(k, R)) *returns a* T*-valuation V, and V satisfies the* UPC *then the plan P indicated by V is a planning solution to R.*

Proof. By Lemma B.2, there exists a unique history H corresponding to V relative to the domain description of R. By Corollary B.11, H is a historical solution of R. By Definition B.15, P is a planning solution to R. □

Definition B.24. Let H be a history that conforms to domain description D. A real value XZ is a *zero crossing* of H with respect to D if there is some numerical precondition $\beta \geqslant 0$ of either an event or a process in D such that either

- There exists $X1 < XZ$ such that
  [if $X1 < X < XZ$ then $\Phi_{H,\beta}(X) < 0$] and $\lim_{X \to XZ^-} \Phi_{H,\beta}(X) = 0$; or
- There exists $X1 > XZ$ such that
  [if $XZ < X < X1$ then $\Phi_{H,\beta}(X) < 0$] and $\lim_{X \to XZ^+} \Phi_{H,\beta}(X) = 0$.

Lemma B.13. *If history H conforms to domain description D, then H has only finitely many zero crossings relative to D.*

Proof. Using Definition B.20, the fact that H has finite complexity, and the fact that every precondition β is a linear function of the numeric fluents, it follows that every such β is piecewise linear. □

Definition B.25. Let H be a history that conforms to D. A time point Y is *significant*-4 if either Y is significant or $Y = \langle X, 0\rangle$ where X is a zero crossing.

Definition B.26. Let H be a history that conforms to domain description D. The *trace* of H relative to D is the T-valuation V constructed as follows: Let $Y_0 \ldots Y_k$ be all the significant-4 time points of H in sequential order. Let $T_0 \ldots T_k$ be time point variables. Then:

- $V(\text{“c}(T_i)\text{”}) = Y_i.\text{X}$.
- For any action A, $V(\text{“active}(A)[T_i]\text{”}) = \text{TRUE}$ iff $A \in H(T).\text{A}$.
- For any event E, $V(\text{“active}(E)[T_i]\text{”}) = \text{TRUE}$ iff $E \in H(T).\text{E}$.
- For any process P, $V(\text{“active}(P)[T_i]\text{”}) = \text{TRUE}$ iff $P \in H(T).\text{P}$.
- For any discrete fluent F, $V(\text{“}F[T_i]\text{”}) = H(T, F)$.
- For any numeric fluent F, $V(\text{“}F[T_i^+]\text{”}) = H(T, F)$.
- For any numeric fluent F, $V(\text{“}F[T_i^-]\text{”}) =$ the value of F before T in H.
- For each numeric fluent F and each action or event Z that potentially changes F incrementally, $\text{V}(\text{“}\Delta(F, Z)[T_i]\text{”})$ is the change that Z makes to F at time Y_i.
- For each numeric fluent F and each process P that potentially influences F, the atom “$\Gamma(F, Z, T_i, T_{i+1})$” is the change that P makes to F between Y_i and Y_{i+1}.

Lemma B.14. *Let R be a planning problem. Let H be a historical solution to R. Let V be the trace of H. Then V satisfies* TM-LPSAT(k, R) *and H corresponds to V.*

Proof. We must establish that if H and the domain description in R satisfy the conditions of Definition B.13 for “conform” and if V satisfies the conditions of Definition B.25 for “trace” then V satisfies each of the axioms of TM-LPSAT(k, R) and H and V satisfy each of the conditions in Definition B.20 for “correspond”. However, this is all a straightforward repetition of the argumentation that we have given above in Section 6 of the paper and in the proof of Theorem B.1. □

Note that the trace V of H will satisfy the strongest version (10.3) and (10.4) of the zero crossing axioms, and hence satisfy the weaker forms (10.6), (10.7), (10.11) and (10.12).

Theorem B.2 (Completeness). *Let Λ be a constraint solver, as in Theorem* B.1 *above. Let R be a planning problem. If there exists a planning solution to R, then for some value of k, Λ*(TM-LPSAT(k, R)) *returns a* T*-valuation V. By Theorem* B.1*, if V satisfies the* UPC *then V indicates a planning solution to R.*

Proof. Let $P1$ be a planning solution to R. Let $H1$ be a projection of $P1$ relative to R. Let k be the number of significant-4 time points in R. Let $V1$ be the trace of $H1$. By Lemma B.14, $V1$ satisfies TM-LPSAT(k, R), so by definition of Λ, Λ(TM-LPSAT(k, R)) returns some valuation V that satisfies TM-LPSAT(k, R). By Theorem B.1, V indicates a planning solution to R. □

Theorem B.3. *Let R be a planning problem.*

- *For any k, if Λ*(TM-LPSAT$^U(R,k)$) *returns a* T*-valuation V, then V indicates a planning solution to R satisfying the* UPC.
- *If there exists a planning solution to R satisfying the* UPC*, then there exists k such that Λ*(TM-LPSAT$^U(R,k)$) *returns a* T*-valuation V that indicates a planning solution to R satisfying the* UPC.

Proof. Immediate from Theorems B.1 and B.2. □

B.7. Actions with numerical parameters

Finally, let us sketch how actions with numerical parameters can be fit into this framework. Define a *symbolic action* to be an atom; intuitively, this corresponds to the action functor and its non-numeric parameters. Define an *action* to be a tuple $\langle SA, P_1 \ldots P_k \rangle$ where S is the symbolic action and $P_1 \ldots P_k$ are values of the numerical parameters. Thus, in the bucket domain, the tuple ⟨POUR(A1,B1,B2,L3), 5⟩ would correspond to the action "(pour a1 b1 b2 5 l3)". Modify the definitions in this appendix as follows:

- In Definition B.1, add the constraint that in any situation S there cannot be two different actions with the same symbolic action.
- In Definition B.16, delete the word "actions" from the specification of the second category of atoms.
- In Definition B.16, add the following two categories of atoms:
  - For each symbolic action *SA*, the atom "active(*SA*)[T_i]".
  - For every symbolic action *SA*, for each numerical parameter P, the atom "$SA_P[T_i]$".
- Add the following definition between Definitions B.17 and B.18:
  For any action $A = \langle SA, X_1 \ldots X_k \rangle$, define V("active(A)[T_i]") to be TRUE if V("active(*SA*)[T_i]") = TRUE and $V(SA_{P_j}[T_i]) = X_j$ for $j = 1 \ldots k$.

The remainder of the definitions and the proofs of the lemmas remain unchanged. (Nothing in the other definitions or in the proofs depends on the class of actions being finite. The new atoms enter into the preconditions and effect constraints for the symbolic action *SA* in the obvious way.)

References

[1] A. Armando, C. Castellini, E. Giunchiglia, F. Giunchiglia, A. Tacchella, SAT-based decision procedures for automated reasoning: A unifying perspective, in: Lecture Notes in Computer Science, vol. 2605, 2003.
[2] J.F. Allen, Maintaining knowledge about temporal intervals, Comm. ACM 26 (11) (1983) 832–843.
[3] G. Audemard, P. Bertoli, A. Cimatti, R. Kornilowicz, R. Sebastiani, A SAT based approach for solving formulas over Boolean and linear mathematical propositions, in: Proceedings of the International Conference of Automated Deduction, in: Lecture Notes in Artificial Intelligence, vol. 2392, 2002, pp. 193–208.
[4] G. Audemard, M. Bozzano, A. Cimatti, R. Sebastiani, Verifying industrial hybrid systems with MathSAT, Electronic Notes Theoret. Comput. Sci. 89 (4) (2004).
[5] C. Barrett, S. Berezin, CVC Lite: A new implementation of the cooperating validity checker, in: Proceedings of the International Conference on Computer Aided Verification (CAV-04), 2004, pp. 515–518.

[6] R. Bayardo, R. Schrag, Using CSP Look-back techniques to solve real-world SAT instances, in: Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97), Providence, RI, 1997, pp. 203–208.
[7] A. Blum, M. Furst, Fast planning through planning graph analysis, Artificial Intelligence 90 (1997) 281–300.
[8] A. Borning, G. Badros, The cassowary linear arithmetic constraint solving algorithm: Interface and implementation, Technical Report UW-CSE-98-06-04, University of Washington, WA, 1998.
[9] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. Rossum, S. Schults, R. Sebastiani, MATHSAT: Tight integration of SAT and mathematical decision procedures, J. Automat. Reason. (Special Issue on SAT) (2005).
[10] R. Brafman, A simplifier for propositional formulas with many binary clauses, in: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-01), Seattle, WA, 2001, pp. 515–522.
[11] D. Chapman, Planning for conjunctive goals, Artificial Intelligence 32 (3) (1987) 333–377.
[12] E. Davis, Representations of Common Sense Knowledge, Morgan Kaufmann, San Fransisco, CA, 1990.
[13] E. Davis, Axiomatizing qualitative process theory, in: Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR-92), 1992, pp. 177–188.
[14] M. Davis, G. Logemann, D. Loveland, A machine program for theorem proving, Comm. ACM 5 (1962) 394–397.
[15] T. Dean, J. Firby, D. Miller, Hierarchical planning involving deadlines, travel times and resources, Comput. Intelligence 4 (4) (1988) 381–398.
[16] Y. Dimopoulos, A. Gerevini, Temporal planning through mixed integer programming: A preliminary report, in: Proceedings of the 8th Conference on Principle and Practice on Constraint Programming (CP-02), 2002, pp. 47–62.
[17] M.B. Do, S. Kambhampati, Sapa: A scalable multi-objective metric temporal planner, J. Artificial Intelligence Res. 20 (2003) 155–194.
[18] B. Drabble, EXCALIBUR: A program for planning and reasoning with processes, Artificial Intelligence 62 (1993) 1–40.
[19] S. Edelkamp, J. Hoffman, PDDL2.2: The languages for the classical part of the 4th International Planning Competition, Available at http://ipc.icaps-conference.org, 2004.
[20] N. Een, N. Sorensson, An extensible SAT-solver, in: Proceedings of Conference on Theory and Applications of Satisfiability Testing (SAT-03), 2003, pp. 502–518.
[21] M. Ernst, T. Millstein, D. Weld, Automatic SAT-compilation of planning problems, in: Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97), Nagoya, Japan, 1997, pp. 1169–1176.
[22] K. Forbus, Qualitative process theory, Artificial Intelligence 24 (1984) 85–168.
[23] M. Fox, D. Long, The automatic inference of state invariants in TIM, J. Artificial Intelligence Res. 9 (1998) 367–421.
[24] M. Fox, D. Long, PDDL+ Level 5: An extension to PDDL2.1 for modelling planning domains continuous time-dependent effects, Available at http://www.dur.ac.uk/d.p.long/competition.html, 2001.
[25] M. Fox, D. Long, PDDL2.1: An extension to PDDL for expressing temporal planning domains, J. Artificial Intelligence Res. 20 (2003) 61–124.
[26] A. Gerevini, L. Schubert, Inferring state constraints for domain independent planning, in: Proceedings of the 15th National Conference of Artificial Intelligence (AAAI-98), St. Paul, MN, 1998, pp. 905–912.
[27] A. Gerevini, A. Saetti, I. Serina, Planning through stochastic local search and temporal action, J. Artificial Intelligence Res. 20 (2003) 239–290.
[28] E. Giunchiglia, A. Massarotto, R. Sebastiani, Act and the rest will follow: Exploiting determinism in planning as satisfiability, in: Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98), St. Paul, MN, 1998, pp. 948–953.
[29] G. Hendrix, Modeling simultaneous actions and continuous changes, Artificial Intelligence 4 (1973) 145–180.
[30] T. Henzinger, The theory of hybrid automata, in: Proceedings of the 11th Annual Symposium on Logic in Computer Science, 1996, pp. 278–292.
[31] J. Hooker, Logic-Based Methods for Optimization, Wiley, New York, 2000.
[32] H. Kautz, B. Selman, Planning as satisfiability, in: Proceedings of the 10th European Conference on Artificial Intelligence (ECAI-92), 1992, pp. 359–363.

[33] H. Kautz, D. McAllester, B. Selman, Encoding plans in propositional logic, in: Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning (KR-96), 1996, pp. 374–384.
[34] H. Kautz, B. Selman, Pushing the envelope: Planning, propositional logic and stochastic search, in: Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96), Portland, OR, 1996, pp. 1194–1201.
[35] H. Kautz, B. Selman, Unifying SAT-based and graph-based planning, in: Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-99), Stockholm, Sweden, 1999, pp. 318–325.
[36] K. Kichkaylo, A. Ivan, V. Karamcheti, Constrained component deployment in wide-area networks using AI planning techniques, in: Proceedings of the International Parallel and Distributed Symposium (IPDPS-03), 2003, pp. 3–8.
[37] D. Long, M. Fox, I. Sebastia, A. Coddington, An examination of resources in planning, in: Proceedings of UK Planning and Scheduling SIG Workshop, 2000.
[38] D. Long, M. Fox, Exploiting a Graphplan framework in temporal planning, in: Proceedings of International Conference on Automated Planning and Scheduling (ICAPS-03), 2003, pp. 51–62.
[39] D. Long, M. Fox, The 3rd international planning competition: Results and analysis, J. Artificial Intelligence Res. 20 (2003) 1–59.
[40] A. Mali, Encoding temporal planning as CSP, in: Proceedings of IEEE International Conference on Tools with Artificial Intelligence, 2002, pp. 75–92.
[41] D. McDermott, The AIPS-98 planning competition committee, PDDL—the planning domain definition language, Version 1.2, Available at http://www.cs.yale.edu/homes/dvm, 1998.
[42] D. McDermott, The formal semantics of processes in PDDL, in: Proceedings of Workshop on PDDL at International Conference on Automated Planning Scheduling, 2003.
[43] D. McDermott, Reasoning about autonomous processes in an estimated-regression planner, in: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-03), 2003, pp. 143–152.
[44] J. Penberthy, Planning with continuous change, Ph.D. Dissertation, Department of Computer Science and Engineering, University of Washington, WA, USA, 1993.
[45] J. Penberthy, D. Weld, Temporal planning with continuous change, in: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94), Seattle, WA, 1994, pp. 1010–1015.
[46] J. Shin, TM-LPSAT: Encoding temporal metric planning in continuous time, Ph.D. Dissertation, Department of Computer Science, New York University, NY, USA, 2004.
[47] J. Shin, E. Davis, Continuous time in a SAT-based planner, in: Proceedings of the 22th National Conference on Artificial Intelligence (AAAI-04), San Jase, CA, 2004, pp. 531–536.
[48] R. Simmons, Combining associational and causal reasoning to solve interpretation and planning problems, Technical Report AI-TR-1048, MIT AI Lab, MA, USA, 1988.
[49] D. Smith, J. Frank, A. Jonsson, Bridging the gap between planning and scheduling, Knowledge Engrg. Rev. 15 (1) (2000) 61–94.
[50] D. Smith, D. Weld, Temporal planning with mutual exclusion reasoning, in: Proceedings of the 16th International Joint Conference of Artificial Intelligence (IJCAI-99), Stockholm, Sweden, 1999, pp. 326–333.
[51] S. Vere, Planning in time: Windows and durations for activities and goals, Pattern Anal. Machine Intelligence 5 (1983) 246–267.
[52] D. Wilkins, Can AI planners solve practical problems?, Comput. Intelligence 6 (4) (1990) 232–246.
[53] S. Wolfman, D. Weld, The LPSAT engine and its application to resource planning, in: Proceedings of the 16th International Joint Conference of Artificial Intelligence (IJCAI-99), Stockholm, Sweden, 1999, pp. 310–316.
[54] S. Wolfman, D. Weld, Combining linear programming and satisfiability solving for resource planning, Knowledge Engrg. Rev. 16 (1) (2000) 85–99.
[55] L. Zhang, S. Malik, The quest for efficient boolean satisfiability solvers, in: Proceedings of the International Conference on Computer Aided Verification (CAV-02), 2002, pp. 17–36.