Continuous Time in a SAT-based Planner

Ji-Ae Shin and **Ernest Davis**[*]
Courant Institute
New York University
{ jiae | davise }@cs.nyu.edu

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; 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 propositional combinations of propositional variables and linear constraints over numeric variables. In the second stage, the LPSAT constraint engine (Wolfman & Weld 2000) is used to find a solution to the system of constraints. In the third stage, a correct parallel plan is extracted from this solution. We discuss the structure of the planner and show how a real-time temporal model is compiled into LPSAT constraints.

Keywords: Propositional planning, LPSAT, continuous time, numerical quantities, processes.

Introduction

Over the past decade, several new powerful engines for propositional satisfiability have become available and are now being used in a broad range of applications (Alessandro et al. 2002). One very successful application has been the development of *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 (Kautz & Selman 1992; Kautz & Selman 1996; Ernst et al. 1997; Kautz & Selman 1999) .

Recently, a new class of inference engines (Wolfman & Weld 1999; Audemard et al. 2002; Barrett & Berezin 2004) has been developed for systems of propositional combinations of linear constraints over real-valued quantities. In this paper, we show that these can be used to extend propositional planning to many domains that involve real-valued quantities and continuous change.

The TM-LPSAT planner (Shin 2004) 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. These can have the same types of preconditions and effects as atomic actions.
- Autonomous processes can be defined in the language.
- Reusable resources,both multiple-capacity and interval-valued, can be defined in the language.

TM-LPSAT has three modules. The *compiler* takes as input a domain and a problem description in PDDL+ (Fox & Long 2001; 2003) and outputs a set of constraints over propositional and numerical variables. The *LPSAT* constraint engine, developed by Wolfman and Weld (1999) , takes this set of constraints as input and outputs a solution, if one exists, in the form of an assignment to the variables. The *decoder* takes as input the assignment and outputs a correct plan. The overall system is thus a powerful and elegant planner for a wide range of problems.

In this paper[1], we present the encoding of real-time temporal model defined in PDDL+ Level 5 (Fox & Long 2001), consisting of atomic actions, exogenous events and autonomous processes.

Foundations

The TM-LPSAT planner builds on three foundations: the theory of propositional planning, the LPSAT constraint engine, and the PDDL+ specification language.

Propositional planning

In propositional planning, a planning problem in a domain with discrete actions and fluents[2] with discrete values is con-

[*]The research reported in this paper was supported in part by NSF grant IIS-0097537.

[1]We are preparing a paper that deals with the details of the encoding of a durative action, as defined in PDDL+ Levels 3 and 4. Due to space limitations, we do not discuss intervals or reusable resources here.

[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" such as "on(blockA,blockB)", rather than meaning the particular PDDL construct of that name. Temporal logic "fluents" includes PDDL "predicates".

verted into a set of propositional constraints. This is done as follows:

- An upper bound N is guessed for the maximum number of steps in the plan. Time points are labelled 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.
- 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 problem is specified by stating that the start state holds at time 0 and that the goal state holds at time N.
- The constraints are fed to a propositional satisfiability engine. If the constraints can be solved, then the actions that are marked as occurring constitute a valid plan.

Propositional planners can be implemented easily and, with the current generation of satisfiability engines, quite effectively. Since their introduction in (Kautz & Selman 1992), they have been extensively studied (Kautz & Selman 1996; Kautz & Selman 1999), and various ways of formulating the constraints (Kautz et al. 1996; Ernst et al. 1997) have been compared. The major drawback 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.

The LPSAT program

The LPSAT constraint engine is composed from RelSAT a DPLL-based propositional solver, and Cassowary an incremental linear constraint solver. The input to LPSAT (Wolfman & Weld 1999) 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 program first looks for a propositional solution, treating each linear equation as a propositional atom, then tries to solve the set of equations that have been marked as true. If that set is inconsistent, then LPSAT adds a clause stating that these linear inequalities are not all true, and it looks for a new propositional solution. It goes back and forth between propositional and numerical mode until either finding a solution or establishing that no solution exists.

PDDL+

PDDL (Planning Domain Definition Language) (McDermott 1998; 2000) is a declarative language for the definition of causal domains and planning problems. The basis of our work is PDDL+, which is the most recent extension to PDDL. PDDL+ comprises five levels. Level 1 contains discrete actions and fluents. Level 2 adds features 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 (Fox & Long 2003).) Level 5 (Fox & Long 2001), proposed but not approved by IPC (International Planning Competition) committee, is a real-time temporal model that includes atomic actions, exogenous events, and autonomous processes.

The input specification language for TM-LPSAT extends PDDL+ in that we allow actions to have real-valued parameters. For instance, there can be an action "scoop(N, C, B)" of scooping up N cups of flour from bin B into cup C. PDDL2.1 excludes this (Fox & Long 2003) but their arguments do not strike us as cogent. Numerical parameters greatly increase the expressivity power of the language, and, in the LPSAT approach, impose no additional computational burden.

One restriction must be imposed on actions with numeric parameters: There cannot be two or more concurrent actions with the identical non-numeric parameters. For instance, we do not allow the actions "scoop(5,c1,b1)" and "scoop(2,c1,b1)" to be executed concurrently, though "scoop(5,c1,b1)" and "scoop(2,c2,b2)" may be concurrent. The restriction is reasonable because such numerical parameters are typically used in one of two ways. If the value of the parameter is assigned to a fluent – e.g. "scoop(N, C, B)" results in the cup containing N cups of flour — 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. "scoop(N, C, B)" results in adding N cups of flour to cup C — then two such concurrent actions scoop(5,c1,b1) and scoop(2,c1,b1) can be combined into a single action scoop(7,c1,b1). We have not found any cases where it would be reasonable and important to violate this restriction.

By virtue of this restriction, an action type is identified by the name of the functor and the non-numeric arguments. For example, we may speak of the action type "scoop(·,b1,c1)" (scooping some amount from b1 into c1) and be sure that at most one of these occurs at one time.

A few features of PDDL+ cannot be included in TM-LPSAT. First, TM-LPSAT cannot optimize a specified plan metric. 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 that LPSAT cannot handle. Third, no non-linear expressions are allowed in any definitions. All other features of PDDL+ are included.

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 numerical, they may undergo continuous change as a linear function of time. Every discontinuous change, or change in the derivative of a numerical fluent, occurs at a significant time point. Thus, there are two states associated with each time point T; the state *before* T and the state *after* T.

Each time point has a *clock time*, which is a non-negative real value. These clock times become numerical variables in the system of constraints set up by the compiler. It is possible, however, to have two distinct time points with equal clock times. This is used to accommodate the following situation: Suppose that action A is executed at time T_i, and has the effect of either causing fluent P to be true or making a discrete change to a numerical fluent. Suppose further that event E has triggering condition P or a numeric constraint satisfied by the discrete change. Then E should be executed immediately after A. We model that by positing that E is executed at time T_{i+1}, but that the clock time for T_{i+1} is the same as the clock time for T_i. Note that 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.

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 event *must* occur if its precondition hold. For example, the event "flood" has a precondition that the tub is full, the bathroom floor is dry, and that there is a process of inflow into the tub. It has the consequence that the bathroom floor is wet.

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 1984), *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 "fillBath(B - bath; T - tap)" has the precondition that tap T is open and that the level of the bath is less than its capacity; and the constraint that T is a tap of B. (We assume, unrealistically, that flow will stop once the bath is full.) It has the effect of increasing level(B) at the rate flow(T). (We allow only taps that are fully on or off.) At any moment, the derivative of level(B) is the sum of the rates of the active fillBath processes.

PDDL+ permits concurrent actions under fairly restrictive conditions, designed to ensure that the actions do not interact, either destructively or synergistically. The actual condition imposed is one that can be computed easily and statically and that is sufficient, though not necessary, to ensure this non-interaction.

Sample domain

The bath domain includes filling processes with different taps, a draining process, a flood event, and atomic actions of tapOn, tapOff, plugIn, plugOut, and addBubble. Filling processes can be concurrent with draining process. We can add bubble before the bath is half full. Thus, this domain can show concurrent continuous and discrete changes on the level of a bath.

```
(:process fillBath
 :parameters (?b - bath ?t - tap)
 :precondition (and (tap_on ?b ?t)
                    (<= (level ?b) (capacity ?b)))
 :effect (increase (level ?b) (* #t (flow ?b ?t))))
(:process drainBath
 :parameters (?b - bath)
 :precondition (and (not (plug_in ?b))
                    (> (level ?b) 0))
 :effect (decrease (level ?b) (* #t (draining_flow ?b)))
(:event flood
 :parameters (?b - bath)
 :precondition (and (exists (?t - tap) (tap_on ?b ?t))
                    (>= (level ?b) (capacity ?b))
                    (dry_floor ?b))
 :effect (not (dry_floor ?b)))
(:action addBubble
 :parameters (?b - bath ?q - real)
 :condition (and (exists (?t - tap) (tap_on ?b ?t))
                 (not (bubble_added ?b))
                 (<= (level ?b) (/ (capacity ?b) 2)))
 :effect (and (bubble_added ?b)
              (increase (level ?b) ?q)))
```

Compilation to LPSAT constraints

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 $T_0 \dots T_N$.

We define the following propositional atoms:

- For each time T_i, for each propositional 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. (Note: An action/event is identified by the name of the action and the value of the non-numeric parameters, as discussed above.)
- For each time T_i, for each process[3] P, the assertion that P is active at T_i. (This includes the time that P starts but not the time when it terminates.)

We define the following numeric variables:

- The clock time of every significant time point T_i. We will denote this "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.

[3]The encoding assumes that two instances of the same process cannot be concurrent.

- 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 $\Delta(A, Q, T_i)$.
- For each numeric fluent Q, for each durative action or process A that changes Q continually, for each time T_i, the net change in Q due to A between T_i and T_{i+1}. This is denoted $\Gamma(A, Q, T_i, T_{i+1})$.
- Let $A(P_1 \ldots P_k, Q_1 \ldots Q_m)$ be an action where $P_1 \ldots P_k$ are discrete parameters and $Q_1 \ldots Q_m$ are numeric parameters. Then, by the restriction mentioned above, 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 for which an action of the form $A(P_1 \ldots P_k, Q_1 \ldots Q_m)$ begins at time T_i. The value of each such Q_j before and after T_i is a numeric variable.

We will use the following convention for labelling time-dependent terms:

- If a complex term α over fluents that appears in a precondition or on the right side of an assignment statement 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 from T_{i-1} and the values of numeric fluents from *before* T_i.
- If a complex term α over fluents that appears in a precondition or effect 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 using the values of propositional fluents from T_i and the value of numeric fluents from *after* T_i.

A domain definition then gives rise to the following kinds of constraints (see (Shin 2004) for details):

1. Atomic actions
1.1: Effects:
1.1.1: If an effect of action A is to assign term α to discrete or interval fluent F, then add the constraint
active(A, T_i) $\Rightarrow F[T_i] = \alpha[T_i^-]$.
1.1.2: If an effect of action A is to assign term α to numeric fluent F, then add the constraint
active(A, T_i) $\Rightarrow F[T_i^+] = \alpha[T_i^-]$.
1.1.3: If an effect of action A is to increase numeric fluent Q by the term α, then add the constraint
active(A, T_i) $\Rightarrow \Delta[A, Q, T_i] = \alpha[T_i^-]$.
$\neg$active(A, T_i) $\Rightarrow \Delta[A, Q, T_i] = 0$.
1.1.4: 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 constraints:
$\neg$active(E_1, T_i) $\wedge \ldots \wedge$ $\neg$active(E_p, T_i) $\Rightarrow$
$Q[T_i^+] = Q[T_i^-] + \sum_j \Delta[A_j, Q, T_i]$.
(Note that the terms in this sum can be determined statically.)
1.1.5: 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.
1.2: Preconditions: If action A has preconditions β, then add the constraint: active(A, T_i) $\Rightarrow \beta[T_i^-]$.
1.3: Mutual exclusion: If action A is mutually exclusive (mutex) with action or event E then add the constraint
active(A, T_i) $\Rightarrow \neg$active(E, T_i).
The rules for mutual exclusion are complex, but statically determined (Fox & Long 2003; Shin 2004).
2. Events
2.1: Effects: Same as for atomic actions.
2.2: Preconditions: Let β be the preconditions of event E. Add the constraint: active(E, T_i) $\Leftrightarrow \beta[T_i^-]$
2.3: Immediate occurrence of events. Let β be the preconditions of event E. Add the constraint:
$\beta[T_i^+] \Rightarrow c(T_{i+1}) = c(T_i)$.
PDDL+ requires an event to have at least one numerical constraint in its preconditions. This constraint ensures that the event is triggered with no time slip by discrete changes on numeric fluents.
3. Processes
3.1: Effects.
3.1.1: For each process P, for each quantity Q influenced by P, let Φ be the influence of P on the derivative of Q. Add the constraints
active(P, T_i) $\Rightarrow \Gamma(P, Q, T_i, T_{i+1}) = \Phi \cdot (c(T_{i+1}) - c(T_i))$.
$\neg$active(P, T_i) $\Rightarrow \Gamma(P, Q, T_i, T_{i+1}) = 0$.
3.1.2: For each quantity Q, let $P_1 \ldots P_m$ be the processes that potentially affect Q. Add the constraint
$Q[T_{i+1}^-] = Q[T_i^+] + \sum_j \Gamma(P_j, Q, T_i, T_{i+1})$.
3.2: Preconditions: Let β be the preconditions for process P. Add the constraint
active(P, T_i) $\Leftrightarrow \beta[T_i^-] \vee \beta[T_i^+]$.
A process can be triggered either by continuous changes or by discrete changes.
4. Frame Axioms
4.1: Propositional 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
$F[T_{i-1}] = V \wedge$ $\neg$active(A_1, T_i) $\wedge \ldots \wedge$ $\neg$active(A_k, T_i)
$\Rightarrow F[T_i] = V$.
4.2: Numerical fluents: No additional frame axioms are needed. If no processes that change F are continuing between T_i and T_{i+1}, then all the terms in the sum in equation 3.1.2 will be 0, so the equation will state that the quantity does not change. Likewise, if no actions or events occur that change F at time T_i, then all the terms in the sum in equation 1.1.3 will be 0.
5. Order on Significant Time-Points
For each T_i, add the constraint $c(T_{i+1}) \geq c(T_i)$.
6. Zero crossings
One last type of constraint is trickier. This has to do with an event or process being triggered or terminated by a continuously changing numerical fluent attaining a particular value. 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; at time T_a Q is less than V1 and at time T_b Q is greater than V2. 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 termination of processes; we must make sure that they stop as soon as their continuation condition becomes false.

The solution rests on the fact that all numeric conditions are Boolean combinations of linear constraints, and that, within our domains, any numeric parameter that changes continuously is a linear function of time. Therefore, we can proceed as follows. First, we put every such condition that has to be checked over an interval into disjunctive normal form; that is, we express it as the disjunction of a collection of conjuncts. Further, we may assume that every numerical constraint has the form $Q(t) \geq 0$ where $Q(t)$ is a linear function of the numerical variables and of time t. Now, consider any such constraint $F_1 \wedge \ldots \wedge F_k \wedge Q_1 \geq 0 \wedge \ldots \wedge Q_m \geq 0$, where the F_i are propositional expressions and the Q_i are linear functions. The values of the F_i do not change between two consecutive significant time points. (The F_i will include conditions on the activity of processes and durative actions that affect the numeric quantities involved.) Depending on the context of the condition, we either have a case (such as the condition of an active process) where the conjunct is true at T_i and we are interested in the first time when any of the numeric conjuncts becomes false; or a case (such as the precondition of an event) when the conjunct is false at T_i and we are interested in the time when the last remaining numeric conjunct becomes true. Let us consider the second case; the first case is essentially a dual construction. Note that, at the time when $Q_j \geq 0$ becomes true, $Q_j = 0$ by continuity. Thus the time when $Q_j = 0$ is no earlier than T_{i+1}; thus $Q_j[T^-_{i+1}] \leq 0$. We therefore state the following constraints: For each significant time point T_i, for each such conjunction, for each conjunct Q_j, assert

$$[\wedge_p F_p[T_i]] \wedge Q_j[T^+_i] < 0 \wedge [\wedge_{p \neq j} Q_p[T^-_{i+1}] \geq 0] \Rightarrow Q_j[T^-_{i+1}] \leq 0.$$

This is just a continuity constraint over Q_j of a form familiar from qualitative process theory (Forbus 1984); however, we are applying it to the complex terms Q_j and restricting its application to those cases where it might make a difference in terms of triggering an event or process.

The effect of these constraint 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.

Experimental Results[4]

There is no standard benchmark domain with continuous changes to use for performance comparison. We have some experimental results with toy domains. The domain used in Table 1 is a variation of the sample domain shown before. By distinguishing hot taps and cold taps, the goal includes constraints on the range of ratio of flows from hot taps to flows from cold taps, so that the desired bath temperature is achieved.

[4]TM-LPSAT may be compared with McDermott's Optop; however, Optop is not available for comparison at this point.

Since TM-LPSAT cannot optimize plans, the plan returned may have any number of steps, as long as the goal is satisfied. Table 1 shows the encoding size and processing times as the step increases.

Except pruning of the search space in a Graphplan style and Crawford's Compaction procedure, no other optimization is done with the current implementation of TM-LPSAT compiler. The times are measured on Linux 1.8 GHz Pentium IV processor with 512 MB RAM, averaged over 20 runs. The LPSAT engine is set to the option of learning and conflict-driven backjumping.

No. of Steps	5	10	15
Total Time	14.58	184.49	589.98
Compaction Time	0.09	0.4	0.94
Solving Time	14 ± 4	184 ± 85	582 ± 225
Initial Statistics:			
No. of clauses	339	769	1199
No. of Boolean variables	236	496	756
No. of linear constraints	136	296	456
No. of real variables	65	130	195
After Compaction:			
No. of clauses	329	724	1119
No. of Boolean variables	190	405	620
No. of linear constraints	121	256	391
No. of real variables	65	130	195
No. of ground operators	46 (59)	101 (124)	156 (189)

Table 1: Filling the bath with multiple taps. Processing times are measured in seconds. The number inside of parenthesis in the last row shows the number of ground operators before pruning.

Related Work

Obviously, TM-LPSAT is based on LPSAT (Wolfman & Weld 1999). The LPSAT planner both developed the constraint engine that we have used, and applied it to the metric planning in discrete time. The encoding of discrete changes on numeric fluents adopted in TM-LPSAT is almost the same as their encoding.

Among a number of early partial-order planners dealing with continuous changes to a limited extent, ZENO (Penberthy & Weld 1994) permitted a very general plan specification language, though its models of concurrency and of processes were less general than TM-LPSAT. Also, it was extremely slow; (Wolfman & Weld 2000) reports that ZENO was unable to solve even the simplest of the logistic problems that were used to test LPSAT.

McDermott (2003) has extended his estimated-regression planner, Optop, 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.

The best known study of processes in the AI literature is QP theory (Forbus 1984), 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 (1992) gives a logical analysis of QP theory.

Conclusions and Future Work

As far as we know, TM-LPSAT is the first SAT-based planner that can reason about exogenous events and autonomous processes that cause continuous changes. It generates a parallel plan dealing with concurrent continuous and discrete changes. Preliminary experimentation on toy domains is reasonably encouraging this can be an effective framework for planning in continuous time over a wide range of problems.

In continuing this research, we hope to:

- Optimize both the process of compilation and the constraints output by compilation.
- Test the effectiveness of some engines other than LPSAT in solving the kinds of constraint systems generated by our compiler. MathSAT (Audemard et al. 2002) and CVCLite (Barrett & Berezin 2004) are both engines that can find solutions to propositional combinations of arithmetic constraints, so either of these could be used for this purpose.
- Study whether our methods for dealing with continuous change can be applied in the planning architectures based on Graphplan (Blum & Furst 1997).
- Add spatial reasoning by allowing region-valued fluents and motion as a process. If regions are restricted the 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.

Acknowledgments

We would like to thank Steven Wolfman and Daniel Weld for making their LPSAT program available for our research.

References

Alessandro, A., Castellini, C., Giunchiglia, E., Giunchiglia, F., and Tacchella, A. 2002. SAT-Based Decision Procedures for Automated Reasoning: a Unifying Perspective. *Journal of Automated Reasoning* **28**.

Audemard, G., Bertoli, P., Cimatti, A., Kornilowicz, A., and Sebastiani, R. 2002. A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions. In *Proceedings of the International Conference of Automated Deduction*.

Barrett, C., and Berezin, S. 2004. CVC Lite: A New Implementation of the Cooperating Validity Checker. In *Proceedings of the International Conference on Computer Aided Verification*.

Blum, A. and Furst, M. 1997. Fast Planning through Planning Graph Analysis. *Artificial Intelligence* **97**.

Davis, E. 1992. Axiomatizing Qualitative Process Theory. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*.

Ernst, M., Millstein, T., and Weld, D. 1997. Automatic SAT-Compilation of Planning Problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Forbus, K. 1984. Qualitative Process Theory. *Artificial Intelligence* **24**.

Fox, M., and Long, D. 2001. 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.

Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research* **20**.

Kautz, H., and Selman, B, 1992. Planning as Satisfiability. In *Proceedings of the European Conference on Artificial Intelligence*.

Kautz, H., McAllester, D., and Selman, B. 1996. Encoding Plans in Propositional Logic. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*.

Kautz, H., and Selman, B. 1996. Pushing the Envelope: Planning, Propositional Logic and Stochastic Search. In *Proceedings of the National Conference on Artificial Intelligence*.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and Graph-based Planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

McDermott, D. 1998. PDDL - the Planning Domain Definition Language. Available at http://www.cs.yale.edu/homes/dvm.

McDermott, D. 2000. The 1998 AI Planning Systems Competition. *AI Magazine* **21**.

McDermott, D. 2003. Reasoning about Autonomous Processes in an Estimated-Regression Planner. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Penberthy, J., and Weld, D. 1994. Temporal Planning with Continuous Change. In *Proceedings of the National Conference on Artificial Intelligence*.

Shin, J. 2004. TM-LPSAT: Encoding Temporal Metric Planning in Continuous Time. Ph.D. Dissertation, Dept. of Computer Science, New York University.

Wolfman, S., and Weld, D. 1999. The LPSAT Engine and its application to Resource Planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Wolfman, S., and Weld, D. 2000. Combining Linear Programming and Satisfiability Solving for Resource Planning. *Knowledge Engineering Review* **15**.